A LARGE LANGUAGE MODEL-BASED FRAMEWORK FOR ENHANCING PERSONALIZED LEARNING OF C PROGRAMMING

Fredrick Muema Mboya

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Information Technology of Murang'a University of Technology

July, 2025

DECLARATION

Student Declaration

I hereby declare that this Thesis is my original work and to the best of my knowledge has not been presented for a degree award in this or any other university. I understand what plagiarism is and I am aware of the University's policy in this regard. I understand that any false claim in respect of this work shall result in disciplinary action in accordance with the University Anti-Plagiarism Policy.

_____ _____

Fredrick Muema Mboya Date

SC401/5825/2022

Supervisors' Approval

The undersigned certify that they have read and hereby recommended for acceptance of Murang'a University of Technology a thesis entitled "**A Large Language Model-Based Framework for Enhancing Personalized Learning of C Programming.**"

_____ _____

Dr. Geoffrey Mariga Wambugu, Date

Department of Information Technology,

Murang'a University of Technology.

_____ _____

Dr. Aaron Mogeni Oirere, Date

Department of Computer Science,

Murang'a University of Technology.

DEDICATION

I dedicate this work to my family, whose unwavering support, love, and encouragement have been my greatest source of strength throughout this journey. They have instilled in me the value of education and perseverance, and to my siblings, for always believing in my potential. To my supervisors and the faculty, whose guidance and wisdom have shaped my academic and professional growth, I am deeply grateful. Above all, I dedicate this thesis to all aspiring scholars and innovators in the field of artificial intelligence. May this work inspire and contribute to the advancement of knowledge in personalized learning and technology-driven education.

ACKNOWLEDGEMENT

First and foremost, I extend my deepest gratitude to God for granting me the strength, wisdom, and resilience to complete this research. His guidance and grace have been my constant source of inspiration throughout this academic journey. I sincerely appreciate my supervisors, Dr. Geoffrey Mariga Wambugu and Dr. Aaron Mogeni Oirere, for their invaluable guidance, insightful feedback, and unwavering support. Their expertise and mentorship have assisted in shaping this research, and I am truly grateful for their encouragement.

I also wish to express my sincere appreciation to the entire faculty of the School of Computing and Information Technology at Murang'a University of Technology. Their dedication to academic excellence, continuous support, and provision of a nurturing learning environment have significantly contributed to my academic growth. The knowledge, skills, and inspiration I have gained from the faculty members have been instrumental in the successful completion of this thesis.

I extend my heartfelt gratitude to KENET for their financial support through the research grant. The support significantly facilitated the successful completion of this study, enabling me to focus on my research objectives without financial constraints. I am also immensely grateful to Dr. Eric Odhiambo Omuya, Dr. Faith Mueni Musyoka, and Dr. Joyce Wangui Gikandi for their academic insights, encouragement, and constructive feedback. Their valuable contributions have greatly enriched this work, and I deeply appreciate their time and dedication.

Finally, I acknowledge my family, friends, and colleagues who have supported me throughout this research journey. Their encouragement, patience, and understanding have been instrumental in keeping me motivated and focused. To all who have contributed to the success of this work in one way or another, I extend my sincere appreciation.

ABSTRACT

The increasing demand for competent programmers has amplified the need for instructional approaches that are both effective and personalized. Traditional programming education methods often fall short in offering real-time feedback, adaptive task difficulty, and contextual support, challenges that limit learner engagement and hinder skill acquisition. While recent advances in Generative Artificial Intelligence (GenAI), particularly through Large Language Models (LLMs), offer potential to address these issues, existing educational frameworks rarely integrate such tools in a way that ensures accuracy, coherence, adaptability, and personalized feedback. This study addresses this gap by developing and evaluating a framework that operationalizes GenAI for personalized C programming instruction among undergraduate students. A systematic literature review was conducted to identify key features essential for effective AI-enhanced learning, including adaptability, contextual relevance, coherence, accuracy, prompt engineering, personalized feedback, creativity, and usability. These features were validated through expert opinion and used to guide the development of a web-based learning framework. The framework integrates four key components: an LLM interaction module (Node.js/Express.js), an adaptive learning engine, a collaborative coding platform, and a React.js interface to deliver real-time, learner-specific support in C programming. Architecture design followed agile prototyping cycles with iterative expert feedback. To evaluate effectiveness, a mixed-method study was conducted with 139 undergraduate programming students at Murang'a University of Technology. A purposive sample of 75 formed the experimental group and engaged with the GenAI-powered system over two weeks, while 64 students in the control group received conventional instruction. Data collection involved structured questionnaires, open-ended surveys, and a pretest-posttest design aligned to core programming concepts. Performance was assessed through gain scores in pretest-posttest evaluations, mean ratings on personalization features, and learner feedback on usability and engagement. The findings showed high student satisfaction, with a mean of 4.13 for usability and 4.14 for perceived learning enhancement. A pretest-posttest comparison revealed a mean gain of 9.08 points in the experimental group, versus 2.14 in the control group. A Mann-Whitney U test confirmed this difference was statistically significant ($p = 0.008$). This had a real impact, as more than half of the students (55%) said the framework helped them become better at solving programming problems. These results demonstrate that LLM-based tools can enhance personalized programming education and have strong potential for industry use in developer training and reskilling. The study recommends further refinement and ethical integration in academic settings. Cross-sector collaboration is also encouraged to support responsible, scalable adoption.

TABLE OF CONTENTS

DECLARATION ..... **ii**
DEDICATION ..... **iii**
ACKNOWLEDGEMENT ..... **iv**
ABSTRACT ..... **v**
LIST OF TABLES ..... **x**
LIST OF FIGURES ..... **xi**
ACRONYMS AND ABBREVIATIONS ..... **xii**
OPERATIONAL DEFINITION OF TERMS ..... **xiv**
CHAPTER ONE: INTRODUCTION **1**
1.1 Background of the Study 1
1.2 Problem Statement 4
1.3 Research Objectives 5
1.3.1 General Objective 5
1.3.2 Specific Objectives 6
1.4 Research Questions 6
1.5 Scope of the Study 6
1.6 Significance of the Study 7
1.7 Limitations of the Study 8
1.8 Contributions of the Thesis 8
1.9 Organization of the Thesis 9
CHAPTER TWO: LITERATURE REVIEW **11**
2.1 Introduction 11
2.2 Theoretical Foundations for Personalized Instruction 11
2.2.1 Cognitive Theories Supporting Personalized Learning 11
2.2.2 Pedagogical Theories Supporting Personalized Learning 12
2.2.3 LLMs in Educational Frameworks 15
2.2.4 Features & Best Practices from Existing Tools 20
2.2.5 Integrating Technology in Education 22
2.3 Theoretical Framework 25
2.3.1 Existing GenAI Application Frameworks 26
2.3.2 Overview of GenAI in Education 26
2.3.3 AI-Driven Tutoring Frameworks for Programming Education 27
2.3.4 AI Tutoring Systems vs. LLM-Based Tutors 29

2.3.5 Explainability and Trust in LLM-Based Tutors....31
2.4 Empirical Literature....32
2.4.1 Learning Requirements in Computer Programming....32
2.4.2 Integration of Technology in Programming Education....32
2.4.3 Generative Artificial Intelligence....33
2.4.4 Empirical Insights on Personalization....36
2.5 Case Studies of AI-Assisted Programming Education....38
2.5.1 AI-Assisted Code Generation for Novice Coders....38
2.5.2 AI-Generated Hints Versus Human Guidance....38
2.5.3 AI-Generated Explanations for Programming Errors....39
2.5.4 Adaptive Explanations Using Prompt Engineering....39
2.5.5 AI-Generated Content for Programming Instruction....39
2.5.6 AI-Assisted Code Review and Debugging Tools....40
2.6 Analysis of Key Features and Best Practices....40
2.6.1 Adaptability....40
2.6.2 Contextual Relevance....41
2.6.3 Clarity and Coherence....42
2.6.4 Accuracy and Reliability....43
2.6.5 Prompt Engineering Effectiveness....44
2.6.6 Practical Usability....44
2.6.7 Interactivity and Collaboration....45
2.6.8 Creativity and Uniqueness....46
2.7 Gap Identification....47
2.8 Conceptual Framework....48
2.9 Summary....50
CHAPTER THREE: METHODOLOGY....51
3.1 Introduction....51
3.2 Research Design....51
3.3 Research Process....53
3.4 Participants and Data Collection Procedures....55
3.4.1 Target Population....55
3.4.2 Sample Size and Sampling Strategy....57
3.4.3 Data Acquisition....59
3.4.4 Data Analysis....61
3.5 Systematic Analysis of Features....61
3.5.1 Data Sources and Search Strategy....62
3.5.2 Inclusion Criteria....62

3.5.3 Exclusion Criteria 63
3.5.4 Study Selection 63
3.5.5 Data Extraction and Synthesis 64
3.6 Developing the LLM-Based Learning Framework 65
3.6.1 Feature Mapping and Architecture Design 65
3.6.2 Prompt Engineering and Dialogue Flow Design 67
3.6.3 Collaborative Coding and Real-Time Interaction 69
3.6.4 Testing and Deployment 70
3.7 Framework Performance Evaluation 71
3.8 Experimental Materials 73
3.9 Controlled User Studies 74
3.10 Ethical Considerations 75
3.11 Summary 76
CHAPTER FOUR: RESULTS AND DISCUSSION 77
4.1 Introduction 77
4.2 Feature Analysis 77
4.2.1 Overview of Literature Search and Study Selection 78
4.2.2 Thematic Synthesis of Key Features 79
4.2.3 Expert Validation 80
4.3 Framework Development 86
4.3.1 Design Principles and Component-Feature Alignment 87
4.3.2 Architecture of the Developed Framework 88
4.3.3 Core Functional Components 92
4.3.4 End-to-End Framework Workflow 97
4.4 Evaluation of the Developed Framework 100
4.4.1 Survey Design and Administration 100
4.4.2 Quantitative Analysis of Personalization Features 100
4.4.3 Thematic Analysis of Qualitative Feedback 102
4.4.4 Pretest-Posttest Comparison 104
4.4.5 Framework Behavior in Personalized Code Generation 105
4.4.6 Comparative Framework Analysis 107
4.5 Results 109
4.5.1 Expert Survey Results 109
4.5.2 Experimental Results 113
4.6 Discussion 115
4.6.1 Implication of Expert Review Results 115
4.6.2 Implication of Experimental Results 117

4.7 Summary 120
CHAPTER FIVE: CONCLUSION, RECOMMENDATIONS AND FUTURE WORK 122
5.1 Conclusion 122
5.2 Recommendations 123
5.2.1 Recommendations for Practice 123
5.2.2 Recommendation for Policy 124
5.2.3 Recommendation for Industry 124
5.3 Future Work 124
REFERENCES 126
APPENDICES 143

LIST OF TABLES

Table 2.1: Examples of AI-Driven Frameworks for Programming 28
Table 2.2: AI Tutoring Systems vs. LLM-Based Tutors 30
Table 3.1: Summary of Research Objectives 55
Table 3.2: Experimental Materials 73
Table 4.1: Frequency of Personalization Features Identified 80
Table 4.2: Demographic Summary of Student Respondents (n = 12) 82
Table 4.3: Demographic Summary of Professional Expert Respondents (n = 8) 83
Table 4.4: Mean Ratings and Standard Deviations of GenAI Capabilities 84
Table 4.5: Framework Components and Their Supported Features 88
Table 4.6: Student Ratings on Personalization Aspects (n = 75) 101
Table 4.7: Group Performance on Pretest and Posttest Evaluations 104
Table 4.8: Comparison of the Developed Framework with Existing Frameworks . 108
Table 4.9: Mann-Whitney U Test Results 115

LIST OF FIGURES

Figure 2.1: Self-Determination Theory (SDT) by Deci and Ryan 13
Figure 2.2: Problem-based Learning 14
Figure 2.3: Annotated Transformer Architecture 16
Figure 2.4: RAG Architecture 20
Figure 2.5: The TPACK framework 23
Figure 2.6: The Process of Generating Programming Exercises 34
Figure 2.7: Conceptual Framework 50
Figure 3.1: Research Process 54
Figure 3.2: Architectural Data Flow of the Framework 67
Figure 3.3: Prompt Adaptation Workflow 69
Figure 4.1: PRISMA Diagram of Literature Selection Process 78
Figure 4.2: Perceived Effectiveness of GenAI Features 85
Figure 4.3: Architecture and Data Flow in the Developed Framework 90
Figure 4.4: Lifecycle of the Programming Exercises 91
Figure 4.5: Hierarchical Flow of the Developed Framework 99
Figure 4.6: Student Perceptions of Personalization Features 102
Figure 4.7 Pretest and Posttest Scores Comparison 105

ACRONYMS AND ABBREVIATIONS

AI	Artificial Intelligence
ANOVA	Analysis of Variance
API	Application Programming Interface
BERT	Bidirectional Encoder Representation from Transformers
BPS	Board of Postgraduate Studies
CI/CD	Continuous Integration / Continuous Deployment
DSLs	Domain Specific Languages
E2E	End-to-End
GenAI	Generative Artificial Intelligence
GPT	Generative Pre-trained Transformer
HCI	Human-Computer Interaction
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrated Development Environment
IT	Information Technology
ITS	Intelligent Tutoring System
ITS-CAL	Intelligent Tutoring System for Computer Assisted Learning
KT	Knowledge Tracing
LEVER	Learning to Verify Language-to-Code Generation
LLMs	Large Language Models
LMS	Learning Management Systems
ML	Machine Learning
NACOSTI	National Council of Science and Technology
NGINX	Engine X (a high-performance web server and reverse proxy server)
NLP	Natural Language Processing
PBL	Problem-Based Learning
PCA	Principal Component Analysis
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
RAG	Retrieval-Augmented Generation
REST	Representational State Transfer

SDT	Self-Determination Theory
SLR	Systematic Literature Review
SQLAlchemy ORM	Structured Query Language Alchemy Object Relational Mapper
SVM	Support Vector Machine
TA	Teaching Assistant
TPACK	Technological Pedagogical Content Knowledge
UDL	Universal Design for Learning
UI	User Interface
VPS	Virtual Private Server
ZPD	Zone of Proximal Development

OPERATIONAL DEFINITION OF TERMS

Adaptivity: The framework's ability to modify learning pathways, task complexity, and instructional strategies in real-time based on learner performance and behavior.

Coherence: The logical consistency and alignment of instructional elements ensuring that feedback, content, and progression are integrated to support a unified learning path.

Collaborative Learning: A mode of instruction that supports peer interaction, team-based tasks, and shared problem-solving within the GenAI learning environment.

Contextual Relevance: The alignment of content, examples, and feedback with learners' current programming level, task difficulty, and practical applications.

Framework: A structured, coherent system integrating LLM-powered modules, adaptive learning tools, and collaborative platforms to deliver personalized and contextually relevant programming instruction.

Model: A computational system, specifically a large language model (LLM), used to generate, interpret, and adapt instructional content based on learner inputs.

Personalization: The dynamic adjustment of instructional content, feedback, and pacing to align with individual learner profiles, needs, preferences, and prior knowledge.

Prompt Engineering: The design of structured, pedagogically oriented queries used to guide LLM outputs in a way that supports learning goals and content coherence.

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Programming has become a foundational skill in the digital era, driving innovation and problem-solving across industries. As the demand for skilled programmers rises globally, educational systems face increasing pressure to deliver effective programming instruction[1]. However, many learners encounter significant barriers, including abstract concepts, complex syntax, and insufficient feedback mechanisms[2]. These challenges are particularly acute in lower-level languages like C programming, where debugging, memory management, and pointer manipulation add layers of cognitive load for beginners.

In this context, the concept of personalized learning has gained traction as a powerful educational approach. Personalized learning refers to instruction that is customized to match individual students' pace, needs, strengths, and areas for improvement[3], [4]. It shifts away from the one-size-fits-all model by offering differentiated content, adaptive difficulty, and real-time feedback. For programming students, personalized learning can significantly enhance understanding by providing targeted support on specific problem areas, promoting confidence, and improving long-term retention of coding principles[5]. However, implementing personalized learning effectively, especially in technical subjects like programming, requires intelligent systems that can monitor learner progress and respond dynamically.

Despite the proliferation of programming resources, traditional education methods such as lectures, static textbooks, and template-based coding exercises remain dominant[2]. These conventional approaches often fail to accommodate diverse learning preferences, limiting learner engagement and practical skill acquisition[6]. Additionally, educators frequently struggle to update teaching materials to keep pace with fast-evolving technologies, further limiting instructional relevance. Research highlights that many programming learners struggle with understanding foundational concepts and translating theory into practice, especially when instruction lacks adaptability or personalized support[1], [7]. This gap between theory and application underscores the limitations of conventional instructional approaches.

To address these shortcomings, the education sector has increasingly turned to Artificial Intelligence (AI) and, more specifically, to Generative AI (GenAI). GenAI refers to machine learning models capable of creating new content including text, images, or code by learning from vast datasets. These models offer significant potential for transforming programming education by enabling real-time support, contextual explanations, and customized guidance[8]. Among GenAI technologies, Large Language Models (LLMs) such as Deepseek, GPT-3, GPT-4, OpenAI Codex, and CodeT5 have shown promising results in generating meaningful code snippets, translating natural language prompts into programming instructions, and simulating tutor-like interactions[7].

Early tools like GitHub Copilot and Microsoft IntelliCode exemplify the potential of LLMs to enhance productivity by auto-suggesting code and detecting errors[9]. These platforms use machine learning to enhance coding efficiency by suggesting syntax, identifying patterns, and streamlining development[10], [11]. However, these tools are primarily designed for use by professional software engineers and lack the instructional structure, scaffolding, and assessment features needed for novice learners. Their outputs, while technically useful, do not address educational needs such as curriculum alignment, conceptual progression, and personalized remediation[12].

Further developments in Natural Language Processing (NLP) have bridged the gap between human intent and machine-readable code. These advances allow learners to pose natural-language queries and receive code-based responses, explanations, or debugging suggestions in return[13]. However, while these approaches introduce a degree of flexibility, they struggle with capturing the nuances of human language and meeting contextual requirements in education. Many AI-generated responses lack coherence, are contextually irrelevant, or include factual errors. These inaccuracies can mislead learners and undermine trust in the system[11], [14]. Moreover, LLMs often struggle to assess learners' evolving knowledge levels or adjust task complexity accordingly.

Several studies have evaluated the ability of these models to autonomously generate code. One of the most notable models, OpenAI Codex (a descendant of GPT-3), which

was developed to solve coding tasks[15]. Codex can produce code in numerous programming languages and solve a wide range of problems. Additionally, Microsoft IntelliCode and GitHub Copilot use machine learning to assist programmers in writing code more efficiently. However, even with this potential, its raw output often requires refinement to be pedagogically useful. Without instructional design integration, these tools risk producing content that is either too advanced or misaligned with a learner's current understanding. These issues call for a structured, educationally informed framework to guide the responsible and effective deployment of GenAI tools in the classroom.

At present, AI-driven instructional models are still evolving. Many existing systems focus more on demonstrating technical feasibility than ensuring educational effectiveness[16]. As a result, they tend to overlook critical pedagogical dimensions such as engagement, cognitive load, scaffolding, and learner diversity. AI-generated content is often evaluated in terms of code accuracy but not in terms of learning outcomes, learner satisfaction, or knowledge retention. Furthermore, most tools lack feedback loops that allow learners to reflect, retry, and improve their understanding based on interaction history. These limitations reduce their impact on long-term educational outcomes.

Natural Language Processing and Machine Learning have been applied in programming education for years, yet the integration of these technologies into well-structured, personalized learning frameworks remains underexplored. Many students continue to struggle with bridging the gap between abstract programming theories and practical implementation[12], [17]. As programming languages evolve rapidly, instructors find it increasingly difficult to keep content updated and relevant. This situation makes the case stronger for AI-powered systems that not only assist in coding but also support personalized, curriculum-aligned instruction.

This study seeks to bridge this critical gap by proposing and evaluating a web-based instructional framework that utilizes GenAI specifically OpenAI GPT for personalized C programming instruction. The framework incorporates prompt engineering, adaptive difficulty scaling, and real-time learner interaction to create a dynamic, student-centered experience. Emphasis is placed on ensuring content accuracy,

contextual relevance, and usability, all within a pedagogically sound structure. The ultimate goal is to transform traditional approaches to programming instruction by making them more flexible, effective, and inclusive of each learner's needs. The outcomes of this research are expected to contribute to future designs of AI-driven learning systems that align both technological innovation and instructional goals.

1.2 Problem Statement

Traditional approaches to computer programming education adopt a structured, step-by-step learning process, focusing on syntax, logic, and procedural programming. These methods involve learning programming concepts through formal instruction but often overlook the diverse learning styles, prior knowledge, and individual pacing needs of students[18]. This one-size-fits-all methodology can lead to several challenges, such as students experiencing frustration and becoming disengaged when they struggle to grasp programming concepts due to the absence of personalized guidance and tailored learning materials. Additionally, inefficient learning occurs as students spend time on topics they already understand or face unnecessary difficulties with concepts they have not fully grasped. The lack of personalization in programming education limits accessibility for students with diverse learning preferences, thereby hindering their overall educational experience[7], [19], [20].

The emergence of Generative Artificial Intelligence (GenAI), particularly Large Language Models (LLMs) such as OpenAI's GPT series, Gemini, CodeT5, and DeepSeek as well as LLM-powered tools like GitHub Copilot and TabNine, has introduced innovative solutions that can be used to enhance personalization in programming education by generating code and providing contextual explanations[21]. These models have demonstrated potential in supporting learners by explaining complex concepts and automating code generation. Researchers have argued that such models can help automate learning processes by providing real-time feedback tailored to learners' abilities and facilitating adaptive learning paths[4], [22], [23]. Several frameworks have emerged to operationalize these capabilities in educational settings, including ITS-CAL, Artemis GPT-3.5 Tutor, and TutorLLM, which integrate features such as hint generation, adaptive question delivery, and conversational feedback to support personalized programming instruction[24], [25], [26].

However, existing Large Language Model (LLM) frameworks lack the necessary adaptability and responsiveness to provide personalized learning experiences that cater to different learning styles and paces. The lack of standardized input structures and the nuances of natural language processing often lead to inaccurate, contextually inappropriate, or incoherent outputs[7], [8]. Moreover, existing frameworks still face practical and pedagogical limitations, ITS-CAL often leads to over-reliance due to limited scaffolding[24], Artemis GPT-3.5 Tutor suffers from inconsistent feedback quality[25], and TutorLLM depends on complex infrastructure, which restricts scalability and ease of deployment[26]. These shortcomings pose a substantial barrier to the effective deployment of GenAI in educational settings, thereby undermining the quality of the learning experience. Inaccurate outputs can potentially lead to misconceptions, while irrelevant or incoherent AI-generated content may disrupt the learning process instead of enhancing it.

The challenges are particularly evident among undergraduate students studying C programming, where limited access to practical lab sessions, outdated teaching materials, and large class sizes that hinder one-on-one instruction make learning more difficult, especially in resource-constrained educational settings such as those found in Kenya. Additionally, insufficient research exists on how GenAI inputs should be structured and applied to optimize their effectiveness in educational environments[27]. Current implementations of these models lack a standardized framework, resulting in inconsistencies in content quality, contextual relevance, and overall effectiveness[28]. Therefore, there is a critical need for a robust framework that guides the structured implementation of these models in programming education. This research seeks to develop such a framework to ensure the integrity, coherence, relevance, and accuracy of AI-generated educational content, thereby enhancing personalized learning experiences for students.

1.3 Research Objectives

1.3.1 General Objective

To develop a Large Language Model-Based Framework that enhances personalized learning in C programming.

1.3.2 Specific Objectives

i. To analyze existing LLM-based frameworks and tools to identify features and best practices that support effective personalized learning in C programming.

ii. To develop a novel LLM-based framework that enhances personalized programming instruction by incorporating the identified features and best practices.

iii. To evaluate the effectiveness of the developed framework in improving student engagement, learning outcomes, and content quality.

1.4 Research Questions

i. What features and best practices in existing LLM-based frameworks support effective personalized learning in C programming?

ii. How can a personalized programming instruction LLM-based framework be developed using the features and best practices identified from existing LLM-based tools?

iii. To what extent does the developed framework improve student engagement, learning outcomes, and the quality of instructional content in C programming?

1.5 Scope of the Study

This study focuses on developing a personalized learning framework powered by Large Language Models for introductory computer programming education, specifically in the C programming language. The scope includes the design, implementation, and empirical evaluation of a web-based framework that supports adaptive content delivery, real-time assistance, and prompt-driven feedback. While C programming is used as the initial implementation language, the framework is designed to be extensible to other languages such as Python and Java in future iterations. The research setting is confined to undergraduate students at Murang'a University of Technology enrolled in an introductory programming course. The study includes both an experimental group using the LLM-based system and a control group relying on traditional learning resources. Comparative analysis is used to assess their learning outcomes and engagement levels. Additionally, strategies are developed to

mitigate biases in AI-generated content, ensuring fair and equitable learning opportunities for all students.

1.6 Significance of the Study

This research is significant in advancing pedagogical approaches for computer programming education through personalizing the learning experience. The study builds a framework that leverages LLMs to deliver tailored instructional support based on individual learner's needs, abilities, and contexts. Embedding the framework into existing programming curricula, educational institutions can shift from one-size-fits-all instruction to adaptive, learner-centered environments. This shift is expected to enhance student engagement, motivation, and retention by ensuring that each student receives the right support at the right time. Additionally, the findings of this study offer insights for educational policy by offering guidelines for the ethical implementation of AI-driven learning tools, ensuring that they are deployed effectively and equitably while addressing concerns related to bias and data privacy.

Personalized learning in this framework is achieved through multiple integrated mechanisms. These include learner profiling to capture background knowledge and learning styles, real-time analysis of student-written code, adaptive feedback generation, and conversational tutoring powered by the LLM. This dynamic interaction enables the system to identify learner weaknesses, provide targeted explanations, and recommend appropriate resources such as exercises, examples, or code corrections. As a result, learners are supported in a way that is responsive to their specific struggles and progress, reducing cognitive overload and frustration.

This study conducts a detailed analysis of existing GenAI application frameworks to identify features and best practices that contribute to high-quality educational content, specifically in terms of coherence, relevance, and accuracy. These insights inform the design of the proposed framework, ensuring that it not only supports personalization but also upholds content integrity. From an instructional standpoint, educators benefit from the framework's ability to monitor learner interactions and performance. The framework provides actionable insights that allow instructors to offer more precise interventions where needed, while reducing their burden through automated, intelligent tutoring.

Additionally, the study offers guidance for educational policy and system design by addressing potential ethical concerns regarding bias mitigation, fairness, and data privacy. Measures are implemented to anonymize student data, ensuring privacy and compliance with relevant regulations. Although challenges exist, the potential benefits for students such as enhanced engagement, motivation, and deeper comprehension of programming concepts outweigh the limitations. Through the balancing of both advantages and ethical concerns, this research contributes a well-informed argument for the integration of LLMs in teaching and learning environments.

1.7 Limitations of the Study

Despite the framework's promising approach, the study has certain limitations. It focuses solely on C programming, which limits its immediate applicability to other programming paradigms or languages. The research is conducted within a single institution, which may affect the generalizability of the results to other academic or cultural contexts. Additionally, the study only involves undergraduate students in introductory courses, meaning outcomes might differ at advanced levels of study. Long-term retention, code optimization, and the broader socio-cultural implications of LLM integration are beyond the scope of this thesis and are recommended for future research.

1.8 Contributions of the Thesis

This research makes significant contributions to theory, practice, and policy in the domain of personalized programming education using Large Language Models (LLMs). The core contribution is a structured framework designed to enhance individual learning experiences through adaptive and intelligent support systems. This framework is realized through a functional web-based application and supported by empirical evaluation.

1.8.1 Contribution to Theory

This research contributes to the theoretical discourse by developing a web-based framework that advances understanding of how Large Language Models can support personalized learning in programming education. Specifically, it introduces an integrated system in which adaptive learning, GenAI integration, collaborative coding, and real-time feedback function collectively rather than in isolation. Another

contribution is the formalization of prompt engineering as a central mechanism in optimizing LLM output for educational purposes, highlighting how the structure and design of prompts directly influence content relevance, clarity, and coherence. This contributes to existing literature on intelligent tutoring systems and adaptive learning technologies by grounding personalization in both student context and LLM response behaviour.

1.8.2 Contribution to Practice

The research makes a substantial practical contribution by translating the theoretical framework into a fully functional web-based learning application. This application enables undergraduate students learning the C programming language to interact with an LLM for code analysis, feedback, and concept clarification in real time. The system supports personalized learning pathways by adapting responses to each student's input and background. An empirical study involving undergraduate students demonstrated that the system improved learner engagement, enhanced comprehension of programming concepts, and increased overall satisfaction, validating the effectiveness of the practical implementation.

1.8.3 Contribution to Policy

From a policy perspective, the research contributes to ongoing discussions around the ethical deployment of AI in education. It addresses concerns related to data privacy, fairness, and bias in AI-generated learning content. Strategies such as student data anonymization and bias mitigation in prompt responses are embedded into the framework. These considerations provide evidence-based recommendations for the ethical integration of LLMs in learning environments, guiding institutions and policymakers in the responsible adoption of generative AI tools.

1.9 Organization of the Thesis

This thesis is organized into five chapters as follows:

Chapter one describes the background of the study, the problem statement, the research objectives and research questions, the study's significance, scope and limitations, conceptual framework, and the contributions made by this thesis.

Chapter two presents the scholarly literature that was reviewed in this study. The literature synthesizes explored theoretical foundations, empirical evidence, and the challenges and limitations of using LLMs with an aim of exposing the existing gaps.

Chapter three outlines a detailed research methodology followed to achieve the objectives of this study. The Research design, research process, data acquisition and analysis, ethical considerations, and experimental materials are covered in this chapter.

Chapter Four presents the results of this study and discusses their implications for the design and implementation of Generative Artificial Intelligence frameworks in supporting personalized learning experiences.

Chapter five presents the conclusion of the findings of this study, the proposed recommendations from the findings, and future research directions that could be adopted for future work.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter reviews key developments in AI especially the emergence of Large Language Models as applied to programming education. It synthesizes existing theoretical foundations, empirical studies, and practical implementations of LLM-driven tools, and highlights both strengths and shortcomings in personalizing programming learning. The discussion sets the stage for designing an improved LLM-based framework and evaluating its impact on student engagement, outcomes, and content quality.

2.2 Theoretical Foundations for Personalized Instruction

Theoretical frameworks in educational research provide the foundation for designing effective, learner-centered instructional strategies. In the context of this study, the focus is on theories that inform personalized learning approaches in computer programming, particularly those that support the integration of LLMs. These frameworks guide the development of interventions that adapt to learners' individual needs, preferences, and learning trajectories.

2.2.1 Cognitive Theories Supporting Personalized Learning

2.2.1.1 Zone of Proximal Development (ZPD)

The Zone of Proximal Development (ZPD) is the difference between the actual level of development of a person as assessed by his or her solving of problems on his/her own and the prospective level of development as assessed by his/her solving of problems with the help of an adult or a more experienced peer[30]. The trainers can choose the right method for a particular student by evaluating his or her current ability level and then they can make the necessary adjustments to the lesson so that the student can be challenged appropriately but at the same time he or she is getting the help he or she needs to move forward. The fact that students work together on programming tasks helps them in acquiring better results due to the fact that they also acquire the skills of critical thinking and communication[31]. Such group activities enable students to get ready for their future careers as they are strengthening what they have learned and simulating the real-world programming conditions.

2.2.1.2 Adaptive Learning Systems

Adaptive learning systems examine the learners' knowledge, abilities, and learning preferences through the use of algorithms and data analytics. The highly personalized educational experiences which can be obtained by modifying the learning materials, the pace, and the evaluations using this data-driven technique are possible according to the given fact[16]. In programming, adaptive learning systems can change the level of content complexity according to the students' performances, thus ensuring that they are appropriately challenged but not too much so that they become dissatisfied or bored. Research shows that the process of learning programming becomes faster and the learning outcomes are better when the feedback is given right after the errors occur[32]. Through adaptive learning systems, individual learning gaps can be recognized by recommending certain activities, modules, or resources based on the analysis of learner data which in turn makes a group of learners the same. The trainers may use the data to discover the zones of the problem, the performance patterns, and the students' progress[33]. With the use of this data, educators can spot at-risk learners early on, provide tailored assistance, and modify their lesson plans as necessary.

2.2.2 Pedagogical Theories Supporting Personalized Learning

2.2.2.1 Differentiated Instruction

Differentiation is basically the modification of the curriculum according to the needs of the students. This way of instruction is effective because it utilizes flexible grouping and ongoing assessment, irrespective of the fact that the teachers differentiate the material, process, the products, or the learning environment[34]. The differentiated instruction methods can significantly enhance students' learning and in-class experience when it comes to the teaching of programming. Teachers may in turn make the classroom a place where every student can succeed by the use of personalized instruction strategies, resources and evaluation procedures which account for the individual differences of students[35]. Differentiated instruction is based on the basis that each student has unique backgrounds, experiences, learning styles, and skills before the class starts[36].

2.2.2.2 Self-Determination Theory (SDT)

Self-Determination Theory (SDT) puts a lot of importance on satisfying the basic psychological needs that are autonomy, competence, and relatedness, as well as the

intrinsic motivation[37]. SDT, when used with the individualized learning in programming, enables instructors to learn about the ways to create meaningful and effective learning environments that will help students to achieve in-depth understanding and skill development, hence, in this way, the learning process will be made easy and memorable. Students are given a certain degree of freedom in the educational programming which means that they can decide what they want to study and how they want to study. As research shows, learners are more interested and involved in learning situations that allow them to be independent[37]. Enabling learners to select their programming projects or to choose among various learning resources, authorizes them to control their learning process. Figure 2.1 below displays the three major pillars of the SDT: autonomy, competence, and relatedness.

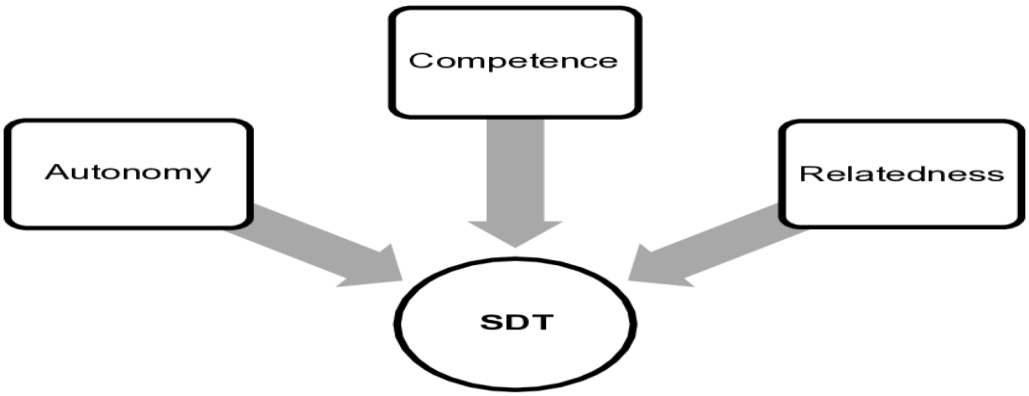


Figure 2.1: Self-Determination Theory (SDT) by Deci and Ryan

Self-Determination Theory (SDT) holds that deep, lasting motivation emerges when three basic needs are met. In autonomy learners choose tasks for example selecting their own coding projects, in competence they receive timely, tailored feedback that confirms and builds their skills, and for the relatedness the learners feel supported through peer collaboration and mentoring. In a personalized programming environment, honouring these needs means offering meaningful choices, delivering scaffolded feedback on code, and fostering a sense of community together fuelling students' intrinsic drive to learn and persist.

As per the research, students who feel that they can learn are more likely to persevere and go on when they are faced with difficulties in the programming assignments[38]. Humans are social creatures and therefore, SDT puts a high importance on the need for connection and the feeling of being a part of a group. In programming, the

opportunities for the students to work in teams, to be in the contact with their peers, and to get mentored by the experienced people create the feeling of community among the students. Via group discussions, online communities, and collaborating on the programming projects, students may interact with one another, share experiences and encourage one another on their path of learning[35].

2.2.2.3 Problem-Based Learning (PBL)

A teaching approach known as problem-based learning (PBL) puts students at the centre of their learning settings by giving them real-world issues to answer. PBL provides a dynamic, interesting method for learning programming principles and abilities in the context of teaching programming[39]. The primary focus of PBL is on how to incorporate real-world challenges into the teaching and learning process. PBL fosters a contextual understanding of topics by immersing students in real-world programming challenges, which aids in knowledge transfer and retention. PBL also promotes active inquiry and investigation, encouraging students to pose questions, look into issues, and refine their answers over time[39]. This method not only develops critical thinking abilities but also gives students a feeling of control and ownership over their education as shown in figure 2.2 below.

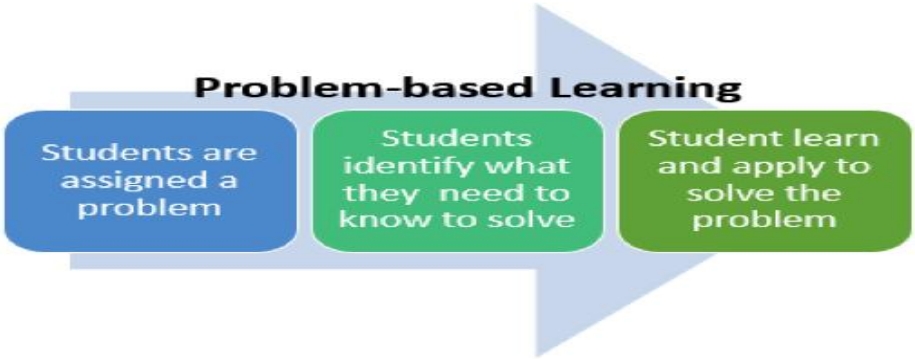


Figure 2.2: Problem-based Learning

According to research, PBL helps students grasp programming ideas more deeply by having them solve challenging issues in relevant situations[39]. Additionally, PBL develops critical thinking and problem-solving abilities, giving students the mental tools needed to understand the complexities of software development. Furthermore, PBL's collaborative style encourages cooperation and communication skills, reflecting the kind of collaborative settings common in the IT sector. To ensure relevance and

engagement, educators need to carefully select real-world situations that align with the interests and goals of their students. In addition, the success of PBL programs depends critically on the efficient facilitation of group work, which calls on trainers to promote a collaborative and respectful environment while offering direction and support.

2.2.3 LLMs in Educational Frameworks

Large Language Models (LLMs) like GPT-3 and BERT have been increasingly used in educational settings to generate content and assist in teaching[12]. These models are known for their ability to process and generate human-like text, making them valuable tools in education. However, they are not without limitations. One significant issue is that these models can sometimes produce outputs that are not entirely aligned with educational goals or that fail to fully grasp the nuances of instructional content[17]. Addressing these limitations is crucial for ensuring that LLMs can be effectively integrated into personalized learning frameworks.

2.2.3.1 Large Language Model Underpinnings

Language models are typically created in the field of natural language processing (NLP) for a variety of specialized tasks, such as information extraction, machine translation, answering questions, among others[40]. On the other hand, large language models (LLMs) are machine learning-based language models with a high number of parameters that have been trained on enormous amounts of data. This allows them to handle a variety of jobs without the need for retraining in addition to having a more powerful capability. LLMs in general are dramatically changing the field of NLP by their outstanding capability to understand, produce and modify human language on a huge scale[41].

The LLMs are based on the trailblazing Transformer Architecture, which is the basic structure for the known examples like OpenAI's GPT series. This architectural style will be the base that the LLMs will use to comprehend the complex long-range connections and contextual subtleties that are included in the textual material. The self-attention mechanism which is a part of the transformer architecture is a major factor of its success, as it allows the LLMs to comprehend the meaning of single words in a phrase as shown in figure 2.3. Through the process of the dynamic assessment of word meaning, this method gives the greatest prediction accuracy and promotes a

comprehensive study of language which has never been like before in both its breadth and depth.

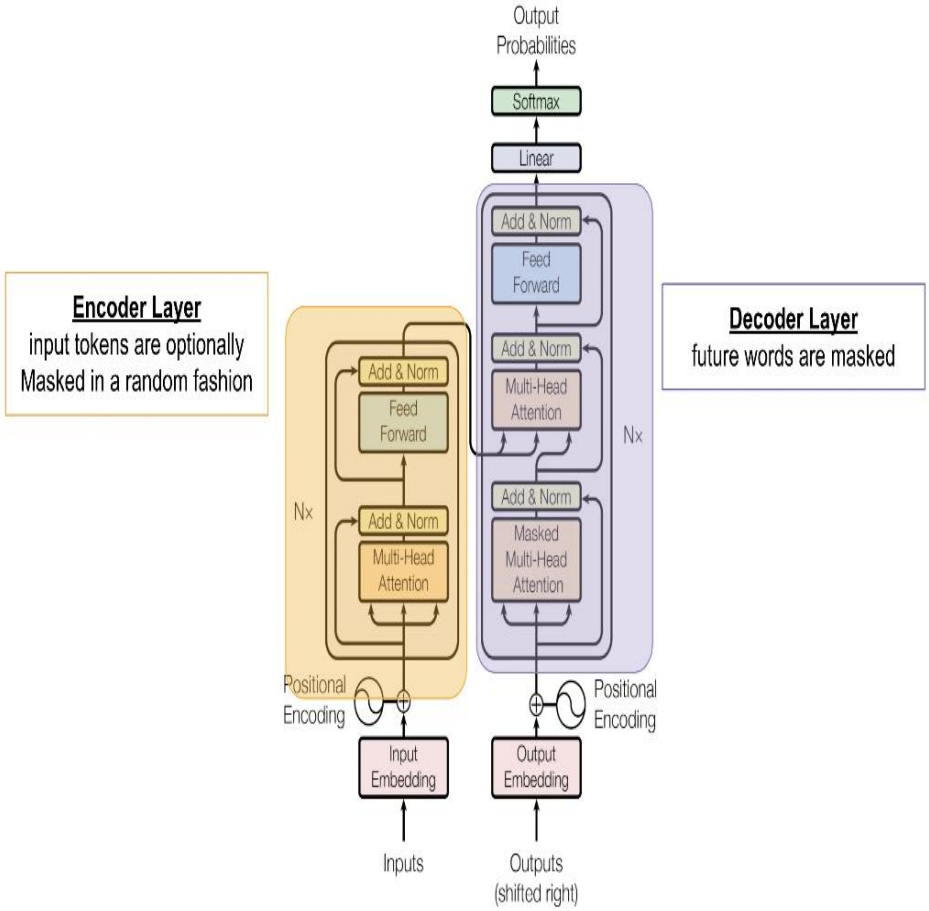


Figure 2.3: Annotated Transformer Architecture

In the field of machine learning, the main idea of transfer learning is the application of the data from one task to the better performance of another[18]. This method has great potential in the field of Natural Language Processing (NLP), especially when used with large language models pre-training[42]. The pre-training is the most crucial part of the transfer learning which enables LLMs to be trained with a huge amount of unlabelled text data and then they are optimized for specific tasks[31]. On the contrary, this lets them have a very deep understanding of the language features, the semantics, and the syntactic structures.

LLMs have the possibility of reviewing a lot of text sources like books, journals, websites and other repositories during the pre-training stage. LLMs go through a phase of fine-tuning in which they change their already acquired representations so that they

fit in the downstream tasks, for example, sentiment analysis, language modelling, and text categorization[12]. The parameters of the pre-trained model are tweaked by the extra training that is done on the task-specific labelled data; thus, the model gets to become specialized in its knowledge and its representations are brought to the level of the target task. Pre-training and transfer learning are extremely important since they help LLMs become more adaptive and generalized[14]. Due to their pre-training exposure to a wide range of linguistic contexts, LLMs get a comprehensive grasp of language patterns and semantics, which makes it easier for them to transfer their knowledge to new tasks and domains with less task-specific training data.

The primary characteristic of the LLMs lies in their capability of turning the raw language input into the dense vector representations, the so-called embeddings. These embeddings are the short, high-dimensional representations of linguistic elements which not only contain semantic but also syntactic information of the text data[22]. The semantic meaning and syntactic structure are the main part of LLMs' language representation. Embeddings allow LLMs to be able to deal with the complexity of human language by representing a wide range of semantic connections, contextual interferences, and linguistic nuances, thus, LLMs can be able to understand the meaning of the words with their correct context[10]. The information input is translated into the dense vector representations which enables the LLMs to perform various language-related tasks with the highest accuracy and efficiency.

2.2.3.2 Potential Applications in Education

Through its advanced natural language processing capabilities, LLMs allow the teacher to change the traditional teaching methods and thus improve the learning of the student[2]. Through the use of their knowledge in language processing, the LLMs can cater to the needs of the students, improve teaching methods, and help in the educational research, thus the educational system can be completely transformed[23]. Some of the innovations that facilitate this activity include automated grading, personalized feedback, adaptive learning systems, and virtual tutoring.

The marking of written tasks and the giving of students personalized feedback through the use of large language models is one of the most thrilling applications of LLMs in education. Teachers' time and resources are often very much used by the traditional

grading techniques, which limits the amount and level of feedback they can give students[15]. Yet, LLMs are in a position to give an opinion by using their language processing capabilities to match student answers to the pre-set rubrics or the model-generated solutions. Among the factors that might be evaluated in a student's work are their understanding, writing ability, critical thinking abilities, and the compliance with some rules or learning objectives, which are just few of the components of their work. Besides, LLMs are capable of giving individualized feedback to each student according to his or her learning needs and the areas in which he or she needs to improve[43]. LLMs can identify the usual mistakes, misunderstandings or the questions that need to be answered further through the analysis of the subtleties of students' answers.

LLMs change the teaching way of the educational information that is adjusted to the different needs and preferences of the individual learners through the adaptive learning systems. The usual educational systems are usually not sufficient to incorporate the different learning styles, levels of understanding, and the rate of comprehension of learners[31]. LLMs offer the student a solution by processing their language skills to analyze how the student is dealing with the course materials and then they modify the lessons in order to reach the learning objectives. Through LLMs, the educators are able to clearly understand the students' individual learning patterns, their strengths and weaknesses, and the extent of their engagement, thus, bridging the gap between the students' personal as well as academic requirements are able to be met[16]. The adaptive learning systems will provide to every student a distinct learning experience that is adjusted to his/her own needs and learning path by changing the level of the difficulty, the speed, and the order of the learning activities according to these insights.

Through the use of the language processing capabilities of large language models, natural language tutoring systems make a big jump in educational technology and give students one-to-one personalized and interactive help[44]. Through the role of virtual tutors, these programs complement the traditional classroom education and motivate the students to take on-demand help, explanations, and guidance in the natural language. The ability of LLMs to comprehend natural language inquiries and to give answers that look like human responses is what makes natural language tutoring systems. Students can ask the tutoring system by raising questions, getting the

explanation for the concepts, or requesting for assistance with the specific projects[44]. LLMs analyze each student's input through the latest language processing algorithms that generate answers that fit the specific needs of every student.

2.2.3.3 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) has gained traction in NLP for knowledge-intensive tasks such as question answering or summarization by combining the deep language understanding of transformer models with an explicit retrieval step. In RAG, a retriever first searches a large corpus (for example, API documentation or programming textbooks) to find passages most relevant to the learner's query. These passages are then fed into a generator, typically a fine-tuned transformer, which weaves together the retrieved context and the original question to produce an accurate, contextually grounded explanation or code snippet[45]. In the context of personalized programming instruction, RAG ensures that generated hints and examples are not only fluent but also factually correct and aligned with authoritative sources. In the RAG pipeline as outlined in the illustration in figure 2.4 below, a retriever module retrieves relevant documents from an external knowledge base; these documents are then passed to a generator module, which produces the final, context-grounded output.

While RAG delivers more reliable responses than unaugmented LLMs, existing educational prototypes fetch broadly relevant documents without tying retrieval priorities to each student's unique misunderstandings or mastery level. As a result, learners may receive precise information, but not necessarily the right information for their current knowledge gaps.

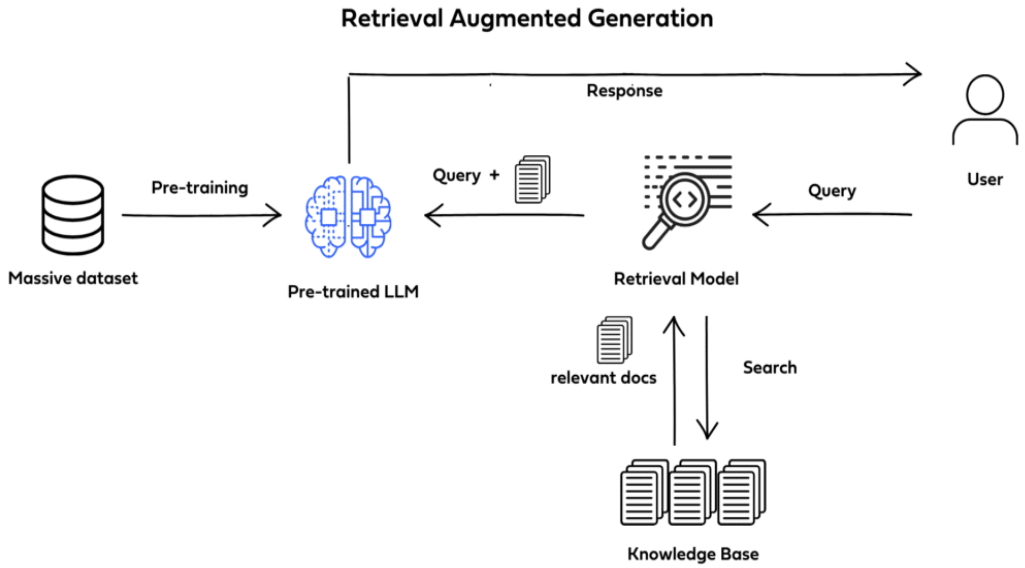


Figure 2.4: RAG Architecture

2.2.3.4 Knowledge Tracing

Knowledge Tracing (KT) models a learner's evolving understanding of discrete skills by analyzing interaction logs such as quiz results or coding exercise attempts and estimating mastery probabilities[46]. KT systems enable adaptive sequencing of practice tasks, offering review where needed and advancing students through mastered concepts. Although KT provides real-time mastery insights, existing RAG-powered tutors rarely integrate these predictions into their retrieval strategy. As a result, students receive high-quality information, but not necessarily the right information that targets their specific knowledge deficiencies.

2.2.4 Features & Best Practices from Existing Tools

Research on LLM-powered programming assistants reveals recurring themes around code generation, conversational tutoring, and adaptive prompt design. Critically examining these features uncovers both strengths to build on and gaps to address.

2.2.4.1 Code Generation & Hint-Generation

GitHub Copilot and its underlying model, OpenAI Codex, pioneered automated code completion by predicting what developers intend to write next. In an observational study involving introductory programming students, researchers found that novices appreciated Copilot's ability to fill in syntactic boilerplate, but often accepted incorrect

or suboptimal suggestions without question, slowing their development of debugging skills[47]. The study on novice interaction reported that while Copilot can generate entire functions, it frequently omits critical edge cases, forcing learners to patch rather than understand underlying logic. Existing code generation tools accelerate coding but lack pedagogical scaffolds to help learners internalize programming concepts, instead encouraging over-reliance on opaque suggestions.

2.2.4.2 Dialogue-Based Tutors

LLM chat interfaces like the ChatGPT offer on-demand, conversational explanations. Researchers conducted a study where students valued ChatGPT for clarifying error messages in real time and found responses were often overly generic and disconnected from course curricula[48]. Other studies similarly reported that AI tutors boost productivity by suggesting alternative solutions, but without structured dialogue flows, students struggled to request deeper elaboration on topics they misunderstood[49]. Effective conversational tutors therefore require guided dialog management predefined question-answer paths and fallback prompts to ensure discussions remain pedagogically focused. The open-ended LLM dialogues provide flexibility but lack structured conversational design to consistently scaffold learner understanding within a programming curriculum.

2.2.4.3 Adaptive Content Delivery

Adaptive content delivery refers to the system's ability to adjust instructional materials including explanations, examples, and practice tasks in real time based on each student's demonstrated understanding. Traditional ITS employ knowledge tracing algorithms such as deep knowledge tracing to model learner mastery and select the next activity[46]. Recent work integrates LLMs into this pipeline: for instance, an LLM-driven tutor can choose between multiple worked examples or scaffolded hints depending on a learner's error patterns logged during coding tasks[40]. In another study, adaptive sequencing of code exercises guided by a Bayesian student model improved problem-solving transfer by 20% compared to static modules[50]. However, most LLM deployments today remain one-size-fits-all: they present the same explanation regardless of whether a student has repeatedly shown mastery or is making the same mistake for the third time. This lack of granularity limits their ability to keep advanced learners challenged or to revisit foundational concepts for struggling

students. Despite promising prototypes, no existing system dynamically curates and re-orders LLM-generated content based on continuous assessment of individual learners' knowledge states.

2.2.4.4 Prompt Engineering Strategies

Prompt engineering shapes how an LLM interprets and answers a query. Techniques such as chain-of-thought prompting coax the model into revealing intermediate reasoning steps, dramatically boosting accuracy on reasoning tasks[51]. In programming contexts, prompts that ask "Explain your debugging process step by step" result in more pedagogically useful guidance than a simple "Fix this code" instruction[52]. Few-shot prompting providing 3–5 exemplar Q&A pairs further anchors the model in the desired explanation style[53]. Yet, these strategies are almost always static: once chosen, the same prompt template is reused for all learners. There is no mechanism to automatically vary prompt complexity, hint depth, or the number of examples based on a student's prior responses. As a result, advanced students may be under-challenged by overly scaffolded prompts, while novices may be overwhelmed by insufficient guidance.

2.2.5 Integrating Technology in Education

Technology integration in school is a complicated and dynamic course of action which is more than just putting devices and software in classrooms. It is based on a complete revision of teaching techniques, which are strongly tied to theoretical models that provide the road map and the assistance for its successful implementation and the significant impact. While theories like constructivism emphasize the active role of learners in constructing their own understanding, applying these theories within AI-driven frameworks presents challenges[54]. For example, ensuring that AI systems can support constructivist learning without overwhelming students with too much information is a delicate balance. Moreover, studies on AI in education have shown mixed results, often due to variations in implementation and context.

2.2.5.1 Conventional Learning Materials for Computer Programming

The learning materials for computer programming which are given in the traditional way have been the most important factor of the education to the students. These materials, such as textbooks, lecture notes, and practice exercises, have been the

mainstay of programming education[9]. Even though they are the main ways to teach the basics, their immobility is often the reason for the difficulties of students' learning styles and needs. Moreover, the fast development of the programming languages and technologies is making it hard for these materials to be up-to-date and applicable.

2.2.5.2 Technological Pedagogical Content Knowledge (TPACK)

The technological pedagogical content knowledge (TPACK) framework is a significant paradigm change in the educational practice, which put forward the complex interactions of topic mastery, pedagogical competence and technological competency[55]. At the core, TPACK underscores the significance of the teachers' constant negotiation with various areas, using technology as a flexible tool to boost the efficiency of the teaching and learning processes as shown in figure 2.5.

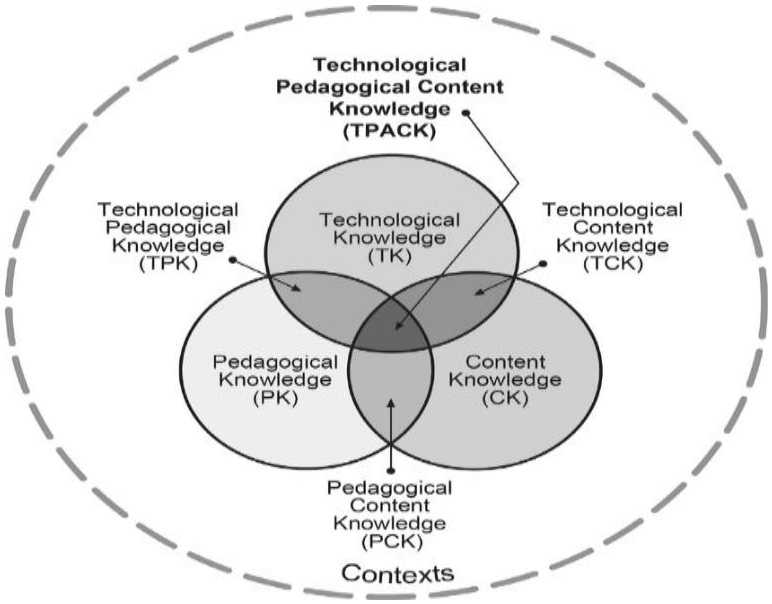


Figure 2.5: The TPACK framework

Based on the TPACK, the teachers who are very good at the technology integration in the classroom, they are good at the coordination of the technical resources, the instructional techniques and the subject-matter knowledge and they should produce the meaningful learning opportunities[55]. Besides, TPACK underlines the role of the context, and it is very important for the teachers to adapt their methods of integrating technology to the subject that they are teaching, the special needs of their students, their interests and their histories.

2.2.5.3 Pedagogical Approaches on Explaining Code

The skills of coding and editing, the understanding of its goals and the explanation of its functionality are the main characteristics of programming students[43], [56], [57]. The computing students must gain the ability to be good not only in coding but also in understanding the reason and explaining the function of the code they write or manipulate. Explaining process of a code snippet to a student is the first step of developing the student connection and his/her own understanding of the whole execution of a given code snippet[57]. Besides that, clarifications are of great importance in the reduction of stress and effort required for bug-fixing of one's code, finally it makes the learning process more enjoyable.

Myriad of teaching methods have been studied in order to improve students' code understanding. Activities that involve groups like pair programming and the techniques such as misconception-based feedback make peer explanations and discussions about code possible which in turn enable a deeper understanding[58]. These techniques are designed to provide students with the ways of studying the code they encounter during the learning process and to increase their ability to explain the code[59]. During the step of explaining code to peers and the one of tracing code execution are cognitively demanding tasks, not all students are willing to participate in these activities[60]. Instructors, due to time limitations, may find it difficult to create a custom explanation for the student, hence the automatic feedback generation of explanations to support the scalable learning will be a possible solution.

2.2.5.4 Principles of Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) refers to the ability of machines to perform tasks that typically require human intelligence. This includes understanding natural language, recognizing patterns, solving problems, and making decisions[61]. AI comprises of technologies which include machine learning, expert systems, neural networks, robotics, and natural language processing (NLP). Machine Learning (ML), a subset of AI, focuses on the development of algorithms that enable computers to learn from and make predictions or decisions based on data[9]. Unlike traditional programming, where specific instructions are coded, ML systems are trained using data to recognize patterns and improve their performance over time.

Supervised learning algorithms are trained on labeled datasets where the input-output pairs are provided. In personalizing computer programming, these algorithms predict student performance based on historical data, enabling the creation of customized learning paths[62]. For example, linear regression and support vector machines (SVM) can analyze student performance data and recommend targeted programming exercises to address specific areas of improvement. Unsupervised learning involves training algorithms on data without labeled responses, identifying hidden patterns or structures within the data. Techniques like clustering can group students based on their learning behaviors and coding styles, allowing for tailored instructional strategies[63]. Dimensionality reduction methods such as principal component analysis (PCA) simplify complex programming concepts, making them more accessible to students.

2.3 Theoretical Framework

The fusion of Large Language Models (LLMs) in the individualized learning settings is one of the most promising innovation strategies that can be able to revolutionize computer programming education. The main goal of this given proposed theoretical framework is to give the mechanism of the LLMs to exploit their potential to make the programming teaching and learning better. The coming to the realization of the power of LLMs in transforming the educational system by providing learners with the individual and flexible support is due to the ongoing development of the technology.

Personalized learning frameworks are crucial in programming education because students often struggle in large class settings where individual doubts may go unresolved. Traditional teaching methods cannot fully address the diverse pacing and support needs of each student[24]. For instance, novices learning C may encounter syntax errors or logical bugs and have to wait a long time for instructor feedback, leading to frustration or loss of motivation[24]. A personalized AI-driven framework can tackle this gap by giving immediate, tailored feedback. Research has shown that AI tutors can significantly improve student performance and satisfaction in programming courses by providing timely hints and adaptive learning paths. In other words, students receive the right help at the right time, allowing them to overcome obstacles and build confidence.

2.3.1 Existing GenAI Application Frameworks

Recent advances in generative AI have introduced powerful tools for programming education. Large Language Models (LLMs) like GPT-3.5 and GPT-4 can engage learners in natural language, generate code examples on the fly, and provide immediate feedback. This enables a level of interactivity and personalization previously difficult to achieve at scale[64], [65]. In programming domains, LLM-based assistants offer students on-demand help with coding tasks, explanations, and debugging tips[65]. These generative tutors can simulate aspects of human tutoring by adapting to a student's input and questions in real time[64]. However, integrating GenAI into education also raises challenges around maintaining instructional quality and accuracy. Ensuring the AI's responses remain coherent, relevant to the curriculum, and free of serious errors is critical[64], [66].

2.3.2 Overview of GenAI in Education

Generative AI is increasingly applied to education as a means to deliver personalized and scalable learning support. In broad educational contexts, LLM-based systems can act as virtual tutors, providing interactive dialogue, answering questions, and adjusting explanations to the learner[64]. Unlike static learning resources, these models generate content dynamically for example, rephrasing a concept or giving a new coding example on request[64]. This dynamic capability has been shown to increase learner engagement and offer more human-like support compared to traditional computer-based training[64]. Particularly in computer programming education, GenAI tools supply students with instant help: they can suggest code snippets, identify syntax errors, or explain programming concepts in natural language[65]. Such on-demand assistance can democratize access to individualized tutoring, since an AI tutor can be available 24/7 for any student with an internet connection. Early studies report positive outcomes, like improved student motivation and faster problem resolution, when AI assistance is available during coding tasks[50], [67]. At the same time, researchers caution that unrestricted use of LLMs might lead to over-reliance. For instance, one study found that students who heavily relied on LLMs for code generation and debugging tended to have lower final exam scores, suggesting that critical thinking skills may suffer if the AI simply provides answers[65]. These findings underscore the importance of using GenAI as a supportive tool rather than a crutch, and designing

frameworks that encourage learning (e.g. by guiding students through problems) instead of just giving away solutions. Generative AI holds great promise in education due to its adaptability and interactivity, but thoughtful framework design is required to maintain pedagogical integrity and effectiveness.

2.3.3 AI-Driven Tutoring Frameworks for Programming Education

Generative AI has been implemented in various forms to support intelligent tutoring systems in programming education. Modern frameworks utilize LLMs as conversational agents, virtual assistants, or automated evaluators to provide personalized instruction that exceeds the capabilities of traditional e-learning systems. Unlike early rule-based systems that followed predetermined scripts, LLM-powered tutors can dynamically generate context-sensitive explanations, questions, and feedback[64]. Several universities have reported improved student engagement and satisfaction following the integration of chatbot tutors into their programming courses [24]. Similarly, commercial platforms like GitHub Copilot, driven by OpenAI Codex, facilitate the learning process by suggesting code completions and functional implementations. To better understand the scope and potential of GenAI in programming education, table 2.1 provides a summary of notable frameworks developed since 2019. These include tools and frameworks that utilize LLMs or similar AI technologies to offer tutoring and coding support, with a specific emphasis on applications relevant to C language instruction.

Table 2.1: Examples of AI-Driven Frameworks for Programming

Framework (Year)	Approach	Features	Reported Outcomes	Identified Gaps
ITS-CAL (2023)	LLM-based coding tutor.	Hints generation, debug assistance, and Q&A support.	Improved pass rates; overuse reduced independence[24].	Lacks scaffolding; no long-term learner tracking
Artemis GPT-3.5 Tutor (2024)	GPT-3.5 in auto-assessment system	Real-time code feedback and corrections.	Scalable, timely help; some responses too generic[25].	Inconsistent response; lacks adaptive control
TutorLLM (2024)	LLM + Knowledge Tracing + RAG.	Personalized questions, grounded responses.	Higher user satisfaction and quiz scores[26].	Requires external retrieval; hard to align with curriculum
Socratic Tutor (SPL) (2024)	GPT-4 Socratic method for ITS.	Dialogue-based questions, dynamic lessons	Better grammar outcomes in English[64]. Engaging and strong adaptivity	Limited to language domain; not tested for coding.

Current GenAI tutoring frameworks employ large language models in different roles. Some frameworks embed an LLM into existing platforms to provide instant feedback on coding tasks[25]. Others build dedicated chat-based tutors that can engage in open-ended problem-solving dialogue with the student[24]. A common thread is that these frameworks leverage the natural language understanding and generation capabilities of LLMs to deliver explanations, hints, or corrections in a human-like manner. This greatly extends what traditional IDE autocompletion or static e-learning content can do. For instance, ITS-CAL's hint and debug modules allow it to walk a student through resolving a C bug step-by-step, mimicking how a human TA might guide the thought process[24].

Artemis's integration in an assessment system means students can get formative feedback on homework immediately, rather than waiting days for an instructor to

grade it[25]. Meanwhile, more experimental systems like TutorLLM combine LLMs with student modeling (tracking what a learner knows) to proactively recommend personalized practice material[26]. This hybrid approach addresses a limitation of out-of-the-box LLMs (their lack of memory about a specific learner) by introducing components that adapt to the individual's progress. There are also domain-specific adaptations; for example, OpenAI Codex a specialized GPT model for code that demonstrated how an LLM fine-tuned on programming data could generate correct C syntax and even explain code snippets[15].

Such advances indicate that an AI tutor for C programming can be built on a strong foundation: an LLM with knowledge of C language syntax/semantics, augmented by a framework that monitors student progress and context to steer the tutoring interaction. These frameworks show improved timeliness and personalization of feedback in programming education, but also underscore challenges like maintaining response quality (avoiding vague or misleading answers) and ensuring that students continue to think critically (mitigating over-reliance on AI hints)[25], [64]. These insights inform the design of a more robust LLM-based framework for personalized learning in C programming.

2.3.4 AI Tutoring Systems vs. LLM-Based Tutors

Intelligent Tutoring Systems (ITS) have a long history in computing education and traditionally follow a modular architecture. A typical ITS includes four core components: an Expert Module (domain knowledge), a Student Model (tracks learner's knowledge/state), a Tutor Module (pedagogical logic to decide what to do next), and a User Interface for interaction[64]. In a conventional tutor, domain experts predefine the curriculum content and rules for feedback. The system then uses the student model to select appropriate next problems or hints, following fixed algorithms. This rule-based approach can deliver effective, targeted instruction but is limited by the scope of its pre-authored content and logic. In contrast, LLM-based tutors leverage large pretrained models to generate responses on the fly, rather than relying solely on predefined rules. An LLM-based system may collapse or augment some of the traditional modules, for example, GPT-4 can function as both the Expert (with its vast trained knowledge) and part of the Tutor module (deciding what explanation to give).

This allows unprecedented flexibility: the tutor can answer novel questions or create new examples beyond a fixed script. Table 2.2 highlights some key differences.

Table 2.2: AI Tutoring Systems vs. LLM-Based Tutors

Aspect	Traditional ITS Tutor	LLM-Based Tutor
Content Generation	Uses pre-authored content and fixed hints[64].	Dynamically generates explanations, code, or hints.
Adaptability	Adapts within predefined scenarios (limited flexibility).	Highly adaptable; can handle open-ended queries and contexts[26].
Knowledge Coverage	Bound to specific curriculum/domain knowledge base.	Broad general knowledge from training data (e.g. many programming libraries)[68].
Integrity & Accuracy	Generally consistent if well-authored; no novel errors.	May produce incorrect or irrelevant output if prompt is ambiguous (hallucinations risk) [64], [66].
Development Effort	Requires significant expert authoring for each subject.	Requires prompt design and model tuning, but less manual content authoring.
Explainability	Transparent logic (rules are known); limited to built-in responses.	Opaque decision-making; can explain in natural language but reasoning is hidden in model weights.

Traditional ITS architectures are valued for their structured approach, the student model explicitly tracks mastery, and the tutor logic can be inspected and aligned with pedagogy. LLM-based tutors trade some of that explicit structure for greater flexibility and scale. They can carry on a free-form dialogue, answer a wide range of student questions, and even generalize to topics not anticipated by the designers[69]. For example, an LLM tutor might help with an unexpected coding error by drawing on general programming knowledge from its training corpus.

This flexibility, however, comes with challenges in maintaining coherence and truthfulness. The LLM might sometimes give an answer that is eloquently worded but subtly incorrect or not aligned with the curriculum[70]. Ensuring coherence (staying on topic and following logical instructional steps) and relevance to the learning

objectives is an active research area for LLM-based education. A study integrated GPT-4 into an existing tutoring framework to generate conversational scripts from lesson plans, speeding up content creation while still using a structured lesson outline [64]. Such approaches aim to get the best of both worlds: the rich, context-aware generation of LLMs plus the pedagogical reliability of traditional ITS design.

2.3.5 Explainability and Trust in LLM-Based Tutors

As the use of large language models (LLMs) in educational contexts grows, the issue of explainability how and why an AI system arrives at a given response becomes increasingly critical. Explainability directly affects the level of trust learners and educators can place in LLM-powered tutoring systems[68]. In traditional intelligent tutoring systems (ITS), logic and feedback mechanisms are rule-based and transparent, meaning that their decision-making pathways can be clearly traced and audited. However, LLMs operate as black-box models; their internal reasoning is embedded within millions of parameters learned from vast datasets, making it difficult to interpret how a specific answer was generated[71].

This opacity introduces several risks to educational use. First, LLMs are prone to generating plausible-sounding but incorrect responses commonly referred to as "hallucinations" which can mislead novice learners[72]. Second, even when LLMs provide accurate outputs, the lack of clarity on why a particular explanation or hint was chosen raises concerns about pedagogical soundness. Learners may be unable to verify or understand the underlying rationale, which can undermine their confidence and critical thinking development. The reviewed frameworks in Table 2.1 reflect these concerns. For instance, the Artemis GPT-3.5 tutor provides real-time feedback but occasionally delivers overly generic or inconsistent responses due to prompt sensitivity[73]. Similarly, TutorLLM combines an LLM with a student modeling component, yet its reliance on external retrieval mechanisms introduces complexity and challenges in maintaining pedagogical alignment.

Efforts to improve explainability in LLM-based tutoring systems include integrating self-explaining prompts, hybrid models that pair LLMs with rule-based scaffolding, and frameworks that log the rationale behind generated responses[66]. Retrieval-Augmented Generation (RAG) techniques are also being explored to ground LLM

outputs in verifiable sources, enhancing both transparency and factual accuracy [74]. These methods are essential to mitigating the risk of blind trust or over-reliance on AI tutors, especially among beginners in programming who may lack the expertise to detect subtle inaccuracies.

2.4 Empirical Literature

2.4.1 Learning Requirements in Computer Programming

Research into the learning requirements of computer programming students has long focused on identifying learners' needs, preferences, common challenges, and effective pedagogical strategies[63]. Numerous studies involving novice programmers have documented frequent difficulties such as understanding syntax, interpreting compiler errors, and managing the complexity of debugging tasks[63]. According to [75], students' preferences are often influenced by the programming language selected, which in turn shapes their learning trajectories. Through surveys and interviews, it has been observed that beginners demonstrate a strong preference for interactive and visually engaging learning environments[75]. These environments which are characterized by immediate feedback, graphical interfaces, and intuitive design, facilitate comprehension and active participation among novices. Although these studies provide valuable insights into beginner-level learning challenges and preferences, most rely on short-term evaluations and self-reported data. As a result, they often fall short in assessing long-term learning outcomes and the sustained effectiveness of the recommended instructional strategies.

2.4.2 Integration of Technology in Programming Education

Recent studies have increasingly focused on the integration of technology particularly Artificial Intelligence (AI) and Natural Language Processing (NLP) into programming education, aiming to improve learning outcomes and instructional efficiency. In one such study, [61] evaluated the performance of AI-assisted tutoring systems in providing personalized guidance and feedback to programming students. The findings confirmed that AI tutors can significantly enhance student performance and satisfaction, as evidenced by experimental results and learner feedback gathered through structured questionnaires[61]. These intelligent systems offer individualized coaching, targeted interventions, and adaptive learning pathways, enabling students to navigate complex programming concepts with greater confidence and effectiveness.

Building on this foundation, [76] explored the use of NLP techniques to automate the evaluation and feedback process for programming assignments. In leveraging advanced NLP algorithms, the researchers developed a system capable of analyzing student code submissions, identifying errors, and delivering detailed, context-specific feedback. The study demonstrated that such tools can increase grading efficiency, streamline the assessment process, and provide learners with timely, actionable insights[76]. As a result, instructors are able to reallocate more time to personalized instruction and mentorship, enhancing the overall quality of the learning experience. While these studies demonstrate the potential of AI and NLP technologies to improve programming education, they often overlook how these tools align with formal curricular objectives or how instructors adapt pedagogical strategies to integrate them effectively.

2.4.3 Generative Artificial Intelligence

The advent of generative AI (GenAI) has led to the emergence of an era of transformation that is characterized by the development of models like Generative Pre-trained Tranformer (GPT), Bidirectional Encoder Representations from Transformers (BERT), Gemini, and other remarkable technologies. The Generative AI is a lot more complex than the mere text comprehension. These models are able to compose code that has a serious impact on the teaching of programming by means of creating instructional content[15]. These generative AI tools have a skill of understanding and generating code snippets in various programming languages.

2.4.3.1 Transition to Generative AI in Learning Programming

Recent literature highlights the increasing interest in using generative AI to supplement or replace traditional programming learning materials. These models present a novel approach to content creation that aligns with diverse student needs by offering personalized, interactive, and contextually relevant learning resources[77]. Their dynamic nature allows for the real-time generation of examples and exercises tailored to individual learner profiles. This flexibility helps bridge the persistent gap between theoretical programming knowledge and hands-on skills development. Nonetheless, existing studies often focus more on the technological novelty rather than evaluating the sustained educational impact of these tools or their alignment with

curriculum goals, indicating a need for more empirical studies assessing long-term learner outcomes.

2.4.3.2 Automatic Generation of Programming Materials

The programming content creation is now experiencing a significant change due to the introduction of the automatic generation, which mainly results from the technological progress in AI. Many scientific works in this area have proved the meaning of the automatic generation of programming resources[57]. The text explains that there are lots of advantages to the automatic generation of the materials, but there are still some issues like the assurance of the accuracy and the quality of the information[7]. Figure 2.6 gives a general view of the procedure used for creating programming exercises. Studies consistently emphasize that while the speed of content delivery improves, the reliability and pedagogical quality of this content still require rigorous validation to ensure effective learning.

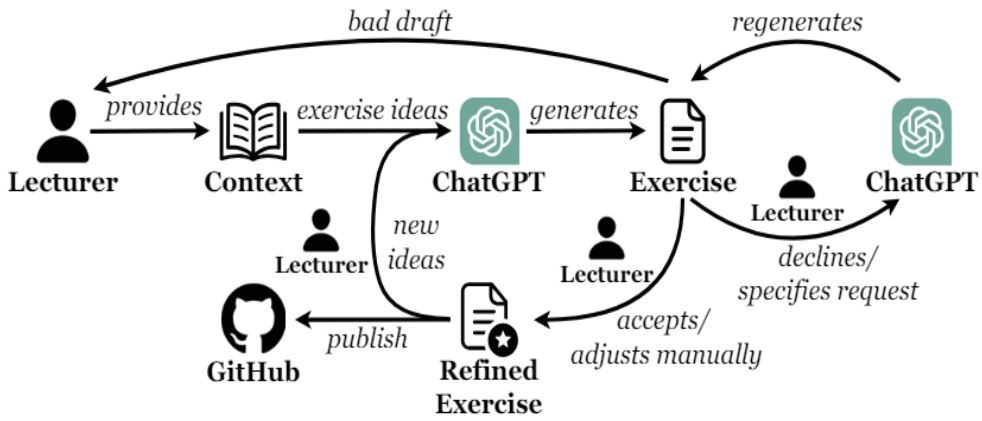


Figure 2.6: The Process of Generating Programming Exercises

2.4.3.3 Frameworks for Code Generation

The automation of the generation of programming materials has been the driving force of the creation of a variety of frameworks like OpenAI Codex that is designed to use this new approach to its full potential[15]. The discussion of the commonalities and variations of the methodologies that are the basis of these frameworks, with the focus on their adaptability to different phases of learning is given special attention[69]. Many existing AI frameworks for personalized learning, such as adaptive learning systems and intelligent tutoring systems, have shown potential in tailoring educational

experiences to individual students[69]. However, these systems often face challenges related to their ability to scale effectively across different educational contexts. Additionally, they sometimes struggle with providing content that remains coherent and contextually relevant for diverse learners.

2.4.3.4 Assessment of AI-Generated Programming Content

In the era of AI-generated programming content, the need for robust assessment and validation methodologies becomes paramount. The methodologies designed to evaluate the quality and effectiveness of content created through automated processes provides a valuable insight into best practices. They are employed to gauge the quality of content, encompassing factors such as accuracy, relevance, and adherence to programming standards[22]. This critical analysis sheds light on the strengths and potential shortcomings of existing assessment approaches, highlighting the complexities associated with evaluating dynamically generated materials. Existing LLM-based approaches such as those by GitHub Copilot and OpenAI Codex which generate and assess code without adapting explanations to learner profiles, tend to adopt a generalized perspective, often overlooking learner-specific factors such as prior knowledge, learning objectives, and individual learning trajectories. Moreover, there is limited integration of pedagogical considerations in assessing how such content supports novice programmers in understanding programming concepts or overcoming misconceptions.

2.4.3.5 Intersection of AI and Programming Material Generation

With the development of the AI-generated programming content, the requirement for the strong assessment and validation techniques becomes high. The methodologies intended for the assessment of the quality and effectiveness of content made through automated processes give a great insight into the best practices. They are used to evaluate the whole content quality which includes the accuracy, the relevance and the compliance to the standards of the program[16]. This pivotal analysis of the present concerted methods for evaluation of dynamically created materials exposes the strengths and possible weaknesses that are inherent in the current system, thus making clear the intricacies associated with the process of assessing such materials[16]. This highlights a critical gap in the literature and points to the need for developing adaptive

validation models capable of aligning AI-generated content with both instructional goals and learner competencies.

2.4.4 Empirical Insights on Personalization

Personalized learning approaches have increasingly been the focus of empirical studies in programming education, with strong evidence emerging to support their effectiveness in improving both student performance and engagement. In a comprehensive meta-analysis, researcher[43] synthesized findings from multiple studies evaluating individualized learning interventions in programming courses. The analysis revealed consistent benefits of personalized strategies, such as adaptive learning platforms and tailored feedback systems in enhancing learning outcomes and student involvement. These techniques were shown to cater to individual learning styles and needs, making the learning process more engaging and effective. However, much of the research in this area tends to prioritize short-term outcomes and lacks in-depth exploration of how these systems function over time across varied student populations.

Complementary research focused specifically on personalized learning platforms has further emphasized the value of adaptive instructional content and individualized learning pathways. Through experiments and learner evaluations, researcher [78] illustrated how such platforms enable students to progress at their own pace, receive support when needed, and build programming proficiency through contextualized, paced learning. These systems have demonstrated their capacity to enhance mastery of programming concepts. Despite these promising results, many existing studies do not fully account for implementation challenges or the need for broader curricular integration, highlighting the necessity for scalable and context-aware personalization strategies.

2.4.4.1 Impact of Large Language Models in Educational Settings

The application of Large Language Models (LLMs) in education has become an emerging area of empirical investigation. Studies have begun to assess the impact of these models on various educational processes, particularly in the realm of automated feedback and assessment. For example, research by [42] demonstrated the utility of LLMs in generating immediate, personalized feedback for programming assignments,

significantly improving both the speed and perceived quality of feedback delivery. Additionally, similar results have been reported in language learning environments, where LLMs were integrated into instructional systems and were found to support gains in both academic performance and language competence[69]. These studies suggest that LLMs are capable of facilitating meaningful educational experiences across diverse domains. Nonetheless, there remains a need to evaluate how consistently these outcomes are reproduced across various learning environments and student demographics.

2.4.4.2 Large Language Models for Code Generation

Large Language Models have proven particularly effective in generating code and providing programming-related explanations, owing to their training on extensive textual and code-based data. This capability has led researchers to explore their integration into programming classrooms, where LLMs offer benefits such as real-time code generation, error detection, and explanatory support[77]. AI-powered tools allow students to engage more interactively with course material, providing instant feedback, clarifying concepts, and automating routine tasks. These features can lead to improvements in code quality, student efficiency, and learning motivation[22], [69], [79]. However, empirical research into the educational effectiveness of these tools remains limited, with calls for more targeted studies to assess their impact on learning outcomes[78], [80], [81], [82].

Historically, code generation tools were confined to domain-specific languages and required formal specifications, which restricted their educational usability[83]. Recent advances in machine learning and deep learning have made it possible to use natural language-based models, such as GPT, for more intuitive and powerful code generation. While multiple studies, including reviews by [12], [84] have discussed the applications of GPT in programming, there is still a notable gap in understanding how such tools affect student learning processes and academic performance. LLMs are increasingly being adopted in classrooms to explain code snippets and assist with assignment generation, yet the pedagogical quality of these explanations is often inconsistent[15], [83], [85]. The present study builds on prior empirical work that has evaluated LLM-based instructional explanations, contributing new insights into their practical effectiveness in programming education.

2.5 Case Studies of AI-Assisted Programming Education

2.5.1 AI-Assisted Code Generation for Novice Coders

Researchers have explored how novice programmers interact with AI-based code generation tools such as OpenAI Codex in a self-paced learning environment. In a study, 33 students between the ages of 10 and 17 were observed using the tool to solve introductory programming tasks[86]. The research identified varied usage patterns, with some learners relying heavily on the AI to generate complete solutions, while others used it incrementally or for specific sub-tasks. The study found that students who relied on the AI to generate entire solutions achieved higher short-term task completion rates. However, these students struggled when asked to modify or expand their code independently, indicating limited understanding of the underlying programming concepts. Conversely, students who engaged in an interactive process with the AI demonstrated stronger comprehension and improved problem-solving skills. The findings from this study highlight an important gap in AI-assisted programming education: while AI can effectively enhance task completion, excessive reliance may undermine deeper learning and hinder the development of independent problem-solving skills.

2.5.2 AI-Generated Hints Versus Human Guidance

Researchers have investigated the effectiveness of AI-generated hints compared to human-provided guidance in an algebra-focused educational setting. They aimed to evaluate whether AI-generated hints could match the pedagogical value of human tutor feedback[87]. Their findings revealed that although AI-generated hints were generally accurate, they often lacked the contextual awareness required to tailor responses to individual student needs. The study further noted that AI-generated hints occasionally introduced technical language or concepts unfamiliar to the learners, resulting in reduced comprehension[87]. In contrast, students who received personalized guidance from human tutors demonstrated better problem-solving outcomes, as human tutors were better equipped to adjust their explanations to align with the student's level of understanding. These findings reveal a significant gap in current AI implementations for educational support. While LLMs can efficiently generate factually correct content, they lack the adaptive capabilities necessary to personalize learning experiences effectively.

2.5.3 AI-Generated Explanations for Programming Errors

In another study, researchers examined the use of GPT-based models to provide explanations for student coding errors in a computer science course. The study compared AI-generated explanations with those provided by human teaching assistants (TAs), with student participants evaluating the responses based on clarity, accuracy, and usefulness[88]. While AI-generated explanations were often clear and well-structured, they occasionally omitted critical details or introduced minor inaccuracies. In contrast, explanations provided by human TAs, although sometimes less polished, more consistently addressed the core issues underlying student errors. The findings from this study suggest that while LLMs can play a valuable role in reducing the burden on educators by automating feedback, their limitations necessitate additional validation steps[88]. An important gap identified in this research is the absence of mechanisms for ensuring the accuracy and completeness of AI-generated explanations.

2.5.4 Adaptive Explanations Using Prompt Engineering

Researchers have explored the potential of prompt engineering to improve the adaptability of AI-generated content in educational settings. Their study demonstrated that by embedding contextual information such as the learner's educational background, prior knowledge, and desired level of detail within the input prompt, AI-generated explanations could be tailored to meet individual student needs more effectively[89]. For instance, a high school student struggling with fundamental syntax errors received simplified explanations that emphasized intuitive analogies, while a university student was presented with technical explanations that incorporated detailed programming concepts. The success of this adaptive prompting approach highlights a key strategy for improving the personalization of AI-generated content[90]. However, the study also identified a significant gap in the scalability of this method. While prompt engineering can enhance personalization, developing effective prompts requires expertise in both educational psychology and prompt design, making it challenging for instructors without technical backgrounds.

2.5.5 AI-Generated Content for Programming Instruction

Several researchers have investigated the application of GPT-4 to generate instructional content for Python programming courses. Their study evaluated the

quality and accuracy of multiple-choice questions generated by the AI model when prompted to align with specific learning objectives. The results revealed that AI-generated questions were well-structured, appropriately challenging, and effectively aligned with key course objectives[91]. The ability of GPT-4 to generate clear explanations for correct and incorrect answers was particularly notable, reinforcing the tool's potential to support self-paced learning. Despite these positive findings, the study also revealed limitations regarding the pedagogical alignment of AI-generated content. While the questions accurately addressed coding concepts, they lacked the scaffolding necessary to guide learners through incremental skill development.

2.5.6 AI-Assisted Code Review and Debugging Tools

Several studies have examined the effectiveness of AI-driven code review and debugging tools such as GitHub Copilot, CodeWhisperer, and TabNine. These tools leverage LLMs to provide real-time code suggestions, identify syntax errors, and recommend improvements[85], [92], [93]. Research findings indicate that these tools can significantly improve coding efficiency by reducing the time students spend troubleshooting common syntax and logic errors. However, studies have also shown that while AI tools excel at providing fast feedback on common coding mistakes, they may misinterpret complex logic flaws or produce ambiguous recommendations when faced with unconventional coding patterns. The primary gap identified in this area relates to the potential for students to develop over-reliance on AI-generated feedback, which may diminish their ability to engage in independent problem-solving.

2.6 Analysis of Key Features and Best Practices

2.6.1 Adaptability

Adaptability refers to the ability of AI systems to adjust learning materials, feedback, and support mechanisms in response to individual learner progress, proficiency, or behavior. In the context of personalized programming education, adaptability is crucial for ensuring that students receive content aligned with their current skill level, cognitive needs, and learning trajectory.

Adaptability in educational systems is grounded in the principles of learner-centered pedagogy and formative feedback, where instruction dynamically evolves in response to real-time learner inputs. In GenAI-based frameworks, adaptability often manifests

through Automated adjustment of content difficulty (e.g., beginner to advanced tasks), context-sensitive hint generation based on learner errors, and progress-aware feedback and learning path modifications. These approaches aim to minimize cognitive overload, sustain engagement, and promote mastery through timely, personalized support. For instance, adaptive learning systems such as Multilingual Code Co-evolution were found to dynamically adjust programming exercises to accommodate different linguistic contexts, thereby enhancing engagement and accessibility[94].

Reviewed studies provided diverse implementations of adaptability. Studies by [29], [73] described frameworks that monitored student performance in real time and altered the complexity of coding tasks accordingly. Similarly, [95], [96] implemented adaptive feedback loops where the AI tutor offered progressively targeted hints depending on user interaction patterns. The other article, [97] demonstrated how GenAI could personalize learning materials based on prior performance history, learner goals, and observed challenges in syntax and logic.

2.6.2 Contextual Relevance

Contextual relevance refers to the ability of GenAI systems to align learning materials, explanations, and tasks with the learner's specific goals, interests, curriculum content, and real-world programming scenarios. It enhances learning effectiveness by ensuring that AI-generated outputs are not only syntactically correct but also pedagogically meaningful and practically applicable to the learner's context. Frameworks such as the Generative AI-Based Personalized Guidance Tool focused on delivering tailored content aligned with learner objectives, significantly enhancing student engagement and real-time feedback integration[73].

Rooted in situated learning theory and constructivist pedagogy, contextual relevance emphasizes the importance of anchoring educational content in authentic and relatable experiences. In programming education, this involves tailoring exercises, examples, and feedback to: match course-specific objectives, reflect real-world programming applications and align with learner projects or domains of interest. This alignment fosters deeper engagement, enhances problem-solving skills, and promotes transfer of learning to practical settings.

Contextual relevance is a recurring feature in frameworks aiming to personalize content beyond surface-level correctness. Articles by [73], [96] reported systems that adjusted feedback based on students' learning paths and career-oriented goals. Article [6] highlighted the value of embedding context-specific examples to overcome common conceptual barriers in programming. Articles [61], [98] illustrated how multimodal or domain-specific prompts improved understanding by situating coding tasks in relevant, real-life problems. Again, [99] emphasized debugging assistants and tutoring tools that customized guidance based on the learner's current codebase and logical flow.

2.6.3 Clarity and Coherence

Clarity and coherence refer to the degree to which AI-generated outputs maintain a logical structure, semantic clarity, and progressive reasoning that is comprehensible to learners. In programming education, coherence ensures that each explanation follows a meaningful flow, enabling students to grasp concepts sequentially and avoid confusion due to disjointed or contradictory statements. Techniques such as Chain-of-Thought prompting and Modular Prompt Engineering found to enhance the logical flow of AI-generated explanations, particularly in complex programming exercises[100].

Clarity and coherence are grounded in cognitive load theory and instructional design principles, which stress the importance of well-structured information to facilitate understanding and long-term retention. Within the context of LLM-powered education, these principles translate to: step-by-step breakdowns of programming logic, avoidance of abrupt topic shifts or contradictions, and maintenance of semantic consistency in generated code and explanations. When coherence is lacking, students especially beginners can misinterpret logic, overlook dependencies, or misapply syntax patterns, leading to conceptual errors. Several studies explored methods to enhance coherence in GenAI systems: researcher [101] introduced structured prompting techniques to ensure logical consistency in explanations and code annotations. Articles by [102], [103] focused on developing metrics for assessing the coherence of AI-generated outputs in programming contexts. Researchers [99], [104] implemented chain-of-thought prompting to break down problem-solving processes into logical sequences that mirror human reasoning patterns. Also, [105] reported that

fine-tuning LLMs with curated educational data led to more coherent and linear explanations, especially in multi-step programming tasks.

2.6.4 Accuracy and Reliability

Accuracy and reliability refer to the extent to which AI-generated outputs particularly code, explanations, and feedback are logically correct, free of syntax and semantic errors, and consistent with programming standards. Reliability also entails the system's ability to consistently produce correct outputs across multiple prompts or contexts. In programming education, accuracy is not optional but rather essential. Errors in AI-generated code can mislead learners, introduce incorrect logic, and hinder concept mastery. This makes reliability a non-negotiable feature in educational applications of GenAI. The foundation of this feature lies in the verification mechanisms for ensuring code correctness, robust evaluation benchmarks, and error detection and debugging support within AI explanations. Inaccuracies, whether syntactic (e.g., missing brackets) or semantic (e.g., flawed logic), can erode learner trust and result in misconceptions that are difficult to unlearn.

The reviewed studies employed several approaches to improve the accuracy and reliability of GenAI systems. Frameworks like LEVER (Learning to Verify Language-to-Code Generation) were found to enhance the reliability and syntactic correctness of AI-generated code by using verification mechanisms to reduce errors[83]. The researcher introduced a verification-enhanced framework that uses a language-to-code validator to reduce logical errors in AI-generated programming solutions. Researchers [91], [106] conducted benchmarking analyses of popular LLMs, identifying persistent challenges in producing functionally accurate code. Researcher [107] used auto-feedback mechanisms to compare student submissions against verified solutions, promoting accuracy through iterative refinement. Researchers [108], [109] explored few-shot learning and code evaluation frameworks that guided the AI toward producing contextually correct answers. Also, [105] emphasized that fine-tuning LLMs on curated educational data significantly improved the precision of syntax and logic across programming tasks.

2.6.5 Prompt Engineering Effectiveness

Prompt engineering refer to the strategic crafting, refinement, and adaptation of prompts used to guide AI output toward accurate, relevant, and pedagogically useful responses. Effective prompt engineering enhances not only the quality of AI responses but also their coherence, relevance, and personalization to individual learner needs. In the context of LLM-powered educational tools, prompt engineering functions as a control interface that bridges human intentions and AI outputs. Well-structured prompts help the AI to understand learner intent, follow a logical reasoning path (e.g., chain-of-thought), deliver scaffolded or step-wise explanations, and tailor responses based on prior learner input or performance. Prompt engineering effectiveness is grounded in instructional design, cognitive scaffolding, and dialogic learning all of which benefit from clear, targeted, and context-aware instructions.

A growing body of work has demonstrated the importance of prompt engineering in personalizing programming education. Structured and iterative prompt techniques were found to enhance output quality by guiding the AI's reasoning process and ensuring logical consistency[110]. From the review, Studies such as Smart Prompt Advisor demonstrated the effectiveness of structured prompts in refining AI-generated explanations and improving output relevance and coherence[101]. Additionally, Adaptive Prompts were found to be particularly effective in personalized programming education, as they dynamically adjusted to individual learner needs[111]. Researchers like [103] cataloged reusable prompt patterns that were shown to produce more accurate and coherent outputs across diverse coding tasks. Researchers, [104], [108] explored chain-of-thought and few-shot prompting, respectively, to enhance step-wise problem solving and contextual understanding. Others like [105], [112] reported that refining prompts based on learner interaction history helped improve relevance and eliminated unnecessary verbosity or semantic drift.

2.6.6 Practical Usability

Practical usability is the extent to which GenAI-powered educational systems are accessible, intuitive, and readily usable by learners and instructors within real-world academic environments. Usability encompasses several dimensions, including ease of navigation, clear system feedback, compatibility with diverse devices, and integration

into existing learning management workflows. Practical usability in educational technologies is grounded in the principles of human-computer interaction (HCI), universal design for learning (UDL), and usability engineering. In programming education, a usable system allows learners to seamlessly access and complete coding tasks, interpret AI feedback without confusion, interact with the system across different platforms (e.g., browser-based IDEs), and avoid distractions caused by system errors or poorly designed interfaces. A highly usable GenAI framework increases student adoption, reduces cognitive friction, and enhances the overall learning experience.

The high prevalence of this feature in the literature reflects a growing recognition that even the most powerful AI models are ineffective if learners find them difficult to use. Researchers [62], [113] focused on multilingual and mobile-friendly interfaces to improve accessibility across learner groups. Articles by [114], [115] described user-centered interface designs embedded in LMS environments to reduce tool fragmentation. Researchers [99], [107] integrated AI feedback directly into student dashboards, allowing immediate and actionable guidance. Researcher [105] demonstrated that simplifying response formats (e.g., using inline comments or error highlights) improved clarity and reduced learner dropout. Again, [116] introduced scaffolding tools and interface cues that reduced the need for constant instructor support. Practical usability ensures that GenAI systems are not only pedagogically sound but also technically accessible and easy to use a crucial factor in any educational deployment.

2.6.7 Interactivity and Collaboration

Interactivity and collaboration refer to the design of systems that not only respond meaningfully to user input (interactivity) but also support social learning, peer discussions, or group-based coding tasks (collaboration). Together, these aspects promote active learning and community-driven problem solving, especially important in programming education. These features are rooted in social constructivist theory, dialogic learning, and interactive system design. Interactivity helps learners feel immersed in the learning process, receiving real-time responses to their actions, while collaboration encourages knowledge co-construction, peer scaffolding, and collective problem-solving. In GenAI contexts, this may involve conversational interfaces for

Q&A, multi-user shared workspaces for coding, discussion threads integrated with AI-generated content, and context-aware feedback that responds to both individual and group submissions.

Numerous studies incorporated interactive and collaborative elements into GenAI-based programming environments. Researchers [3], [73] implemented interactive chat-based agents that provided contextualized, real-time help. Others like [22], [117] described collaborative code editors where learners could co-author, annotate, and review programming solutions. Studies by [8], [107] reported increased engagement when systems allowed two-way communication and feedback loops. Researcher [109] emphasized scalable interactivity, enabling learners in large classes to benefit from AI-mediated collaborative exercises. It is evident from the review that [118] introduced visual collaborative prompt-building interfaces that allowed learners to build prompts together and reflect on prompt effectiveness.

2.6.8 Creativity and Uniqueness

Creativity and uniqueness refer to the AI system's ability to generate original, diverse, and thought-provoking content, such as novel code examples, multiple solution paths, or alternative approaches to problem-solving. It emphasizes not only correctness, but also innovation and exploration, which are critical in nurturing higher-order thinking skills in programming students. In educational contexts, creativity aligns with constructivist and exploratory learning paradigms, where learners are encouraged to discover, adapt, and critique solutions rather than merely follow fixed templates. Uniqueness in GenAI systems involves generating diverse responses to similar queries, proposing unconventional or optimized algorithms, encouraging learners to reflect on alternative coding approaches, and avoiding repetitive or overly generic solutions. Creativity supports engagement, flexible thinking, and transferable problem-solving skills which are essential in programming education.

While not the primary focus of most frameworks, several studies acknowledged the value of creativity as a byproduct or design goal. Studies [15], [103] emphasized prompt variation techniques that led to multiple valid outputs, encouraging student exploration. Researcher [119] investigated how brainstorming-style prompting improved the diversity and novelty of AI-generated code while [91] benchmarked

LLMs on their ability to propose non-trivial solutions to standard programming challenges. Others like [51], [116] highlighted how adaptive hinting and error analysis led learners to discover alternative logic structures, thereby enhancing creativity through experimentation.

2.7 Gap Identification

Despite the rapid advancement of AI in education, existing literature reveals notable gaps that the present study aims to address. Firstly, many studies exploring generative AI have limited access to individualized tutoring and minimal exposure to adaptive technologies[7], [36], [37]. Secondly, much of the current research emphasizes surface-level engagement metrics like learner satisfaction or usage frequency, but offers limited evidence on actual personalization effectiveness particularly in C programming, which is inherently difficult due to its low-level syntax and abstract concepts[2]. Most frameworks fail to align feedback, project scaffolding, or assessments with learners' individual competencies, prior knowledge, or preferred learning styles[16], [21], [40]. Thirdly, explainability and trust in LLMs remain underexplored. Although LLMs can dynamically generate feedback and suggestions, they often operate as black-box systems, raising concerns over opaque reasoning and inconsistent pedagogical alignment[120]. This undermines both learner trust and instructional transparency, especially in high-stakes environments such as formal university education.

Finally, while some frameworks integrate personalized elements like adaptive pacing or topic selection, they often lack a structured conceptual model that integrates learner characteristics, instructional design, and ethical considerations[73]. There is insufficient guidance on how these variables interact within GenAI-powered tutoring systems to enhance learning outcomes in programming. This study seeks to bridge these gaps by developing and evaluating a structured, ethically grounded LLM-based framework that offers real-time, personalized, and explainable learning support for C programming students in a Kenyan university context. The framework integrates theories such as Self-Determination Theory (SDT) and differentiated instruction, and explicitly considers learning style, prior programming experience, and data privacy dimensions that are rarely addressed in a unified manner in current literature.

2.8 Conceptual Framework

This study is guided by a conceptual framework that explains how instructional strategies and AI-driven interventions particularly those powered by Large Language Models (LLMs) can enhance personalized learning in C programming education. The framework has been structured around four key components: learner characteristics, LLM framework features, pedagogical approaches, and ethical considerations. These components interact dynamically to achieve the study's objective of improving learner engagement, comprehension, and satisfaction through adaptive and personalized instruction.

The first component, learner characteristics, represents the moderating variables that influence how effectively personalized instruction can be delivered. These characteristics include students' prior programming experience, preferred learning styles (such as visual, auditory, or kinesthetic), and levels of motivation and engagement. Understanding these differences is essential for tailoring instructional content and ensuring the relevance and accessibility of learning materials. Moderating variables such as learner profiling based on individual performance, background knowledge, and preferences plays a critical role in informing how adaptive features are implemented.

The second component involves the core features of the LLM framework, which serve as the independent variables in this study. These include prompt engineering, adaptive content delivery, interactive tutoring, real-time feedback, and collaborative coding. Prompt engineering allows for the generation of learner-specific instructional content by crafting precise, context-aware input queries. Adaptive learning mechanisms adjust task complexity and content based on learner performance data, ensuring that students receive appropriate levels of challenge and support. The framework also includes interactive tutoring features, where learners engage in natural language conversations with the LLM to receive explanations or coding guidance. Real-time feedback enables students to quickly correct mistakes and reinforce their understanding, while collaborative coding fosters peer learning and engagement through joint problem-solving activities.

Pedagogical approaches represent the third component of the framework and serve as the theoretical foundation for the instructional strategies applied. Drawing on

constructivist learning theory, the framework encourages students to build their own understanding through exploration and guided discovery. Scaffolding techniques are embedded within the framework, offering structured support that is gradually reduced as learners gain competence and confidence. Collaborative learning is also emphasized, with opportunities for learners to interact, share ideas, and solve problems together, promoting deeper understanding and increased motivation.

Finally, ethical considerations are integrated into the conceptual framework to ensure the responsible design and deployment of the AI-powered instructional system. While this study does not focus on evaluating ethical dimensions in depth, it acknowledges the importance of data privacy, bias mitigation, and equitable access as essential elements of a trustworthy learning environment. Measures to protect learner information and ensure confidentiality are considered, along with strategies to reduce biases in LLM-generated content and to ensure that all students regardless of background can benefit from the framework's features.

These four components work together to achieve the dependent variable of this study: enhanced personalized learning. This is operationalized in terms of increased student engagement, improved comprehension of C programming concepts, greater satisfaction with the learning process, and reduced frustration among undergraduate learners. The interaction between learner characteristics and LLM features ensures that adaptive mechanisms respond effectively to individual needs, while pedagogical principles guide the application of these technologies in a manner consistent with best practices. Ethical considerations underpin the entire framework, ensuring that personalization efforts are conducted fairly and responsibly.

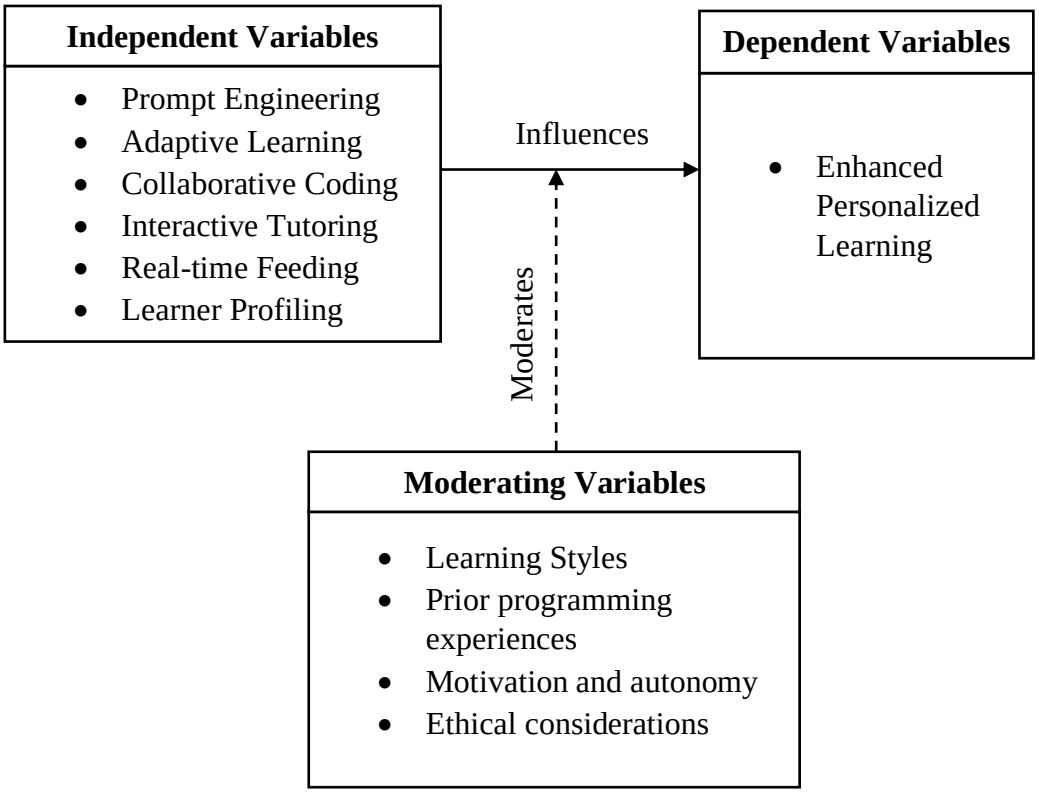


Figure 2.7: Conceptual Framework

2.9 Summary

This chapter reviewed existing literature on personalized learning in programming education and the role of generative artificial intelligence, particularly large language models, in enhancing instructional experiences. It began by establishing the theoretical foundations of personalized learning and examining how individual learner profiles, prior knowledge, and pacing needs influence programming education outcomes. A detailed review of existing LLM-based educational frameworks was conducted and analyzed based on its features, outcomes, and limitations, identifying gaps such as over-reliance, limited adaptivity, and lack of fine-grained control. The chapter concluded by identifying critical instructional elements prompt engineering, contextual relevance, coherence, personalization, and feedback that should inform any GenAI-powered framework for programming education. These insights provided the foundational blueprint for developing the LLM-based personalized instructional framework proposed in this study.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

This chapter presents the methodology adopted in the study, describing the research design, research process, target population, and the implementation of the framework in a classroom setting. It also outlines the data collection instruments, as well as their reliability and validity. Additionally, this chapter details the experimental materials, data analysis techniques, and ethical considerations that were observed during the study. In systematically outlining the methodological approach, this chapter ensures the credibility, reliability, and reproducibility of the research findings.

3.2 Research Design

Research design refers to the overall strategy that guided the researcher in collecting, analyzing, and interpreting data. This study employed a mixed-methods research design, integrating both qualitative and quantitative approaches to ensure a holistic exploration, development, and evaluation of a personalized learning framework for introductory C programming using Large Language Models (LLMs). The rationale for adopting this design lies in its ability to capture both the contextual depth offered by qualitative data and the statistical generalizability afforded by quantitative analysis. This dual approach provides a more comprehensive understanding of the research problem than either method alone could offer.

The mixed-methods framework aligns with the study's three objectives. First, a Systematic Literature Review (SLR) was conducted using qualitative methods to analyze existing models and technologies in personalized programming instruction. This review established the foundational insights and conceptual gaps that informed the framework design. Second, the Design Science Research (DSR) paradigm was adopted to guide the creation of the LLM-based instructional framework. DSR is particularly suitable for studies focused on innovative artifact development, as it supports iterative design, implementation, and evaluation phases.

Design-Science research is built around three iterative activities namely problem identification, solution design, and evaluation[121]. This makes it ideally suited for a

study that first diagnoses gaps in personalized programming education, then designs an LLM-based instructional framework, and finally evaluates it in a classroom setup. Third, a controlled implementation and evaluation phase was conducted using both qualitative tools (e.g., thematic feedback analysis) and quantitative instruments (e.g., pre/post tests, Likert-scale surveys) to assess the framework's pedagogical impact and technical reliability.

The research adopted a mixed-methods research design, selected to leverage the complementary strengths of qualitative and quantitative approaches, ensuring a comprehensive assessment of both the framework's technical fidelity and its pedagogical impact[122]. The qualitative component of the study involved systematic synthesis of literature to gain in-depth insights into user needs, learning preferences, and perceptions of AI-assisted programming education[123]. Qualitative data were analyzed using thematic analysis, which facilitated the identification of recurring themes and patterns, providing a deeper understanding.

The quantitative component of the study aimed to objectively measure the effectiveness of the developed framework in enhancing programming education. This was achieved through structured surveys and pre-post assessments, which allowed for statistical analysis of learning outcomes before and after using the AI-driven learning framework. In combining both qualitative and quantitative methods, the study achieved a balanced and rigorous analysis of the research problem[122]. The initial qualitative findings from the surveys informed the design of quantitative surveys, ensuring that survey questions effectively addressed key themes identified during the qualitative phase.

Qualitative data provided rich contextual insights into student experiences, challenges, and preferences, capturing nuances that would be difficult to quantify[123]. However, qualitative methods alone lacked the statistical rigor and generalizability needed to assess the framework's impact across a larger population. To address this limitation, quantitative analysis was used to validate and expand upon qualitative themes, ensuring statistical robustness and broader applicability of the findings.

This mixed-methods research design ensured a comprehensive, systematic, and evidence-driven approach to addressing the research questions. The integration of qualitative and quantitative data facilitated a deeper understanding of user needs, enabled the empirical development of an AI-based learning framework, and ensured robust evaluation of its effectiveness[122]. To synthesize qualitative insights with quantitative metrics, the study enhanced the credibility and reliability of the research findings, ultimately contributing to the development of a well-founded AI-assisted framework for personalized programming education.

3.3 Research Process

The research process outlined the procedures followed to achieve the study's objectives. This section details the methodological approach undertaken to address each research objective, ensuring a structured and systematic investigation of the effectiveness of the developed framework in personalized programming education. The first specific objective was to answer a research question on determining features and best practices in existing LLM-based frameworks that support effective personalized learning in programming education. A systematic literature review was conducted as described in section 3.5 to answer this research question. The output from the first specific objective that is, the identified features and best practices where validated through an expert opinion survey to ensure reliability. These features where necessary as they informed the second specific objective.

The second specific objective was to develop a personalized programming instruction framework using the features identified from the existing LLM-based tools. This development was informed by the findings of the systematic literature review and expert validation conducted in the first phase. The framework was designed with a modular architecture to support real-time interaction, content personalization, and learner performance tracking. In addressing gaps highlighted in the literature, the framework also incorporated enhancements to mitigate known limitations of transformer-based models, such as hallucination and lack of contextual alignment. The resulting artifact, described in detail in Section 3.6, served as the main output of this phase and laid the foundation for empirical evaluation in the next stage of the study.

The third specific objective was to evaluate the effectiveness of the proposed framework in improving student engagement, learning outcomes, and the quality of instructional content in programming education. This objective was addressed by conducting a controlled classroom study, where the developed framework was deployed alongside traditional instructional methods. The evaluation focused on quantifiable metrics such as pre- and post-test performance and structured questionnaire responses measuring learner engagement and content quality. Additionally, qualitative feedback was gathered to capture user perceptions and contextual insights into the learning experience. Section 3.7 provides a detailed account of the evaluation procedures, while Figure 3.1 illustrates the overall research process flow across the three specific objectives.

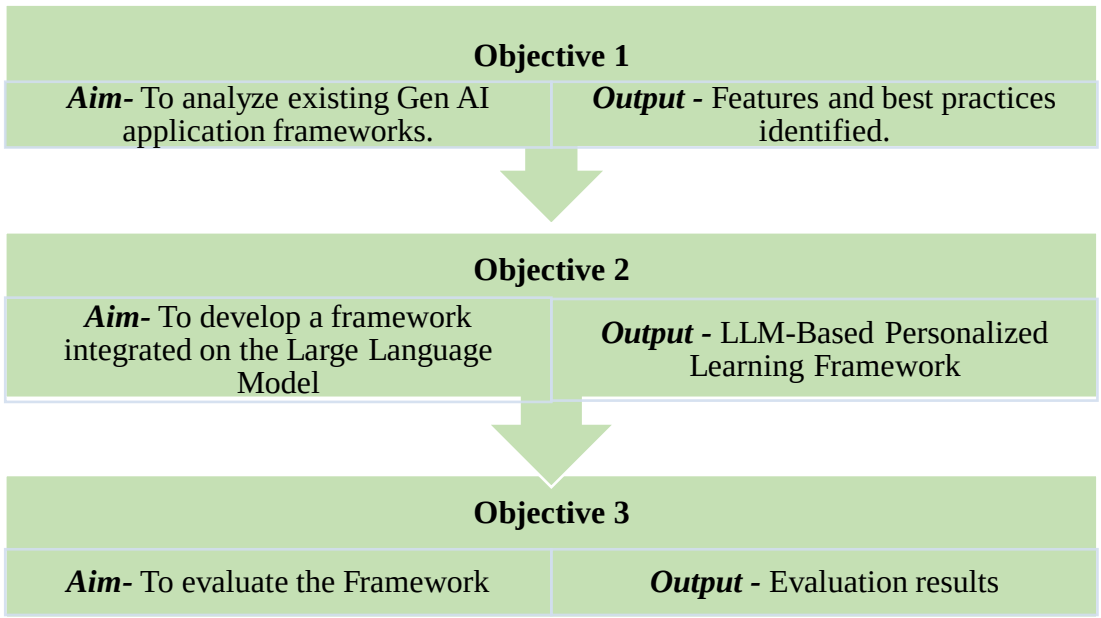


Figure 3.1: Research Process

Table 3.1 shows a summary of objectives, method used to address the objectives, whether the method is qualitative or quantitative, and the analyses technique used.

Table 3.1: Summary of Research Objectives

S/No.	Objective	Method Used	Nature of Method	Analysis Technique
1	Analyze existing LLM-based frameworks and tools	Systematic Literature Review and Expert validation	Qualitative	Thematic analysis and content synthesis
2	Develop a personalized LLM-based programming framework	Design Science Research (Artefact Development)	Qualitative	Iterative prototyping, expert reviews, and design validation
3	Evaluate the effectiveness of the developed framework	Quasi-Experimental Study (Pre-test and Post-test)	Mixed Methods	Descriptive statistics, Mann-Whitney U test (quantitative); Thematic analysis (qualitative)

3.4 Participants and Data Collection Procedures

This section outlines the procedures used to identify study participants, determine sample size and sampling strategies, and collect data in alignment with the study's objectives. Participants were drawn from two primary groups: expert stakeholders for feature validation and undergraduate students for framework evaluation. Each subgroup was selected using methodologically sound approaches, and data were gathered using validated instruments tailored to their respective roles. The following subsections detail the target population, sampling methods, and data collection procedures.

3.4.1 Target Population

The target population for this study comprised two distinct groups: undergraduate students enrolled in computing-related degree programs at Murang'a University of Technology, and expert stakeholders engaged in programming education and practice. These groups were selected to support the dual focus of the study, developing and validating a personalized instructional framework and evaluating its effectiveness in an authentic learning environment.

The primary student population included undergraduate learners undertaking the Introduction to Computer Programming course during the September–December 2024 semester. This group was selected based on two key criteria: (1) foundational exposure to programming concepts, enabling meaningful engagement with the AI-powered framework, and (2) the course's relevance to the study's aim of enhancing personalized learning in early-stage programming education. Targeting this cohort ensured that participants possessed sufficient background knowledge to benefit from and critically interact with the learning tool, without introducing bias from advanced programming expertise.

The expected population size was set at 225 participants, based on historical enrollment data for the course across multiple computing degree programs. This number was chosen based on an estimated minimum class size of 45 students per computing degree program across the five programs offered in the School of Computing and Information Technology, in alignment with typical enrollment trends and academic planning benchmarks reported in educational research. This sample size was deemed appropriate to ensure statistical power for comparative analysis between experimental and control groups, support subgroup representation (e.g., based on prior programming experience or learning style), and accommodate potential attrition while maintaining the validity and generalizability of the study findings. This number was considered sufficient to allow for the formation of two balanced cohorts, the experimental and the control groups, while preserving statistical validity for subsequent comparative analysis. Stratified participation was encouraged to reflect diversity in learning styles, prior programming experience, and academic backgrounds, thereby enhancing the representativeness of the study sample.

To address potential challenges in access and engagement such as varying attendance levels, schedule conflicts, and differing levels of digital fluency the study collaborated closely with course instructors and departmental administrators. Flexible participation schedules, instructor-led awareness sessions, and the integration of the framework into official classroom activities were adopted to encourage participation. These measures ensured that the selected participants formed a representative subset of the broader student population, thereby strengthening the validity and generalizability of the research outcomes.

In addition to the student participants, the study also targeted a population of expert stakeholders for the feature validation phase. This group consisted of current senior computing students, recent graduates, university lecturers involved in programming instruction, and ICT professionals with practical experience in software development or AI-assisted education tools. Each subgroup was purposefully selected for their unique perspective: senior students contributed current user experience, recent graduates provided reflective insights, lecturers offered pedagogical evaluation, and industry experts ensured alignment with practical standards. This diverse but focused group provided the necessary depth and breadth to assess the relevance, clarity, and applicability of the proposed framework features. Inclusion of both learners and experts ensured that the framework was shaped by pedagogical relevance, usability, and practical feasibility.

To address the heterogeneity of the target population, comprising both experts (e.g., lecturers, senior students, and graduates) and novices (first-year undergraduates with little or no C programming experience), the study employed stratified sampling, followed by purposive selection within each group. This approach ensured that both subgroups were proportionately and meaningfully represented in the evaluation phase. Stratification controlled for differences in prior programming knowledge, allowing the research to capture diverse perspectives on the framework's usability and effectiveness across varying learner profiles.

3.4.2 Sample Size and Sampling Strategy

This study employed a two-phase sampling strategy aligned with the sequential structure of the research objectives. The sampling approach was guided by the need to gather both expert feedback for framework development and valid learner data for framework evaluation. Purposeful and stratified random sampling methods were used to ensure the selection of participants with relevant expertise and learning contexts, while also maintaining representativeness and analytical rigor.

For the expert validation phase, a sample of 20 participants was selected to validate the features and best practices extracted from existing LLM-based tools. According to prior research, qualitative studies typically require 10 to 50 participants, depending on the research question and methodology[123]. The selection of 20 participants allowed

for detailed analysis while maintaining a manageable sample size that facilitated deeper engagement with participants. These participants were chosen using purposive sampling strategy, based on their prior experience in programming education and exposure to AI tools in learning. The sample consisted of six current third- or fourth-year students, six recent computing graduates, six university lecturers actively involved in teaching programming, and two ICT industry experts with experience in AI-enhanced tools. This diverse composition ensured that the feature validation incorporated perspectives from both learners and practitioners across academic and industry settings. The expert sample size was guided by prior literature on opinion-based surveys and was sufficient to achieve content validation while remaining manageable for in-depth analysis.

In the framework evaluation phase, a separate sample of 139 undergraduate students enrolled in the 'Introduction to Computer Programming' unit of study at Murang'a University of Technology during the September–December 2024 semester was selected. These students formed the study's core participants for assessing the effectiveness of the developed framework. Stratified random sampling was employed to ensure representation across learning styles, academic performance levels, and gender, while maintaining comparability between groups. The students were randomly assigned to either a control group (traditional learning methods) or an experimental group (using the LLM-based framework). This random allocation controlled for confounding variables and enhanced the internal validity of the study by ensuring that any observed differences in learning outcomes were attributable to the intervention rather than individual characteristics.

The sample size for the evaluation phase was informed by statistical power analysis, targeting a minimum threshold required to detect significant differences using non-parametric tests such as Mann-Whitney U test. At the same time, the size allowed for practical feasibility within the institutional context. Consideration was also given to the principle of data saturation in qualitative feedback collected from the experimental group, whereby data collection continued until no new themes emerged[123]. Through this dual sampling strategy, the study ensured methodological robustness and stakeholder inclusivity, strengthening the reliability and applicability of its findings.

3.4.3 Data Acquisition

This study employed both quantitative and qualitative data collection methods to comprehensively evaluate the effectiveness of the developed LLM-based personalized programming framework. Data were collected in two distinct phases, expert validation and classroom evaluation. The total number of participants involved in data collection was 159, comprising 20 expert participants in the validation phase and 139 participants in the evaluation phase.

3.4.3.1 Expert Opinion Survey

For the validation of features and best practices identified from the systematic literature review, structured expert opinion questionnaires were used. The questionnaire contained a 5-point Likert-scale items and a few open-ended questions to allow for additional feedback. These items were directly derived from the feature categories synthesized in the review. A sample of this expert questionnaire is provided in the appendix section. The expert questionnaire was validated using content validation and internal reliability testing. Content validity was established through peer review by two senior faculty members who assessed the relevance and clarity of each item. To ensure internal consistency of the instrument, a Cronbach's Alpha test was conducted during a piloting phase conducted with five participants to check on the reliability of the questionnaire.

3.4.3.2 Framework Evaluation

In the second phase, a structured questionnaire was administered to a sample of 75 undergraduate students who participated in the framework evaluation. The purpose of this instrument was to collect data on key indicators such as student engagement, perceived usability, personalization, content clarity, and overall satisfaction with the GenAI-powered programming framework. The questionnaire was administered after the students interacted with the framework in a classroom setting. The instrument comprised two sections. Section A featured 15 closed-ended items rated on a 5-point Likert scale (ranging from 1 = Strongly Disagree to 5 = Strongly Agree). These items were designed to assess constructs such as usability, motivation, adaptability, and perceived effectiveness. Section B included open-ended questions that invited students to share reflections on their learning experiences such as which programming

concepts became clearer, challenges encountered, and suggestions for improving the tool.

The questionnaire used in the framework evaluation was constructed by adapting items from established survey instruments previously used in AI-in-education research. The complete questionnaire is provided in the appendix section. To ensure reliability and clarity, the questionnaire was pilot-tested with 10 students who were not part of the main study. The pilot responses were analyzed using internal consistency techniques, specifically Cronbach's Alpha, as a measure of reliability. The feedback from the pilot phase also informed minor adjustments to item wording and structure. The final questionnaire was administered using Google Forms, immediately after the intervention sessions. Clear instructions and researcher guidance were provided to reduce confusion and ensure meaningful responses. This questionnaire instrument formed the sole basis for data collection in the framework evaluation phase, providing both quantitative and qualitative insights for assessing the framework's effectiveness in enhancing personalization in programming education.

3.4.3.3 Data Collection Procedures and Ethics

Data collection was carried out over a six-week period. For the expert phase, surveys were distributed electronically via Google Forms, and reminders were issued to maximize response rates. During the framework evaluation phase, questionnaires were administered at the start and end of the intervention sessions, and qualitative data collection took place in the two weeks following the classroom trial. The two-week duration was selected based on the academic calendar constraints and the structured timing of the course's lab sessions, ensuring that the intervention could be seamlessly integrated into ongoing class activities without disrupting the syllabus. To address potential challenges such as low response rates and scheduling conflicts, the study offered flexible data collection times and multiple formats (online and in-person) based on participant availability. Clear instructions accompanied all instruments to reduce the risk of incomplete or misunderstood responses.

All data collection procedures were conducted in compliance with ethical research guidelines. Participants were informed of their rights, assured of confidentiality, and asked to sign an informed consent. No personally identifiable data were collected or

stored. In integrating both quantitative and qualitative data collection methods across both expert validation and classroom deployment phases, this study ensured a robust and multi-perspective evaluation of the developed LLM-based learning framework. The use of validated instruments, appropriate sampling strategies, and ethical safeguards contributed to the overall validity, reliability, and credibility of the findings.

3.4.4 Data Analysis

The data analysis process integrated qualitative and quantitative approaches to comprehensively evaluate the effectiveness of the LLM-based personalized programming framework. Qualitative data, collected from open-ended survey responses and participant reflections, were analyzed using NVivo through thematic analysis. Codes were developed inductively, allowing key themes related to learner engagement, usability, and perceived benefits to emerge. To ensure reliability, two independent coders reviewed the data, and Cohen's Kappa coefficient ($\kappa = 0.78$) confirmed substantial inter-coder agreement.

Quantitative data, derived from pre- and post-intervention questionnaires, were analyzed using the Mann–Whitney U test, a non-parametric method suitable for comparing two independent samples when normal distribution cannot be assumed. This test was used to assess whether there was a significant difference in learning outcomes and engagement metrics between the control and experimental groups. Descriptive statistics (medians and interquartile ranges) were also reported to support interpretation. In combining thematic insights with statistical evidence, this mixed-methods approach enabled a nuanced understanding of both the learner experience and the measurable impact of the proposed LLM-based framework.

3.5 Systematic Analysis of Features

To address the first specific objective a systematic literature review (SLR) was conducted. This approach enabled the researcher to synthesize current knowledge in a rigorous, structured, and reproducible manner, ensuring the framework developed in the second phase was grounded in evidence-based design principles. The review was guided by the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines, ensuring a comprehensive and transparent selection

process[124]. The primary aim was to identify and evaluate relevant studies focusing on key features that enhance the personalization of computer programming learning.

3.5.1 Data Sources and Search Strategy

The literature search was conducted across five reputable academic databases: IEEE Xplore, ACM Digital Library, Google Scholar, SpringerLink, and Scopus. These sources were selected due to their extensive collection of peer-reviewed articles and conference papers in the fields of artificial intelligence, educational technology, and computer science. The review targeted studies published between 2019 and 2024 to capture the most recent advancements in GenAI technologies and their applications in education.

A structured search strategy was employed using a combination of keywords and Boolean operators. The primary search terms included; “Generative AI” AND “personalized learning”, “AI frameworks” OR “adaptive learning systems” AND “computer programming”, and “Content accuracy” AND “coherence in educational AI”. Boolean operators such as AND, OR, and NOT were used to refine the search and exclude irrelevant results. This approach ensured a comprehensive coverage of relevant studies while maintaining focus on educational applications of GenAI. Additionally, the search was restricted to peer-reviewed articles published in English, ensuring the inclusion of high-quality and credible research sources and exclusion criteria. To ensure the relevance and quality of the studies included in the review, specific inclusion and exclusion criteria were established.

3.5.2 Inclusion Criteria

The selection of literature for the review was guided by clearly defined inclusion criteria to ensure the relevance and quality of the studies considered. Only peer-reviewed journal articles and conference papers published between 2019 and 2024 were included, ensuring that the reviewed content reflected recent advancements and ongoing trends in the field. The review focused on studies that explored the application of Generative AI and Large Language Models (LLMs) within educational settings, with particular emphasis on computer programming education. Preference was given to research that examined how AI technologies enhance personalized learning experiences. Key features of interest included adaptability, contextual relevance,

coherence of instructional content, and content accuracy. These criteria ensured that the selected literature aligned closely with the research objectives and provided a robust foundation for developing and validating the proposed instructional framework.

3.5.3 Exclusion Criteria

To maintain focus and methodological rigor, specific exclusion criteria were applied during the literature selection process. Studies that were not peer-reviewed, including opinion articles, white papers, and blog posts, were excluded to ensure the academic credibility of the sources. Additionally, research that did not address the use of AI in educational contexts or that primarily examined non-educational applications of Generative AI was excluded, as such studies did not align with the scope of this research. Finally, articles published in languages other than English were omitted to avoid potential misinterpretation and ensure consistency in the review process. These exclusion parameters helped refine the literature pool to the most relevant and reliable sources for informing the development of the LLM-based personalized programming framework.

3.5.4 Study Selection

The study selection process was conducted in multiple phases to ensure that only high-quality, relevant studies were included in the final synthesis. The process began with an initial search across five major academic databases using pre-defined search strings. Articles retrieved from these databases were subjected to a structured screening procedure involving a title and abstract review, followed by a full-text evaluation of eligible studies. At the title and abstract screening stage, articles were assessed against the predefined inclusion and exclusion criteria to eliminate non-peer-reviewed publications, studies unrelated to AI in education, and works focusing on non-instructional applications of Generative AI. Articles that met the initial criteria were then reviewed in full to determine their methodological soundness and relevance to the study's focus on LLM features supporting personalized programming education. A detailed account of the study selection results and key outcomes is presented in Chapter 4 of this thesis.

3.5.5 Data Extraction and Synthesis

Data extraction was conducted using a standardized data extraction form, designed to ensure consistency, accuracy, and replicability throughout the review process. This structured tool allowed for the systematic collection of information across all studies selected during the review. Each entry in the form captured bibliographic details (author, title, year), study objectives, research context, and a brief description of the intervention or framework under study. The standardized approach minimized data handling errors and supported a transparent audit trail for future reference.

In addition to basic bibliographic data, the extraction process specifically examined the methodological approaches employed in each study. These were classified into qualitative, quantitative, or mixed-methods designs based on explicit declarations in the articles or through methodological inference from data collection techniques (e.g., use of interviews, surveys, or experimental trials). This categorization was important for understanding the nature of evidence presented and for evaluating the robustness and generalizability of the reported findings.

Further, the extraction focused on capturing information about each framework's functional design and instructional capabilities. This included identifying whether the GenAI-powered tools implemented adaptive feedback loops, real-time learner modelling, context-aware response generation, or scaffolded instruction. Studies were reviewed in terms of how these mechanisms were integrated into programming education environments such as intelligent tutoring systems, learning platforms, or AI plug-ins used in classroom instruction. The design and operational descriptions were collected verbatim or paraphrased, then stored in the extraction form for thematic synthesis.

Following the extraction phase, a preliminary synthesis was conducted by grouping similar attributes and feature descriptions across studies. This thematic organization focused on how the tools addressed pedagogical principles such as content relevance, adaptability, coherence, and feedback quality. A formal synthesis and critique of these themes, including gaps and inconsistencies in current GenAI frameworks is presented in Chapter Four. All data used in the review were obtained from publicly available academic databases and peer-reviewed publications. No primary data collection

involving direct contact with human participants was conducted at this stage of the study. This means no surveys, interviews, or observational procedures were applied during the SLR phase. Ethical considerations were therefore limited to proper citation, avoidance of misrepresentation, and ensuring that secondary data were used responsibly in line with academic integrity standards. In adhering to a rigorous and transparent extraction and synthesis process, this phase of the study laid a sound evidence-based foundation for the design of the LLM-powered instructional framework, while maintaining methodological and ethical integrity throughout.

3.6 Developing the LLM-Based Learning Framework

This section outlines the development process of the LLM-based instructional framework, aligned with the second specific objective of the study. The framework was designed to support personalized programming education by integrating pedagogically informed features derived from the systematic review. The development process followed an iterative, modular approach informed by design-science methodology, focusing on framework creation through problem identification, solution design, and testing. The framework was developed using GPT-4o as the core LLM component, selected for its conversational fluency, multilingual support, and advanced reasoning capabilities[64]. Key development stages included requirement analysis, system design, component integration, and testing to ensure the framework was well-adapted to the instructional needs of computer programming learners.

3.6.1 Feature Mapping and Architecture Design

The development of the personalized programming instruction framework began by mapping the features and best practices identified in the systematic review to system-level functional components. This translation process ensured that each pedagogical requirement such as real-time feedback, adaptive learning, prompting, and learner progress tracking was addressed through specific, modular backend services. These features were organized into the main components of the framework. To implement these components, the backend was built using FastAPI, chosen for its speed, asynchronous capabilities, and ease of integration with modern Python-based AI toolchains. The modular backend design followed domain-driven principles, with clearly defined directories for routes, services, database models, and configuration

settings. This separation of concerns allowed for clean scalability, code maintainability, and rapid prototyping cycles.

The framework included a structured RESTful API layer, exposing endpoints that allowed student inputs, code attempts, and interactions with the AI model to be logged and processed in real time. Prompts from students were routed through this layer to the Prompt Processing Engine, which handled formatting, contextual augmentation, and interaction with the LLM (GPT-4o). The system's architecture incorporated an adaptive feedback loop, which used metadata from student sessions such as number of attempts, hint requests, and time-on-task to inform subsequent prompts dynamically.

All learner interaction data was persisted in a PostgreSQL database via SQLAlchemy ORM, enabling detailed learner modeling and longitudinal progress tracking. To ensure user data security, role-based access control and token-based authentication (OAuth2) were implemented at the API layer. This architecture was designed not only to support LLM-driven tutoring but also to serve as a scalable foundation for integration with frontend components and learning management systems. The choice of FastAPI and PostgreSQL, along with modular encapsulation of routing and service logic, aligned with the goal of building a robust, testable, and extensible educational platform tailored to programming instruction. Figure 3.2 below shows the system architectural data flow, which integrates a user interface, FastAPI backend, prompt and feedback engines, and external services (ChatGPT API, PostgreSQL, OAuth2, and GitHub) to deliver secure, adaptive, and personalized programming support.

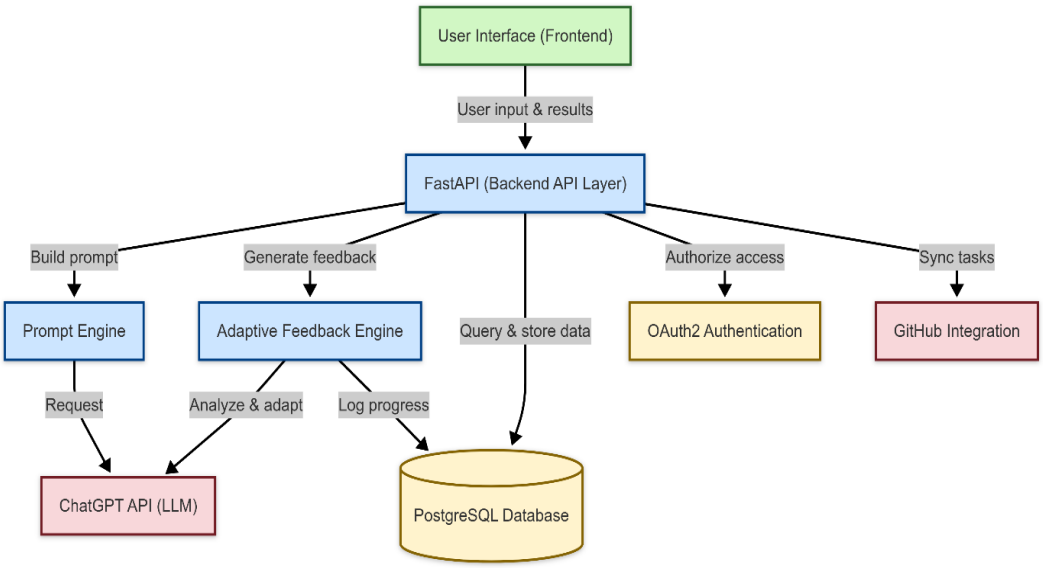


Figure 3.2: Architectural Data Flow of the Framework

3.6.2 Prompt Engineering and Dialogue Flow Design

Following the architectural design, the next phase of development focused on engineering effective prompts and structuring dynamic dialogue flows to support personalized, context-aware learning. Prompt engineering was prioritized because LLMs are highly sensitive to input phrasing, and poorly designed prompts often lead to hallucinations, irrelevant suggestions, or misaligned content. Therefore, this component was central to enabling coherent and pedagogically valid learner interactions. To ensure relevance and instructional clarity, a set of base prompt templates was created using insights from the literature review[89]. These templates were organized by instructional intent explanatory prompts, code debugging prompts, feedback prompts, and motivational scaffolds. Each template followed a structured format that specified the tone, instructional objective, and expected output type from the LLM. For instance, templates guiding learners through code tracing tasks included "Think step-by-step" tokens to improve LLM reasoning coherence.

Prompt complexity was not static. A dynamic adaptation mechanism was implemented in the backend using conditional logic that monitored student performance indicators such as repeated errors, hint requests, or extended inactivity. Based on these indicators, the system modified prompt phrasing in real-time, offering either more detailed explanations, simplified examples, or conceptual hints. This dynamic flow ensured that learners received tailored support matching their current

cognitive load and level of understanding. The dialogue orchestration followed a turn-based model, with the learner and the LLM engaging in a controlled back-and-forth. Each LLM response was logged alongside the student's previous code submission or query, allowing for traceability and data-driven reflection. The backend stored session context and prior exchanges, enabling continuity in multi-turn interactions.

Furthermore, a layer of content verification filters was added to the prompt output phase to flag potentially incorrect or low-confidence responses from the LLM. This safeguard served as a preliminary quality check before delivering responses to students. The overall prompt interaction logic was encapsulated within dedicated service files in the backend, ensuring modularity and testability. These services were exposed through REST endpoints, allowing the frontend interface to trigger prompt generation, record user responses, and update learner feedback dashboards in real time. This prompt engineering and dialogue design process laid the foundation for a personalized and pedagogically sound interaction flow that maximized the instructional potential of the LLM, while reducing cognitive overload and minimizing misleading outputs. Figure 3.3 illustrates a feedback-driven LLM interaction loop where user input is processed through a prompt engine and adaptation logic to generate refined prompts, with low-confidence responses triggering re-adaptation before final display to the user.

In addressing bias, prompt design strategies were engineered to avoid culturally insensitive phrasing, gender stereotypes, or regionally irrelevant examples. The prompt templates were stress-tested using multiple simulated scenarios to observe any biased or misleading completions. Additionally, a verification layer was employed to flag potentially biased or hallucinated outputs by scoring the LLM's confidence and factual consistency. Low-confidence completions were automatically re-prompted or filtered, ensuring that only pedagogically aligned and equitable responses were delivered to students. These bias mitigation steps improved trustworthiness and instructional neutrality in the system's outputs.

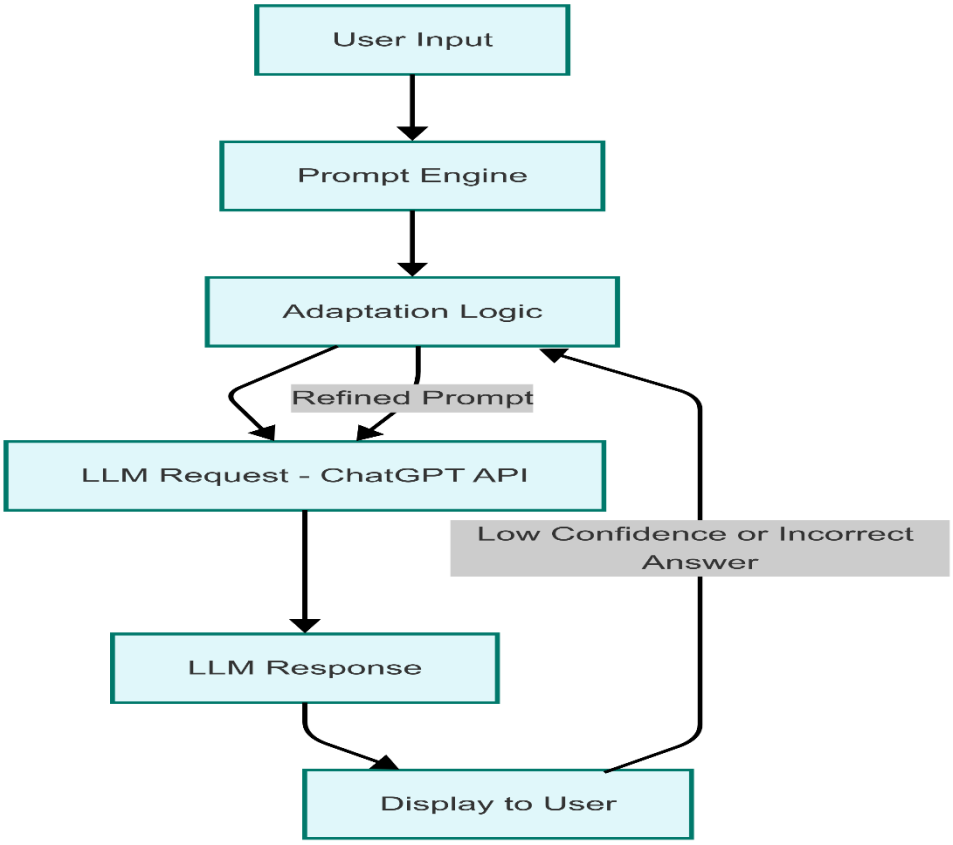


Figure 3.3: Prompt Adaptation Workflow

3.6.3 Collaborative Coding and Real-Time Interaction

To support peer-assisted learning and mirror real-world software development practices, the framework incorporated a collaborative coding environment. This module was designed to allow multiple learners to co-edit programming tasks, exchange feedback, and engage in structured group problem-solving within a shared digital space. The integration of collaborative features was guided by pedagogical principles emphasizing social learning, critical thinking, and engagement through active participation. Technically, the collaborative environment was implemented using Monaco Editor, the open-source foundation behind Visual Studio Code, embedded within the web interface. This editor supported syntax highlighting, real-time linting, and autocompletion for C programming, the course's primary language. To enable live collaboration, the backend leveraged WebSocket communication channels to broadcast code changes across participants' sessions in real time. This approach ensured low-latency interaction, enabling students to view updates as their peers typed, much like a Google Docs experience for code.

Each collaborative session was associated with a unique room ID and managed by the session orchestration service. Users could either join existing rooms via secure access tokens or initiate new collaborative exercises. GitHub API integration was included to simulate real-world version control workflows, allowing students to fork code examples, push changes, and conduct basic pull request-style reviews within a classroom context. This functionality was essential in promoting team-based coding practices and version awareness among novice programmers. The communication between learners and instructors was facilitated through a chat and notification module, built using Socket.IO for real-time messaging. This allowed students to ask for help, provide feedback to peers, or receive instructor tips during coding sessions. A threaded message structure was implemented to help group conversations by topic or task, preventing fragmentation in peer discussions.

All collaborative interactions were logged in the backend database, enabling instructors to review participation patterns and identify support needs. This data was also useful in evaluating engagement metrics during the framework's classroom deployment. The inclusion of real-time collaborative coding was essential not just for technical exposure but also for fostering a supportive learning community. Providing students with the ability to view each other's problem-solving approaches, suggest improvements, and ask questions in context, the framework aimed to simulate an active lab environment that encouraged knowledge sharing and continuous peer-driven feedback.

3.6.4 Testing and Deployment

Following the successful implementation of core framework components, a structured testing and deployment process was undertaken to ensure system reliability, usability, and readiness for classroom integration. This phase aligned with the evaluation stage of the design science paradigm and served to verify that the developed system behaved as intended under real-world usage conditions. Unit testing was conducted across all backend modules using the Pytest framework. This ensured that core functions, such as LLM integration for prompt handling, response generation, and data persistence operated correctly and independently. Test coverage included edge cases such as malformed input, session timeouts, and database connection failures. Additionally,

mock LLM responses were used during testing to isolate system logic from third-party API fluctuations, allowing for controlled verification of internal components.

End-to-end (E2E) testing was implemented using Cypress, focusing on user workflows within the frontend. Scenarios tested included logging in, submitting a programming query, receiving a hint, joining a collaborative session, and receiving instructor feedback. These tests simulated learner interactions to confirm that all integrated services worked seamlessly across modules. Any anomalies identified during E2E tests were logged, triaged, and resolved during iterative refinement. The application was containerized using Docker, facilitating consistent deployments across development, staging, and production environments. Docker Compose scripts were used to orchestrate services such as the API server, database, and WebSocket engine. Deployment was carried out on Truehost Cloud, a reliable local cloud hosting provider. The backend was hosted on a Truehost Linux VPS instance, chosen for its affordability, high uptime, and support for custom configurations. A reverse proxy was configured using NGINX to manage routing, enforce HTTPS, and optimize load handling. User data was stored in a PostgreSQL instance managed within the same server cluster, with scheduled backups enabled.

For continuous improvement, a lightweight CI/CD pipeline was set up using GitHub Actions, enabling automated testing, linting, and deployment upon updates to the repository. This ensured fast feedback during development and reduced the risk of deploying unstable changes. The final deployed version of the system was then made accessible to students and instructors through a secure web portal for real-world usage, enabling logging of all interactions and feedback as part of the subsequent evaluation phase.

3.7 Framework Performance Evaluation

The final phase of the research focused on evaluating the effectiveness of the developed LLM-based personalized programming framework in a real-world classroom setting. This phase was guided by the third specific research objective, which aimed to assess the framework's impact on student engagement, learning outcomes, and the quality of instructional content. To achieve this, a quasi-experimental design was implemented at Murang'a University of Technology during

the September–December 2024 semester. The evaluation adopted a parallel instructional model in which students enrolled in an "Introduction to Computer Programming" unit of study were assigned to either a control group or an experimental group. The control group received instruction using traditional programming resources and materials, while the experimental group interacted with the developed LLM-based framework as a supplementary instructional tool. This arrangement was selected to isolate the framework's contribution by comparing learning outcomes across the two groups under similar instructional conditions.

Data collection was carried out using structured pre- and post-tests consisting of 10 standardized questions, each worth 5 marks, to objectively measure students' understanding of core C programming concepts before and after engagement with the framework (see Appendix 4). The tests covered a range of topics including program writing, control structures, functions, arrays, pointers, debugging, and conceptual explanations. The same set of questions was administered to both the experimental and control groups under identical conditions, allowing for a consistent comparison of knowledge gain attributable to the framework. This allowed for an objective assessment of knowledge gain attributable to the framework.

Given that the sample size was relatively small and the data did not meet the assumptions of normality, the Mann-Whitney U test, a non-parametric alternative to the independent samples t-test, was used for statistical analysis. This method enabled the researcher to determine whether there was a statistically significant difference in learning outcomes between the two groups without relying on distributional assumptions. The use of this test ensured the robustness and appropriateness of the analysis method for the given dataset. Additionally, qualitative responses from students were coded and analyzed thematically to uncover emerging insights into learner engagement, perceived usefulness, and content quality. This mixed-method approach allowed for triangulation of findings, enhancing the validity of the evaluation. This comprehensive evaluation ensured that the framework was rigorously tested under authentic learning conditions, providing both empirical and experiential evidence to support its instructional effectiveness. The detailed analysis of results is presented in Chapter Four.

3.8 Experimental Materials

This section outlines the tools, technologies, and computing resources used to design, develop, and test the LLM-based personalized programming learning framework. The selection of experimental materials was guided by the need for scalability, interoperability with modern AI services, and pedagogical suitability in a programming education context. The development environment consisted of a cloud-hosted backend, a modern front-end framework, and integration with OpenAI's GPT API. The backend logic was implemented using Python and FastAPI, while the user interface was built with React.js for responsiveness and interactivity. PostgreSQL was selected for structured data persistence, and GitHub served both version control and code collaboration purposes. For testing and deployment, CI/CD pipelines and virtual private servers (VPS) were used to simulate live environments and maintain continuous updates. Table 3.2 provides a summary of the main experimental materials employed in this study.

Table 3.2: Experimental Materials

Experimental Material	Description
Development Machine	Windows 10 (64-bit), 16GB RAM, Intel® Core i7 CPU
Backend Stack	Python 3.11, FastAPI, SQLAlchemy ORM
Frontend Stack	React.js (with JSX), TailwindCSS
LLM Integration	OpenAI ChatGPT API (GPT-4o model)
Database	PostgreSQL 14
Code Collaboration	GitHub + Git CLI
Authentication	OAuth2 Protocol (Implemented via FastAPI dependencies)
Deployment Environment	VPS via Truehost, NGINX server, HTTPS enabled
CI/CD Pipeline	GitHub Actions (auto deploy on push to main)
Visualization Tools	Python, Chart.js for analytics
Testing Tools	Jest (frontend), PyTest (backend), Cypress (E2E testing)

The use of cloud deployment ensured real-time accessibility for students during evaluation, while secure authentication and structured data storage-maintained system integrity and learner data privacy. Integration of modern development tools and

automated testing frameworks facilitated efficient prototyping and iterative improvement.

3.9 Controlled User Studies

To evaluate the effectiveness of the LLM-based framework, controlled user studies were conducted with undergraduate students enrolled in an Introduction to Programming course at Murang'a University of Technology. Participants were randomly assigned to either the experimental group (which used the LLM-based framework) or the control group (which continued with traditional teaching methods).

Prior to the experiment, both groups underwent pre-tests to establish their baseline knowledge and skills in programming. Throughout the study, the experimental group interacted with the AI-powered framework, receiving personalized recommendations, feedback, and interactive tutoring based on their performance. The control group followed conventional instruction methods, without AI assistance. After the intervention period, both groups completed post-tests to measure their learning progress and comprehension levels. Additionally, structured questionnaires and interviews were conducted to collect qualitative feedback from participants regarding their learning experiences. The structured questionnaire assessed variables such as usability, effectiveness, and engagement, while the interviews explored student perceptions, challenges, and areas for improvement.

Quantitative data from the pre- and post-tests were analyzed using the Mann–Whitney U test, a non-parametric statistical method suitable for comparing two independent groups when the assumption of normality is not guaranteed. This test facilitated comparison of learning gains between the two groups to determine whether observed differences were statistically significant. Qualitative data were analyzed thematically to extract recurring patterns and insights about the usability and pedagogical value of the framework. This rigorous mixed-methods evaluation ensured that the study findings were empirically validated, contextually rich, and reproducible, providing a solid foundation for assessing the impact of the LLM-powered learning framework on programming education.

3.10 Ethical Considerations

Ethical standards were upheld throughout all phases of the research to ensure the rights, privacy, and confidentiality of participants. Given the application of Large Language Models (LLMs) in education, special attention was given to ethical concerns such as bias, fairness, and data privacy. The development process of the framework included rigorous testing and validation to identify and mitigate potential biases in LLM-generated outputs, ensuring that all students received equal and fair treatment in the learning process.

To safeguard the rights of participants, the research received approval from the Board of Postgraduate Studies (BPS) at Murang'a University of Technology and a research permit from the National Council of Science and Technology (NACOSTI), as required by government regulations. Participants were provided with an informed consent form outlining the study's purpose, procedures, potential risks and benefits, and their rights as participants. The form clearly stated that participation was voluntary and that students could withdraw at any time without penalty. Additionally, a verbal explanation of the study was provided to ensure that participants fully understood the research before giving their consent. Depending on the mode of data collection, consent was documented through signed forms or recorded verbal agreements. Confidentiality and data protection protocols were strictly observed. All collected data were anonymized and stored securely, ensuring that personal identifiers were removed before analysis. The data were used solely for research purposes, with restricted access granted only to authorized research personnel.

Academic integrity was maintained by ensuring proper citation of all secondary sources, respecting intellectual property rights, and adhering to recognized citation formats. Additional measures were implemented to address potential ethical concerns related to AI-generated content. This included regular audits of LLM outputs for biases, the integration of feedback mechanisms to enhance fairness and accuracy, and strict compliance with data protection laws and institutional policies. Participants were informed about these measures and their significance in ensuring ethical compliance. Through these steps, the research maintained high ethical standards, ensuring fairness, transparency, and accountability in the study of AI-driven personalized programming education.

3.11 Summary

This research employed a design science research paradigm supported by a mixed-methods design to guide the development and evaluation of a GenAI-based personalized programming framework. The methodology unfolded in three phases: problem identification through literature and expert consultation, solution development via iterative prototyping, and classroom evaluation. The target population comprised undergraduate students enrolled in programming courses across five computing degree programs. Based on a conservative estimate of 45 learners per program, consistent with academic planning norms, the minimum population size was set at 225 students. Sampling involved purposive selection for both expert validation and framework evaluation stages. Three distinct instruments were used: expert surveys (students and professionals) to validate framework features, and a structured questionnaire for evaluating the framework's usability, interactivity, and pedagogical effectiveness. The reliability and validity of the instruments were confirmed through pilot testing and expert review. This chapter establishes the methodological rigor and empirical foundation for assessing the framework.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This chapter presents the results and findings of the study in line with the three specific research objectives. The chapter is structured to demonstrate how the proposed Large Language Model (LLM)-based framework addressed the problem of limited personalization in traditional computer programming education. The chapter begins with an analysis of existing GenAI-based instructional tools, identifying the core features that support personalized learning. This is followed by the design, development, and implementation of the proposed LLM-integrated framework. Finally, the chapter presents the results of the framework's evaluation, drawing from both quantitative and qualitative data to assess its effectiveness in improving learner engagement, instructional quality, and learning outcomes.

4.2 Feature Analysis

This section presents the results addressing the first objective of the study: to analyze existing LLM-based frameworks and tools to identify features and best practices that support effective personalized learning in programming education. To accomplish this, a Systematic Literature Review (SLR) was conducted using a structured protocol detailed in the methodology chapter. The review synthesized existing empirical and theoretical contributions on the integration of Generative AI (GenAI) into programming education, with a specific focus on identifying capabilities that enhance personalization.

The SLR followed a rigorous multi-stage process involving database searches, title and abstract screening, full-text review, and quality assessment. A total of 120 articles were initially retrieved from reputable academic databases including IEEE Xplore, ACM Digital Library, SpringerLink, Scopus, and Google Scholar. After applying the inclusion and exclusion criteria, 42 articles were selected for in-depth analysis. Through thematic analysis of the selected articles, eight key features were identified as critical in enabling personalized learning through GenAI tools. These are: i) Adaptability ii) Contextual Relevance iii) Clarity and Coherence iv) Accuracy and Reliability v) Prompt Engineering Effectiveness vi) Practical Usability vii)

Interactivity and Collaboration viii) Creativity and Uniqueness. These features form the foundation for the development of the proposed personalized learning framework discussed in later sections. Their relevance to educational practice was subsequently assessed through expert opinion surveys, presented in a separate section.

4.2.1 Overview of Literature Search and Study Selection

To systematically gather relevant studies for this review, a multi-stage literature search process was conducted, as outlined in the methodology section. The goal was to identify high-quality, peer-reviewed research focusing on the application of Generative AI (GenAI) in personalized programming education. The search was carried out using PRISMA guidelines and the initial database query yielded 120 articles. The first screening stage involved reviewing the titles and abstracts to determine their relevance to the study objectives. At this stage, 63 articles were excluded for reasons including duplication, non-peer-reviewed status, unrelated domains (e.g., AI in business or healthcare), and a lack of focus on educational outcomes. This left 57 articles for full-text assessment. A detailed review of each article was then conducted to evaluate their alignment with the key personalization-enhancing features under investigation in this study. During this phase, an additional 15 articles were excluded because they either lacked empirical backing, did not focus on GenAI in personalized learning contexts, or failed to provide actionable findings relevant to programming education. Figure 4.1 presents a summary of the article selection stages using a PRISMA-style flow diagram.

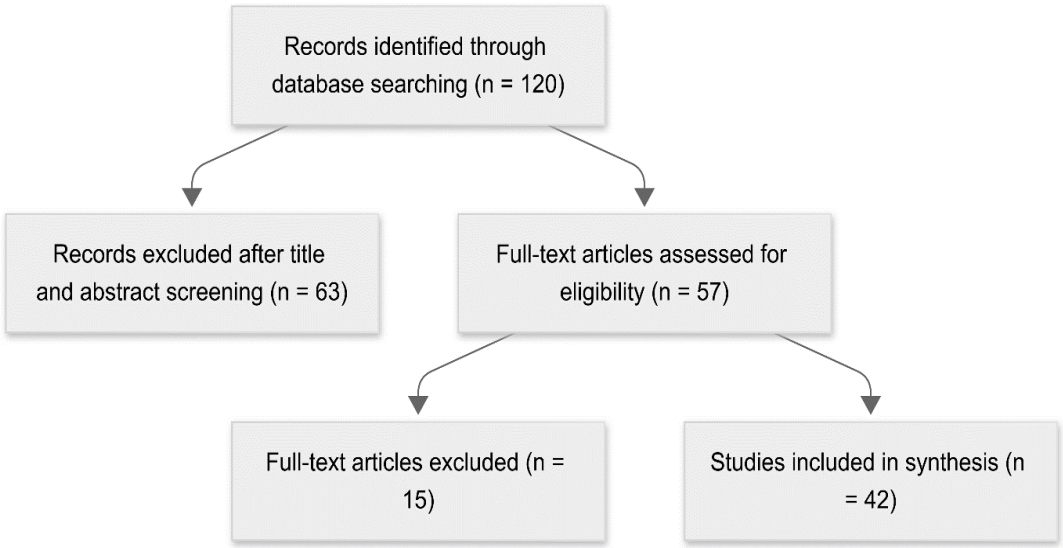


Figure 4.1: PRISMA Diagram of Literature Selection Process

Ultimately, 42 articles were selected for inclusion in the final synthesis. These studies span the years 2020 to 2024 and provide comprehensive insights into the state-of-the-art application of Generative AI (GenAI) in programming education. The selected literature includes diverse research methodologies such as systematic reviews, empirical case studies, experimental evaluations, and framework development. This methodological diversity offered both breadth and depth, enabling a nuanced understanding of the pedagogical, technical, and design principles guiding the use of GenAI in personalized programming instruction.

Each entry in the 42 articles included in the final synthesis summarizes the study's citation, year of publication, methodology, key findings, and the core personalization-enhancing features it supports. These studies collectively form the evidence base for the thematic analysis in Section 4.2.2, which examine how these features contribute to GenAI-driven personalized programming education. A complete table with the entire 42 articles is available in appendix 3.

4.2.2 Thematic Synthesis of Key Features

Based on the final 42 studies synthesized in this review, eight core features emerged as critical to the successful implementation of personalized programming education using Generative AI (GenAI) technologies. These features represent recurring themes across the literature and capture both the pedagogical and technical affordances of GenAI in tailoring instruction to individual learner needs. The thematic synthesis was guided by a qualitative coding approach, in which each study was analyzed for explicit or implicit support of one or more personalization-enhancing features. The resulting themes are not mutually exclusive; most studies contributed to more than one feature, reflecting the interdependence of factors like adaptability, accuracy, and contextual relevance in GenAI-driven educational systems. Table 4.1 provides a frequency distribution of the eight synthesized features and best practices across the 42 selected studies.

Table 4.1: Frequency of Personalization Features Identified

Features and Best Practices	Number of Supporting Studies	Percentage of Total (N = 42)
Practical Usability	17	**40%**
Adaptability	16	**38%**
Accuracy and Reliability	16	**38%**
Prompt Engineering Effectiveness	14	**33%**
Interactivity and Collaboration	13	**31%**
Contextual Relevance	8	**19%**
Clarity and Coherence	7	**17%**
Creativity and Uniqueness	7	**17%**

The frequencies represent the number of studies that directly addressed or implemented each feature as part of their methodology or design. While some features such as practical usability and adaptability appeared in a larger proportion of studies, others like creativity and uniqueness were less common but nonetheless significant in advancing personalized learning.

4.2.3 Expert Validation

Following the thematic synthesis from the SLR, an expert validation exercise was conducted to assess the relevance, clarity, and practical applicability of the eight features identified for enhancing personalized programming education through Generative AI (GenAI). This validation step served a dual purpose: (i) to ensure that the synthesized features reflect real-world instructional needs and expectations, and (ii) to provide empirical support for their inclusion in the development of the proposed framework.

4.2.3.1 Participants and Instruments

To validate the relevance and practical applicability of the eight features derived from the SLR, expert opinion was gathered using structured questionnaires. Two distinct groups were purposively selected to represent both the pedagogical and technical dimensions of programming education. The first group consisted of 12 expert students, including current third- and fourth-year undergraduate students and recent graduates

from programs such as BSc. Information Technology, Computer Science, and Business Information Technology. These participants were selected based on their demonstrated proficiency in C programming, all had attained either a grade A or B, and their advanced level within their academic programs, positioning them to critically assess the learning experience. The second group comprised eight professional experts, including six university lecturers with experience teaching programming and two ICT industry professionals engaged in educational technology and AI-related work.

Separate but thematically aligned questionnaires were developed for the two groups. Each instrument was informed by the eight personalization-enhancing features previously identified and was structured into two main sections: (i) closed-ended items using a 5-point Likert scale to rate the importance, clarity, and applicability of each feature; and (ii) open-ended items designed to elicit qualitative feedback on the strengths, limitations, and improvement areas of GenAI in programming instruction. The combination of learner and practitioner perspectives, along with both quantitative and qualitative data, provided a robust basis for evaluating the conceptual and functional validity of the proposed features.

4.2.3.2 Expert Reviews

This section presents the results from the expert validation survey, designed to assess the relevance, clarity, and practical applicability of the eight personalization-enhancing features identified in the SLR. A total of 20 participants, 12 expert students and 8 professional experts contributed both quantitative and qualitative data to the validation process. To ensure the validity and reliability of the questionnaire instrument, a pilot test was conducted prior to full deployment. The internal consistency of the survey items was assessed using Cronbach's alpha, yielding a coefficient of 0.84, which exceeds the commonly accepted threshold of 0.70. This indicates that the items used to evaluate each feature were highly reliable.

Table 4.2 summarizes the demographic information of the student respondents while Table 4.3 summarizes the demographics of the professional respondents.

Table 4.2: Demographic Summary of Student Respondents (n = 12)

Attribute	Category	Frequency	Percentage (%)
Gender	Male	8	66.7
	Female	4	33.3
Study Program	BSc. Information Technology	5	41.7
	BSc. Business Information Technology	2	16.7
	BSc. Computer Science	4	33.3
	BSc. Cyber Security	1	8.3
Year of Study	Year 3	3	25.0
	Year 4	3	25.0
	Recent Graduate	6	50.0
C Programming Grade	Grade A	7	58.3
	Grade B	5	41.7

The student group consisted of third- and fourth-year undergraduates as well as recent graduates, with 50% having recently completed their studies. All student participants had attained either an A or B in C programming, indicating strong technical competence. Most were enrolled in Information Technology and Computer Science programs, and 66.7% were male. This diverse group provided valuable insights into the practical use of GenAI tools from the perspective of advanced learners.

Table 4.3: Demographic Summary of Professional Expert Respondents (n = 8)

Attribute	Category	Frequency	Percentage (%)
Role/Sector	Teaching	6	75.0
	Industry	2	25.0
Academic Qualification	First Degree	2	25.0
	Master's Degree	6	75.0
GenAI Experience	Basic	2	25.0
	Intermediate	2	25.0
	Advanced	2	25.0
	Expert	2	25.0

The professional experts included six lecturers and two industry practitioners, with 75% of participants drawn from the teaching sector. The majority (75%) held a Master's degree, while the remaining 25% held a first degree. Their experience with Generative AI was evenly distributed, with equal representation across basic, intermediate, advanced, and expert levels. This group contributed practitioner-oriented insights into the pedagogical applicability, accuracy, and classroom integration of GenAI systems in programming instruction.

To enable quantitative validation of the eight key features identified in the systematic literature review, five evaluation parameters were derived and used in the expert opinion survey. These parameters, feedback on code syntax and explanations, prompt response relevance, logical organization of GenAI responses, relevance of content to real-world applications, and accuracy of content were carefully selected to reflect the most observable and measurable aspects of the eight synthesized features. Each evaluation parameter integrates two or more related features from the SLR. For example, the parameter "feedback on code syntax and explanations" encapsulates aspects of clarity and coherence, accuracy and reliability, and practical usability, while "prompt response relevance" draws from contextual relevance and prompt engineering effectiveness. Similarly, "logical organization" is rooted in clarity, coherence, and prompt design, whereas "relevance to real-world applications" aligns with contextual relevance, usability, and creativity. The parameter "accuracy of content" directly reflects the precision and reliability of AI-generated outputs, as

emphasized in the SLR. In condensing the eight qualitative features into five practical evaluation items, the survey maintained conceptual integrity while ensuring clarity, focus, and ease of response for participants. This methodological alignment ensured that the quantitative validation remained firmly grounded in the thematic findings of the literature review. Participants rated key GenAI capabilities in programming education on a 5-point Likert scale. The results are presented in Table 4.4 and visualized in Figure 4.2.

Table 4.4: Mean Ratings and Standard Deviations of GenAI Capabilities

Evaluation Parameter	Mean Score	Standard Deviation
Feedback on code syntax and explanations	4.50	0.51
Prompt response relevance	3.65	0.49
Logical organization of GenAI responses	3.95	0.51
Relevance of content to real-world applications	3.95	0.51
Accuracy of content	3.70	0.47

Each of the 20 participants rated the extent to which GenAI tools demonstrated each capability using a five-point Likert scale. The scale ranged from 1 ("Not at all") to 5 ("To a great extent"). Responses were recorded for each parameter, and the mean score was calculated by summing all participant responses for a given parameter and dividing the total by the number of participants (n = 20). The standard deviation for each parameter was also computed to determine the extent of variation or agreement among participants' responses. A lower standard deviation indicates greater consensus.

The highest-rated parameter was "Feedback on code syntax and explanations," which received a mean score of 4.50 and a standard deviation of 0.51. This indicates strong consensus that GenAI tools are particularly effective in providing immediate, corrective guidance on code-level issues, a function highly valued by both learners and educators. Both "Logical organization of GenAI responses" and "Relevance of content to real-world applications" received a mean score of 3.95 and a standard deviation of 0.51. These scores suggest that the AI outputs are generally well-

structured and applicable to practical contexts, although there remains some moderate variability in user experiences. “Accuracy of content” was rated with a mean of 3.70 and a standard deviation of 0.47, showing that participants found the outputs to be mostly accurate but occasionally flawed, especially in more complex coding situations. The lowest-rated parameter was “Prompt response relevance,” which received a mean of 3.65 and a standard deviation of 0.49, highlighting a commonly shared concern that the AI responses sometimes failed to fully align with user intent or learning objectives.

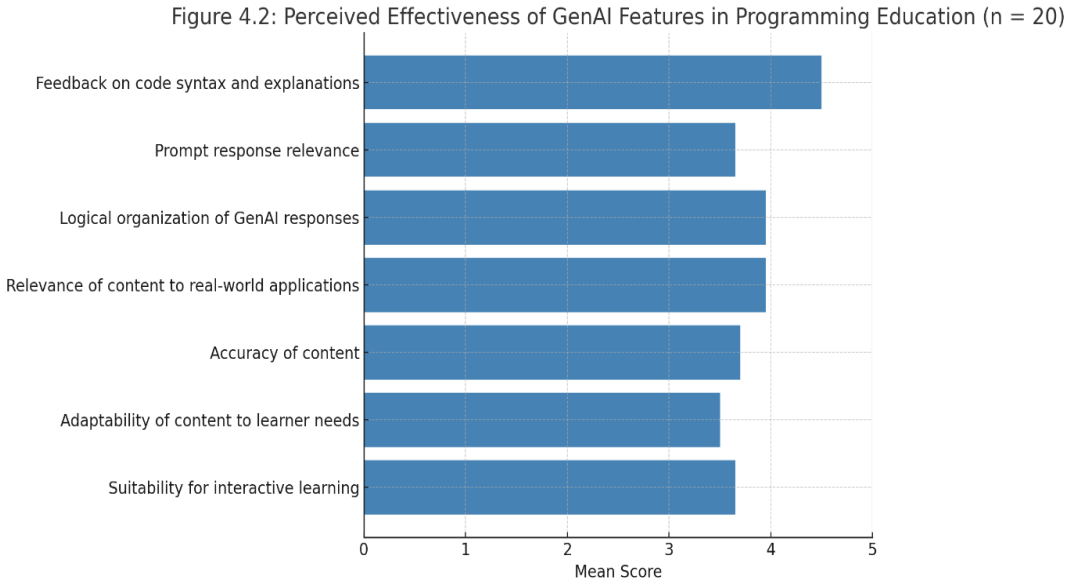


Figure 4.2: Perceived Effectiveness of GenAI Features

These findings illustrated in figure 4.2 visualizes the mean scores across the five parameters. The chart clearly shows a strong perceived strength in feedback capabilities, followed by moderate satisfaction with structure and applicability, and comparatively lower confidence in accuracy and prompt interpretation.

The qualitative responses provided rich, context-specific feedback on the instructional potential of GenAI in programming education. A recurring strength highlighted by both students and experts was GenAI’s ability to deliver clear, step-by-step explanations, especially in relation to syntax and debugging tasks. This supports the high quantitative rating for feedback and organizational clarity. Participants also acknowledged GenAI’s relevance to practical programming contexts, though some

raised concerns about occasional inaccuracies in more complex scenarios, echoing the moderate scores for content accuracy.

One of the most critical insights related to prompt engineering. Several respondents noted that effective use of GenAI depended heavily on the user's ability to craft well-structured prompts. As one participant observed, learners may fail to access the full benefits of GenAI without prior knowledge of prompt design, potentially limiting its impact in educational settings. This concern underscores the importance of designing GenAI tools that are both responsive to diverse input styles and embedded with scaffolding to support novice users. Suggestions for improvement included enhanced adaptability to learner input, better alignment with course objectives, and integration into collaborative learning environments.

The analysis indicates that GenAI tools are perceived as highly effective in providing code-level feedback and logical structure in programming explanations. However, areas such as prompt alignment and content adaptability warrant targeted enhancements, especially if GenAI systems are to be integrated into formal learning environments. The standard deviations across all parameters were relatively low (ranging from 0.47 to 0.51), suggesting a strong level of agreement among respondents regarding the evaluated features. These findings support the continued refinement of GenAI frameworks, particularly in improving responsiveness to prompt context and ensuring content adaptability to individual learner needs. The expert validation results, therefore, reinforce the feature set established through the SLR and guide the next phase of framework development.

4.3 Framework Development

This section presents the development of the proposed framework designed to enhance personalized learning in computer programming through the integration of Generative AI (GenAI). Guided by the eight key features identified through the systematic literature review and validated by expert opinion, the framework was operationalized into a set of five core components: (1) the LLM Integration Module, (2) the Adaptive Learning Tool, (3) the Collaborative Coding Platform, (4) the Learning Management System (LMS) Interface, and (5) the Database and Analytics Layer. These components work in tandem to support real-time feedback, dynamic content generation, intelligent

prompt handling, learner adaptability, and programming practice in a practical and accessible environment.

The design and development of the framework prioritized functionality, accessibility, and alignment with pedagogical goals. While the system integrates state-of-the-art language model capabilities with existing tools such as an online coding platform, it also offers a unified user interface to facilitate the delivery of content, prompts, feedback, and learner progress tracking. The following sections describe the principles that informed the system's architecture and how each component contributes to the realization of personalized learning features in programming education.

4.3.1 Design Principles and Component-Feature Alignment

The development of the framework was informed by a clear set of instructional and technical principles that prioritize personalization, scalability, and practical usability in programming education. Grounded in the findings of the systematic literature review and expert validation, the framework operationalizes eight key pedagogical features through five interconnected system components. These components were designed not only to work independently but also to reinforce each other in delivering a coherent and responsive learning experience.

At the core of the framework is the principle of feature-driven design. Each component was mapped directly to specific personalization-enhancing features to ensure that the final system was both theoretically aligned and practically functional. Rather than treating features and architecture separately, the development process embedded the features within the logic and responsibilities of the five primary modules. This approach ensured that abstract qualities such as adaptability, contextual relevance, and coherence were translated into tangible system behaviors. Another guiding principle was modular scalability. Structuring the framework into self-contained components, each responsible for a critical function such as content delivery, code execution, or prompt generation, the system supports future extension and refinement without requiring a complete redesign. This allows for seamless integration of new functionalities, including expanded prompt repositories, learner analytics, or additional collaboration tools.

The third principle was learner accessibility. The framework was built as a browser-based solution, requiring no local installation and compatible with standard computing environments. The interface was kept simple and intuitive to support self-paced learning while offering real-time feedback and guided practice opportunities. The inclusion of an online coding platform further enhanced practical usability by enabling learners to write, run, and test code without leaving the system environment. The alignment between system components and pedagogical features is summarized in Table 4.5, which illustrates how each module contributes to the realization of the eight validated personalization features.

Table 4.5: Framework Components and Their Supported Features

System Component	Features Embedded
LLM Integration Module	Contextual Relevance, Accuracy, Clarity and Coherence, Creativity, Prompt Engineering
Adaptive Learning Tool	Adaptability, Feedback Personalization, Progress Monitoring
Collaborative Coding Platform	Interactivity, Peer Engagement, Practical Usability, Creative Exploration
LMS Interface	Practical Usability, Prompt Input, Learning Navigation, Content Organization
Database and Analytics Layer	Accuracy, Adaptability, Data-Driven Feedback

This alignment strategy ensures that the framework is not only conceptually grounded in current pedagogical best practices, but also directly implementable through functional components. The next section provides a detailed view of the framework's system architecture and the interaction among its modules.

4.3.2 Architecture of the Developed Framework

The developed framework is structured as a modular, web-based system comprising five primary components: the LLM Integration Module, Adaptive Learning Tool, Collaborative Coding Platform, LMS Interface, and Database and Analytics Layer.

These components are designed to work in tandem to support personalized programming instruction by integrating real-time feedback, dynamic content generation, adaptive task allocation, and learner tracking into a unified educational platform. The system is implemented as a cloud-based application accessible through standard web browsers. The architecture leverages modern web technologies to ensure scalability, usability, and responsiveness. Learners interact with the system through a simplified dashboard-like interface that presents tasks, content, prompts, and feedback mechanisms, while instructors can configure content, track learner progress, and initiate discussions or support via integrated communication tools.

4.3.2.1 System Workflow

The instructional workflow begins with the instructor uploading or generating programming exercises aligned with predefined learning objectives. These exercises, along with contextual prompts and sample code, are made available to students through the LMS interface. Students engage with these tasks by interacting with the LLM to receive AI-generated explanations, hints, and error diagnostics. Upon submission of code or queries, the adaptive learning tool monitors learner inputs, solution quality, and task completion patterns. Based on this data, the system automatically adjusts task difficulty, suggests next learning steps, or provides remedial support. Students complete their programming tasks in the collaborative coding platform, which allows code execution and practice within an interactive environment. All activities are logged and analyzed via the database and analytics layer, providing instructors with performance dashboards and feedback loops.

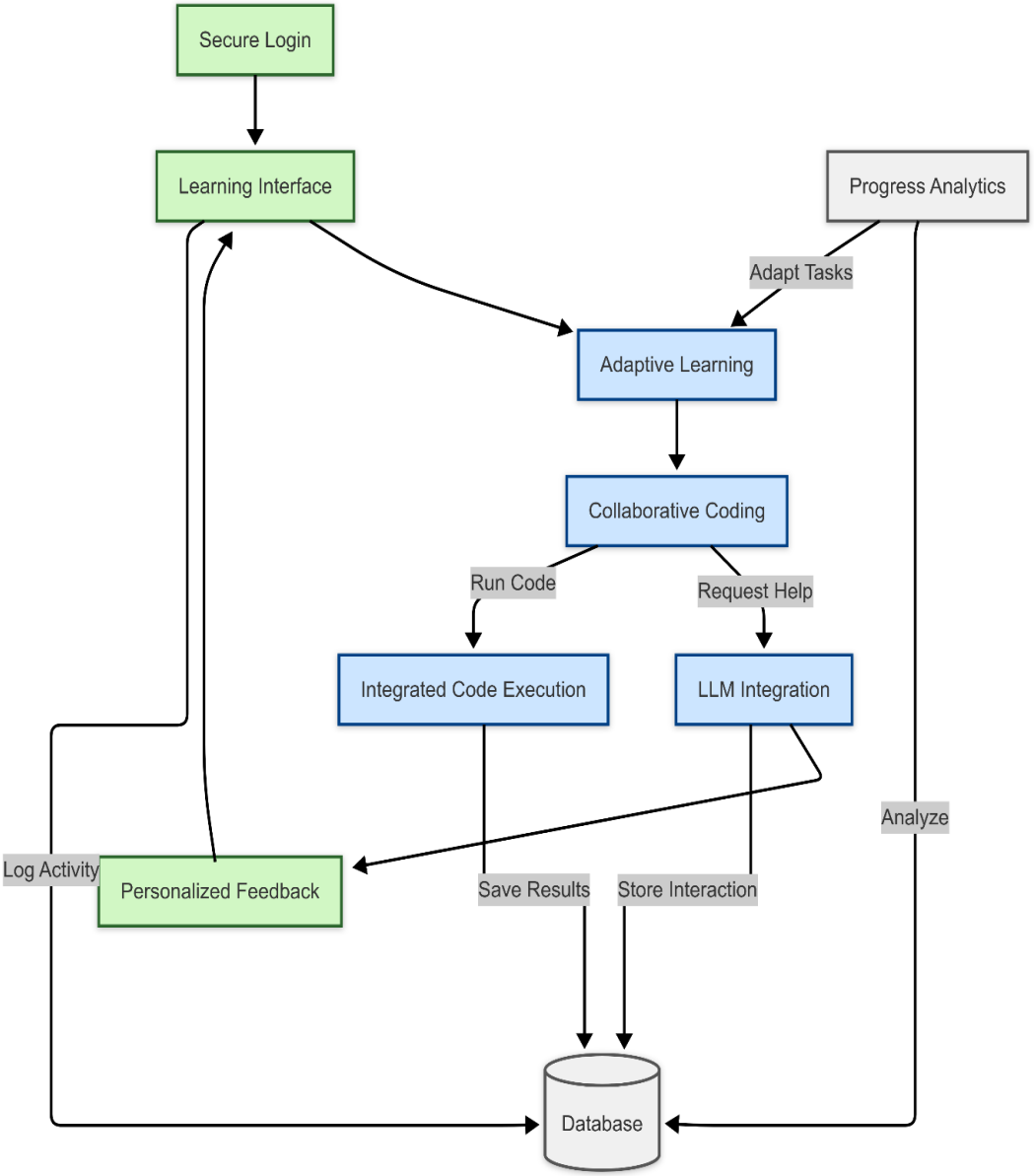


Figure 4.3: Architecture and Data Flow in the Developed Framework

The overall architecture of the developed GenAI-based personalized programming framework is illustrated in figure 4.3. It is composed of five functionally distinct yet highly interconnected components, each designed to fulfill a specific instructional or operational role. These include the LLM Integration Module, which enables intelligent content generation and personalized feedback; the Adaptive Learning Tool, which adjusts task complexity in real time based on learner performance; the Collaborative Coding Platform, which facilitates hands-on programming in a guided environment; the LMS Interface, which acts as the learner's central access point and communication hub; and the Database and Analytics Layer, which stores user interactions and informs

adaptive decisions. These modules are supported by secure authentication and integrated with external services such as GPT integration and an integrated online compiler, enabling a seamless learning experience that is responsive, data-driven, and context-aware. Each component was developed in alignment with the pedagogical principles identified through literature and expert validation, ensuring the system delivers personalized and scalable programming instruction.

4.3.2.2 Lifecycle of Programming Exercises

The learning experience within the framework follows a structured, feedback-rich loop. The cycle begins with instructors or the system generating exercises using the LLM. Students then attempt these exercises, receiving instant feedback and AI-generated hints. Based on this feedback, learners revise their code before submitting the final solution. Throughout this loop, the adaptive engine monitors learner behaviour, and instructors have the option to intervene via the communication module if needed. The interactive nature of this feedback loop ensure that students continuously refine their coding skills, leading to deeper comprehension and improved problem-solving abilities[85]. Figure 4.4 below shows the lifecycle of programming exercises. The dashed arrows show the use of Generative AI for creating programming exercises and providing feedback on students' programming exercise attempts.

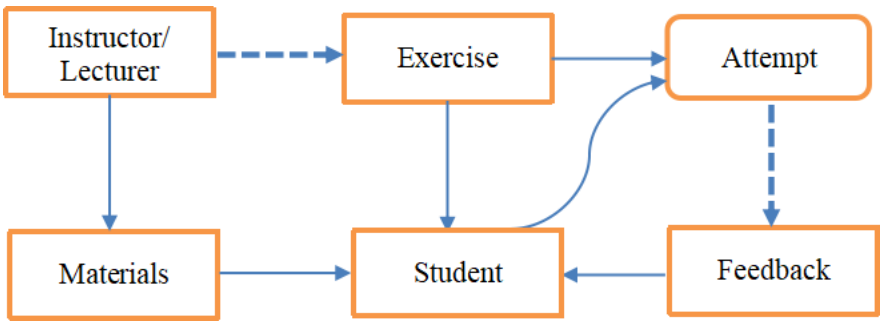


Figure 8.4: Lifecycle of the Programming Exercises

4.3.2.3 Technology Stack and Design Considerations

The system was built using a modular, cloud-based architecture to ensure maintainability and scalability. The front-end interface was developed using React.js to deliver a responsive and intuitive user experience. The backend was implemented with Node.js and Express.js, facilitating communication with both the OpenAI API and the database. Data management was handled using PostgreSQL, which securely

stores user profiles, learning progress, and AI-generated content. OAuth was implemented for secure user authentication and role-based access, while session management and data privacy features were aligned with standard ethical and security guidelines.

Communication between modules is enabled through RESTful APIs, and the system supports real-time interaction features via WebSockets, particularly in collaborative coding and messaging contexts. This architecture supports easy integration with external platforms such as online coding platform and GitHub, while maintaining a central user interface for unified learning experiences. The resulting framework is not only technically robust but also pedagogically aligned with the eight validated personalization features. The next section describes the implementation details of each core component and the specific role it plays in delivering adaptive, relevant, and effective programming instruction.

4.3.3 Core Functional Components

The architecture of the developed framework is composed of five interrelated functional components. Each component plays a distinct role in delivering the system's personalization capabilities, while collectively supporting adaptive, interactive, and context-aware learning experiences for programming students. These components were designed in alignment with both the pedagogical features identified in the literature and expert validation, and they form the backbone of the system's ability to offer individualized support, AI-generated instruction, and hands-on coding practice.

The components include the LLM Integration Module, which generates content and explanations in real time; the Adaptive Learning Tool, which adjusts learning difficulty based on user performance; the Collaborative Coding Platform, which enables students to experiment with and test their solutions; the LMS Interface, which houses all learning materials and user interactions; and the Database and Analytics Layer, which tracks learning progress and supports adaptive decision-making. Each module was developed using widely adopted technologies and structured software design practices to ensure modularity, accessibility, and future scalability.

4.3.3.1 LLM Integration Module

The Large Language Model (LLM) Integration Module serves as the intelligent component of the personalized programming framework. It enables real-time, context-aware interaction between learners and the system by interpreting user prompts, generating tailored instructional content, and providing dynamic coding feedback. Acting as a digital tutor, the LLM supports users with explanations, code hints, error analysis, and concept reinforcement aligning with the framework's goal of delivering personalized, adaptive learning experiences.

This module was developed using a RESTful API architecture built on Node.js, with Express.js handling routing and Axios managing communication with the OpenAI API for GPT-4o accessed through an API key. Prompt queries from the learner interface, developed in React.js, are captured and refined using structured prompt templates, a practice known as prompt engineering. For instance, when a student enters a basic query such as "Explain this code: for (i = 0; i < n; i++)", the system internally enhances it with pedagogical scaffolding like "Explain this C code to a beginner. Include a line-by-line explanation, describe the output, and mention common mistakes." This ensures the LLM delivers responses that are coherent, context-aware, and aligned with instructional goals.

The module directly supports several validated personalization features, including contextual relevance, clarity and coherence, prompt engineering effectiveness, and creativity. It dynamically structures explanations to match a learner's proficiency, invites multiple valid solutions, and ensures content accuracy and reliability confirmed during expert review. Learners interact with the LLM via a structured course navigation panel, where students select topics and submit prompts. The LLM responds with tailored explanations within the same environment. There is also an exercise interface where learners work on tasks such as checking whether a number is even or odd, with the LLM providing inline feedback or clarification based on learner input upon request.

4.3.3.2 Adaptive Learning Tool

The Adaptive Learning Tool is one of the core innovations of the framework, responsible for dynamically adjusting the difficulty and structure of programming

tasks based on individual learner performance. This component enables real-time personalization by continuously monitoring user interactions such as task completion time, code correctness, and the frequency of request for assistance and then automatically tailoring the next set of exercises or support materials to suit each learner's pace and proficiency level. In doing so, it ensures that students remain appropriately challenged without becoming overwhelmed, thereby maintaining engagement and promoting mastery.

The tool was developed using a rule-based adaptation technique, implemented in the backend using Node.js. During each learner session, the system evaluates multiple performance parameters: whether the student completed a task correctly, the time taken, how many hints were used, and the number of failed attempts. Based on predefined rules, the adaptive engine classifies the learner into a performance tier beginner, intermediate, or advanced. This classification then informs the selection of subsequent exercises. For instance, if a student completes two consecutive tasks quickly and correctly without assistance, the system automatically progresses them to a higher level of complexity, such as multi-dimensional arrays or pointer arithmetic. Conversely, if a student repeatedly requests feedback on loops or syntax errors, the tool delivers additional foundational tasks with step-by-step LLM support.

This adaptive functionality was integrated directly into the exercise routing logic of the learning interface. Once the student submits a task, a backend process checks their recent performance history from the PostgreSQL database, computes an engagement score, and determines the appropriate difficulty level for the next exercise. Exercises are stored in indexed difficulty bands, allowing the system to dynamically select content aligned to learner performance. To enhance transparency, feedback is provided at each stage, letting learners know when the system is offering reinforcement or elevating the challenge based on their recent successes.

From a pedagogical standpoint, the Adaptive Learning Tool addresses three key personalization features identified in this study. First, it embodies adaptability by modulating the learning experience according to real-time performance metrics. Second, it contributes to contextual relevance by ensuring that students engage with content appropriate to their current skill trajectory, avoiding both repetition and

premature advancement. Third, it reinforces feedback-driven progression, as learners are encouraged to reflect, revise, and build confidence through timely difficulty adjustments.

Initial testing of the tool during pilot implementation showed promising outcomes. Students reported that the progression of tasks felt "appropriately challenging" and appreciated the clarity in how the system responded to their performance. Instructors also noted that the tool helped identify struggling learners early by flagging repeated failures or extended task durations. While the adaptation logic is rule-based in this version, it is designed to support future integration with machine learning algorithms that could offer finer-grained personalization over time.

```
Input: task_result, completion_time, hint_count, failure_count
Output: next_exercise_level
BEGIN
 IF task_result == "correct" AND completion_time < threshold_time AND hint_count == 0 THEN
   success_counter += 1
   failure_counter = 0
 ELSE IF task_result == "incorrect" OR hint_count > max_hints THEN
   failure_counter += 1
   success_counter = 0
 END IF
 // Determine learner tier
 IF success_counter >= 2 THEN
   learner_tier = "advanced"
 ELSE IF failure_counter >= 2 THEN
   learner_tier = "beginner"
 ELSE
   learner_tier = "intermediate"
 END IF
 // Select next exercise from corresponding difficulty band
 next_exercise_level = getExercise(learner_tier)
```

```
  // Store decision in database
  logPerformance(learner_id, task_result, learner_tier, next_exercise_level)
  RETURN next_exercise_level
END
```

This pseudocode outlines the core logic behind the adaptive engine, which calculates the learner's current tier based on their ongoing task performance and selects the next task accordingly. This algorithm ensures the engine operates with consistent and transparent decision-making logic that can be scaled or refined in future implementations.

4.3.3.3 Collaborative Coding Platform

The collaborative coding platform provides learners with an interactive environment to write, test, and refine programming solutions. It serves as the practical coding space within the framework, allowing students to apply concepts learned through the LLM and adaptive learning tool. This component is integrated with a web-based compiler that enables real-time code execution, enhancing understanding through experimentation and iterative feedback. The platform was implemented using a browser-embedded editor that supports preloaded tasks linked from the learning interface. Students can write and run code directly in the browser without additional setup. The system also supports personalized AI hints and solution feedback via integration with the LLM module. It promotes learner engagement, peer comparison, and problem-solving autonomy core principles of personalized, exploratory learning.

4.3.3.4 Learning Interface and Communication Module

The learning interface serves as the central hub for accessing all learning content, AI interactions, and communication tools within the developed framework. Built using React.js, it was designed for responsiveness, modularity, and ease of use. Learners use the interface to navigate course topics, submit prompts to the LLM module, view AI-generated feedback, and track their progress through personalized dashboards. The interface supports OAuth-based authentication and interacts with backend services via RESTful APIs, ensuring secure and dynamic content delivery.

A core feature of this module is the real-time communication system, developed using Socket.io, which enables students to receive immediate notifications, chat with instructors, and engage in peer-supported discussion. These interactions are logged and used to inform adaptive learning decisions. The interface also supports prompt engineering by guiding learners in how to structure their queries for better AI output, and it promotes collaborative learning by facilitating instructor feedback and system-driven alerts. This intuitive design ensures that learners can access structured content, coding tasks, and feedback without needing any external setup or technical configuration. The interface screenshot is provided at appendix 5 together with the code of the entire framework pushed to GitHub.

4.3.3.5 Database and Analytics Layer

The database and analytics layer is the core data management component of the framework, enabling real-time and historical analysis to support personalization. Implemented using PostgreSQL, it securely stores structured learner data, including profiles, submissions, prompt histories, performance logs, and system-generated feedback. This data is critical for informing adaptive content delivery, particularly by the adaptive learning tool and LLM integration module. Analytics functionality, developed with Node.js and optimized SQL queries, allows the backend to track learning patterns and performance trends. A dedicated API connects the database to the LMS dashboard, enabling instructors to access key metrics such as task completion rates and common learner errors. Security is enforced through user authentication, role-based access control, encryption, and data anonymization to ensure ethical handling of sensitive information. This component directly supports personalization features like adaptability, accuracy, and feedback refinement, by enabling data-driven decisions based on long-term learner behaviour. During testing, it demonstrated high reliability, effectively supporting concurrent system activity while enhancing both instructional oversight and learner progression.

4.3.4 End-to-End Framework Workflow

This section outlines the sequential flow of operations within the developed LLM-based personalized learning framework, from user login to the final delivery of personalized learning outcomes. The objective is to provide a detailed account of how each functional component interacts with user data and system logic to deliver tailored

instruction in C programming. Upon launching the system, learners are required to log in using their institutional credentials. This authentication step ensures that individual learning profiles are securely accessed. Once authenticated, the system retrieves the user's historical interaction data, such as prior tasks attempted, feedback received, and recorded performance metrics.

Following authentication, the system executes a learner profiling procedure. This involves analyzing stored data to estimate the user's current proficiency level. Variables such as the number of errors in previous submissions, speed of task completion, and the nature of feedback consumed are used to categorize the learner into one of three predefined levels: beginner, intermediate, or advanced. This classification directly influences how prompts are structured, how feedback is delivered, and how content is sequenced. Learners then interact with the system by selecting or entering a programming task. The LLM interaction module preprocesses this input to ensure alignment with pedagogical goals. Prompts are dynamically structured based on the learner's profile, for example, a beginner's prompt may include scaffolded instructions, while an advanced learner might receive minimal guidance to encourage problem-solving autonomy.

The system forwards the contextualized prompt to the OpenAI API via a secure Node.js backend. This module formats the query, adds appropriate examples or constraints, and routes it for inference. The returned output includes code samples, explanations, or error diagnoses. These are post-processed to ensure syntactic and semantic correctness before display. One of the core innovations of the framework is the adaptive learning engine, which continuously monitors user interactions to update the learner model. If a user completes a task with high accuracy and minimal support, the engine increments their proficiency score, thereby increasing the difficulty level of subsequent tasks. Conversely, repeated errors or help requests trigger the assignment of simpler exercises and more detailed explanations. This adaptivity ensures that learning remains appropriately challenging without becoming frustrating.

The system optionally offers access to a collaborative coding interface where learners can engage in peer discussions or seek help on difficult tasks. This module operates asynchronously, logging interactions to inform future adaptations. It also contributes

to the relatedness component of self-determination theory by promoting a sense of academic community. After the learner submits a solution, the system immediately evaluates it using rule-based validators and heuristic checks. Identified errors are mapped to likely misconceptions, and the feedback engine retrieves targeted explanations and example fixes. These are displayed alongside the user's submission to promote reflection and correction. All learner actions, from prompt input to response consumption, are logged in a secure backend database. These logs enable instructors to monitor progress and allow the system to maintain continuity across sessions. The learner profile is updated in real time, forming the basis for subsequent task selection and feedback customization. Figure 4.5 illustrates the complete operational sequence of the developed framework.

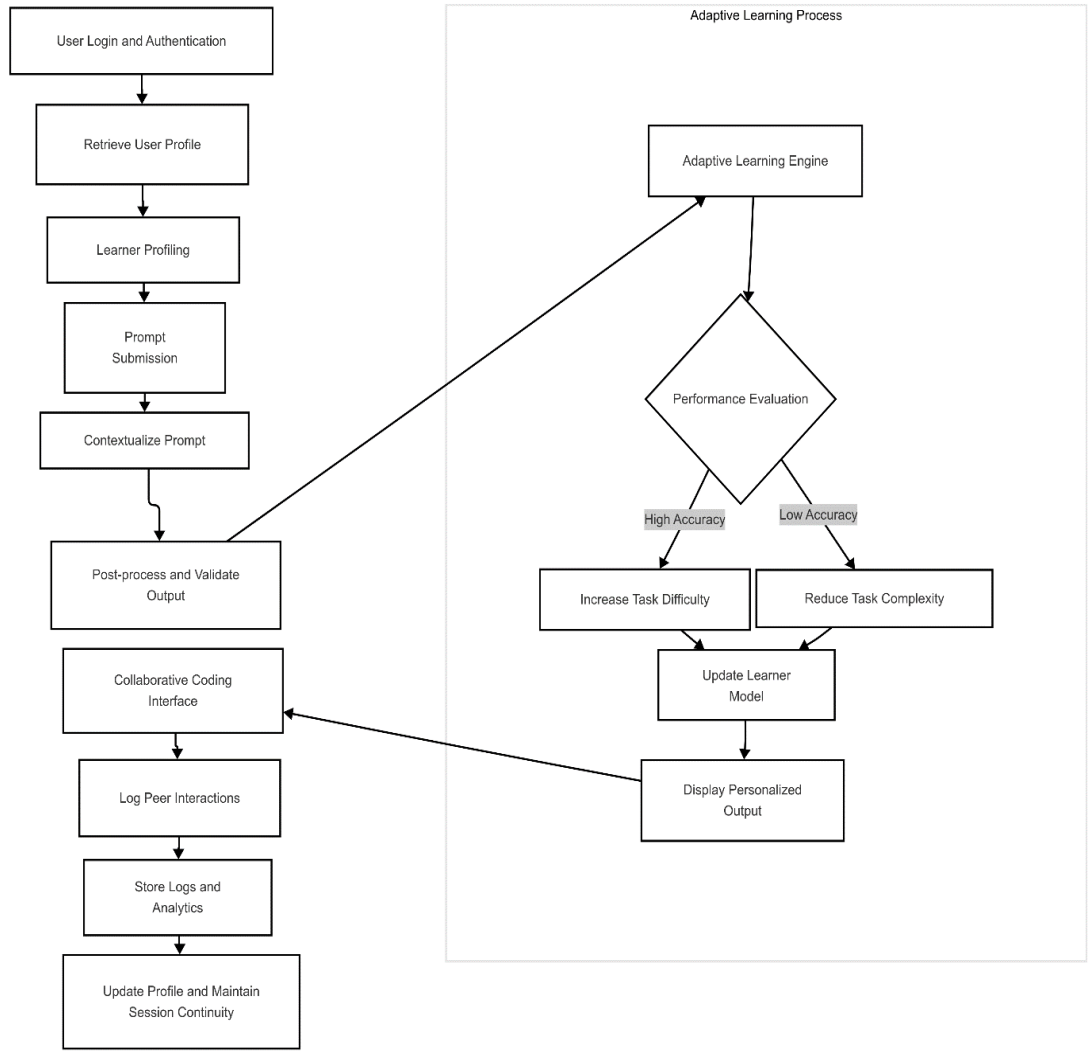


Figure 4.5: Hierarchical Flow of the Developed Framework

The flow begins with user authentication and learner profiling, followed by prompt submission and interaction with the LLM. The adaptive learning engine evaluates user performance and adjusts task difficulty dynamically. Personalized feedback is generated and displayed in real time. Optionally, learners may engage in collaborative coding. All interactions are logged and used to update learner profiles, ensuring session continuity and progressive personalization.

4.4 Evaluation of the Developed Framework

This section presents a comprehensive evaluation of the developed GenAI-powered programming education framework, drawing from survey data collected from 75 participants. The evaluation focused on usability, learning effectiveness, and, most importantly, the extent to which the system enhanced personalized learning.

4.4.1 Survey Design and Administration

The evaluation utilized a structured online questionnaire comprising both closed-ended (Likert-scale) and open-ended questions. The instrument was designed to gather feedback on key educational dimensions, including usability, learning motivation, personalization, and technical functionality. Quantitative data were analyzed using descriptive statistics to determine the average performance of different system features. To ensure the reliability of the instrument, internal consistency was evaluated using Cronbach's alpha. The personalization section of the questionnaire (comprising five Likert-scale items) yielded a Cronbach's alpha coefficient of 0.84, indicating high internal consistency and confirming the reliability of the instrument for assessing perceived personalization. Additionally, qualitative responses were thematically analyzed to contextualize numeric trends and identify emergent needs. A total of 75 students were invited to participate in the system evaluation, and all 75 completed the survey, yielding a response rate of 100%. This high response rate justifies the statistical validity of the survey results and indicates strong engagement from participants throughout the study period.

4.4.2 Quantitative Analysis of Personalization Features

One of the primary objectives of the study was to assess how well the framework enhanced personalized learning. To measure this, five Likert-scale items focused specifically on personalization were analyzed. Table 4.6 summarizes the findings.

Table 4.6: Student Ratings on Personalization Aspects (n = 75)

Personalization Aspect	Mean	Std. Dev.
The tool adapted to your individual learning needs	3.72	0.98
The tool offered personalized feedback or guidance that was useful for your learning	4.11	0.78
The suggestions and resources provided by the tool were relevant to your learning	3.96	0.83
You were motivated to complete assignments and exercises using the tool	4.27	0.98
The GenAI tool stimulated learning collaboratively or from peers	3.81	1.04

Figure 4.6 presents a visual summary of the participant's perceptions regarding the personalization features of the GenAI learning framework. The data highlights the system's strong performance in motivating learners, with the highest average rating of 4.27. This suggests that the adaptive task progression and timely feedback mechanisms were effective in sustaining learner engagement. Personalized feedback and guidance also received a high rating (4.11), affirming the tool's ability to tailor support to individual learning contexts. Similarly, the 3.96 rating for relevant content suggestions indicates that students perceived the AI-generated resources as aligned with their programming needs and goals.

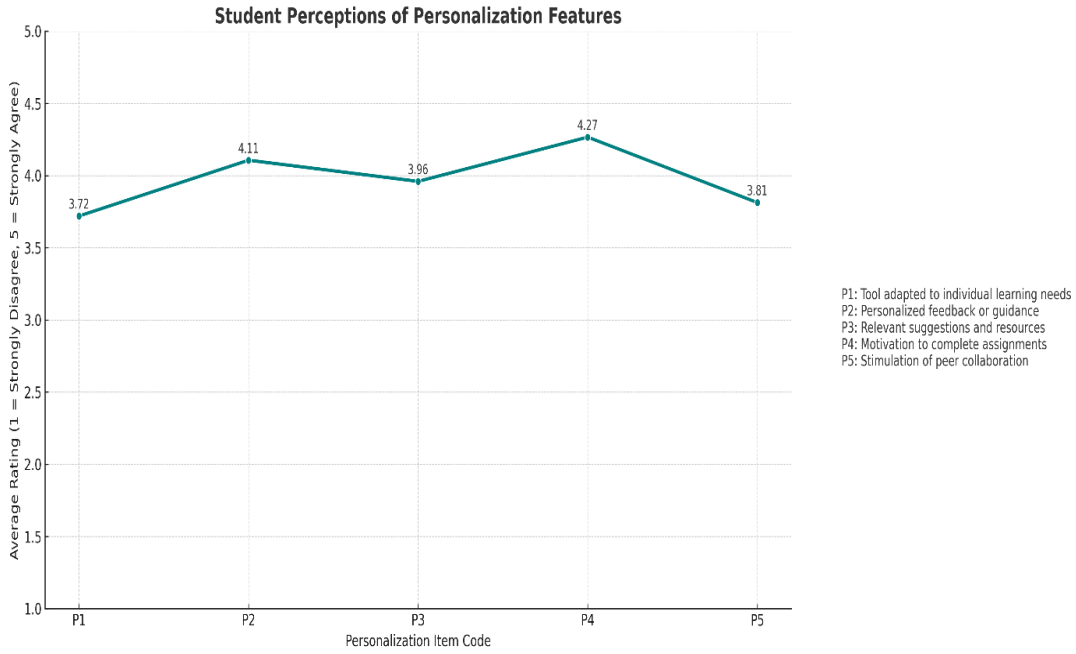


Figure 4.6: Student Perceptions of Personalization Features

In contrast, the tool's ability to dynamically adapt to each learner's performance scored moderately at 3.72, reflecting room for improvement in real-time responsiveness. Peer collaboration features received a slightly higher rating (3.81), but were still noted as limited likely due to minimal integration of interactive or group-based features. The higher standard deviation in this area (1.04) suggests varied student experiences, pointing to inconsistencies in perceived collaboration opportunities. Taken together, these results show that the framework successfully delivered a functional and appreciated personalized learning experience. However, the findings also emphasize the need for further refinement, especially in areas of dynamic adaptivity and collaborative learning integration, to fully realize the potential of personalized education using GenAI technologies.

4.4.3 Thematic Analysis of Qualitative Feedback

Thematic analysis was conducted manually using an inductive coding approach. Responses were first open-coded to identify recurring themes, followed by axial coding to categorize related concepts. Codes were reviewed iteratively to ensure consistency and were validated through peer discussion. NVivo software was not used; instead, spreadsheet-based matrix coding was applied to organize patterns. To complement the quantitative findings, open-ended responses collected from students

during the evaluation were thematically analyzed to gain deeper insights into their experiences with the GenAI-powered learning framework, particularly its personalization capabilities. Thematic coding followed an inductive approach, identifying both praised aspects and areas where students recommended improvement. Many students emphasized the clarity and structure of the explanations provided by the framework. They frequently mentioned that the system's ability to deliver line-by-line or step-by-step breakdowns of programming tasks significantly aided their understanding, especially when handling unfamiliar code. This structured guidance was noted to be particularly helpful for reinforcing core concepts and debugging errors independently.

Another widely cited strength was the personalized and timely feedback received after submitting queries or exercises. Students appreciated how the framework provided immediate and relevant responses, which they felt addressed their specific learning needs. This responsiveness not only helped them resolve problems more efficiently but also reduced dependency on instructor intervention. A number of respondents shared that the platform helped them build confidence in managing tasks on their own, thereby supporting independent learning. For many, the system's open-ended response style and provision of multiple solution paths encouraged critical thinking and exploration.

Despite the generally positive sentiment, the participants also offered constructive suggestions for enhancement. Language accessibility was mentioned by a few participants. Although the feedback was generally informative, they felt that the framework occasionally used overly technical or abstract terminology, which could overwhelm beginners. Simplifying the language or offering layered explanations, starting from the basics, was suggested as a way to improve accessibility for learners at different levels. A recurring recommendation was the integration of peer collaboration features. Students expressed interest in functionalities that would allow them to engage in real-time group tasks, compare solutions, or seek feedback from classmates. The thematic analysis reinforces the quantitative data by highlighting that the framework effectively supports personalization through structured feedback, relevant suggestions, and motivational support.

4.4.4 Pretest-Posttest Comparison

To objectively assess the effectiveness of the GenAI-powered personalized programming framework, a pretest-posttest experimental design was implemented involving two distinct student groups. The experimental group (n = 75) used the GenAI-enhanced learning system, while the control group (n = 64) received traditional classroom instruction over the same period. Both groups were administered equivalent pretests and posttests aligned to the same programming course objectives. Table 4.7 summarizes the average scores for both groups. While the control group began with a slightly higher pretest mean (60.94) compared to the experimental group (57.16), the experimental group demonstrated a significantly greater learning gain. After the intervention, the experimental group improved to a mean score of 66.24, indicating a 9.08-point increase, compared to a 2.14-point increase in the control group.

Table 4.7: Group Performance on Pretest and Posttest Evaluations

Group	Pre-test Mean	Post-test Mean	Mean Gain
Control Group	60.94	63.08	+2.14
Experimental Group	57.16	66.24	**+9.08**

These findings are further illustrated in Figure 4.7, which presents a visual comparison of performance progression across the two groups. The steeper trajectory in the experimental group highlights the positive impact of the GenAI-based framework in enhancing students' programming skills. The results suggest that the adaptive instructional design and real-time feedback features integrated into the GenAI framework contributed substantially to improved learning outcomes. Despite beginning with a lower baseline, students in the experimental group outperformed their traditionally taught peers in the posttest phase.

Both the experimental and control groups had previously completed the same foundational C programming module as part of their curriculum, ensuring a comparable entry behavior and basic understanding of concepts such as syntax, data types, and control structures. The slightly higher pretest performance of the control group (mean = 60.94) compared to the experimental group (mean = 57.16) is attributed to natural variation and does not indicate significant baseline differences. To minimize

confounding factors, both groups received equivalent instructional content, with the only difference being the use of the GenAI-powered framework by the experimental group. This controlled approach supports the conclusion that the observed performance gain in the experimental group was predominantly due to the intervention.

A Mann-Whitney U test was conducted to compare the posttest scores of students in the experimental group and those in the control group. The results revealed a significant difference in learning outcomes between the groups ($U = 1725.5$, $p < 0.008$). This indicates that students who used the GenAI-powered learning framework demonstrated significantly greater improvements compared to those taught through traditional methods.

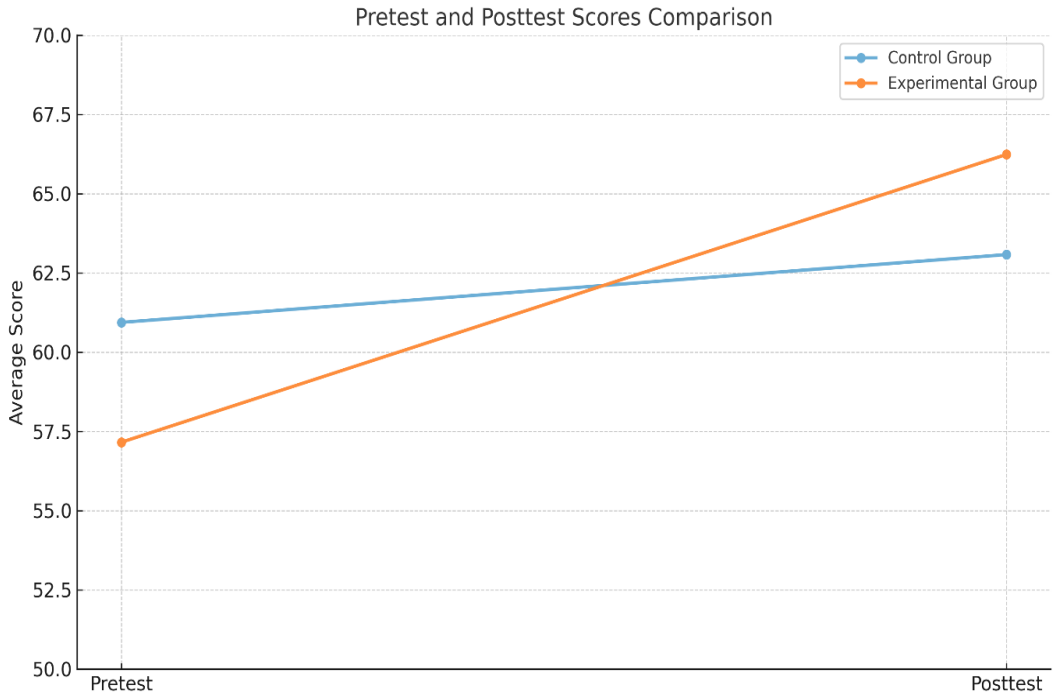


Figure 4.7 Pretest and Posttest Scores Comparison

4.4.5 Framework Behavior in Personalized Code Generation

This section demonstrates how the developed framework dynamically adapts its instructional output based on individual learner data. The example provided illustrates a scenario in which a beginner-level student interacts with the system to receive personalized support while learning C programming. The objective is to present the

internal logic and flow of operations that result in tailored code generation, thereby validating the framework's responsiveness to learner profiles.

In this instance, a student logs into the system and initiates a request to generate a basic C program that determines whether a number is even or odd. Upon authentication, the system automatically retrieves and analyzes the learner's profile, which includes historical performance indicators such as frequent syntax errors, high reliance on feedback tools, and prolonged task completion times. These data points are interpreted by the learner profiling engine to classify the user as a novice. Once the learner's proficiency level is confirmed, the adaptive learning engine is triggered. This component is responsible for selecting the appropriate instructional complexity and content formatting. In this case, the engine identifies the task as suitable for beginner-level instruction and prepares to scaffold the learning experience accordingly. The prompt handler then reformats the learner's initial input by embedding guiding instructions for the LLM. Specifically, the system adds meta-instructions that request the model to include explanatory comments, simplify logic, and avoid advanced syntax constructs. This process is invisible to the user but serves to shape the LLM's output in a pedagogically responsive manner.

The structured prompt is sent through the LLM interaction module, which relays the query to the OpenAI API via a secure backend. The returned output undergoes internal validation to ensure that the generated code is syntactically correct, coherent, and aligned with the prompt's instructional intent. Below is the actual code produced by the system in response to the learner's query:

```
#include <stdio.h>

int main() {
    int number;

    // Prompt the user for input
    printf("Enter an integer: ");
    scanf("%d", &number);
```

```
    // Check if the number is even using modulo operator
    if (number % 2 == 0) {
        // If divisible by 2, it's even
        printf("%d is even.\n", number);
    } else {
        // Otherwise, it's odd
        printf("%d is odd.\n", number);
    }

    return 0;
}
```

The generated output is noticeably beginner-friendly. It includes a high density of inline comments to support code comprehension and explain the logic behind the even-odd check. The syntax is deliberately simplified to exclude advanced structures, thereby reducing cognitive load for the learner. Console prompts are phrased clearly, enhancing user interaction and minimizing confusion.

Internally, this behavior reflects the collaboration of several components within the system. The learner profiling engine establishes the user's knowledge level, while the adaptive learning engine selects the difficulty of the response. The prompt engineering strategy, managed by the prompt handler, then guides the LLM to return content that is educationally aligned with the learner's needs. Finally, the feedback engine prepares the output for display and updates the learner model with new interaction data to inform future responses.

4.4.6 Comparative Framework Analysis

This section presents a structured comparison between the developed personalized learning framework and selected existing frameworks. The purpose of the comparison is to highlight how the proposed framework aligns with or improves upon prior approaches, especially in terms of adaptability, learner modeling, and instructional delivery. The comparison was conducted based on an evaluation of key functional and pedagogical features derived from literature, including real-time feedback, learner profiling, adaptability, prompt structuring, and collaborative support. Three existing

frameworks were selected for comparison with the developed framework: (i) ITS-CAL: A rule-based intelligent tutoring system, (ii) Artemis GPT-3.5 Tutor: A natural language-based AI tutor, and (iii) TutorLLM: A GPT-powered educational assistant with prompt templates. Each framework was analyzed using publicly available documentation, published evaluations, and where applicable, hands-on pilot testing (for Artemis and TutorLLM). The comparison focused on whether these systems addressed the challenges identified in the literature review, and how they implemented personalization and adaptivity in instructional delivery. Table 4.8 provides a high-level summary of these distinguishing features.

Table 4.8: Comparison of the Developed Framework with Existing Frameworks

Feature / Metric	**ITS-CAL**	**Artemis GPT-3.5 Tutor**	**TutorLLM**	**Proposed Framework**
Real-Time Feedback	✗	✔	✔	✔
Adaptive Learning Engine	✗	✗	Limited	✔ (Continuous)
Learner Profiling	✗	✗	Initial only	✔ (Ongoing)
Prompt Engineering Strategy	Static	Semi-Structured	Templated	Dynamic + Context-Aware
Collaborative Coding Support	✗	✗	✗	✔
Task Adaptation Based on History	✗	✗	Partial	✔
Instructional Coherence	Moderate	Variable	Moderate	High
Ease of Deployment	High	Moderate	Low	Moderate
Scalability and Modularity	Limited	Moderate	Limited	High

The comparison clearly demonstrates that while existing systems such as ITS-CAL and Artemis GPT-3.5 Tutor incorporate elements of intelligent feedback and natural

language interaction, they lack robust support for real-time learner modeling and adaptability. ITS-CAL is largely rule-based and static, while Artemis, though built on GPT-3.5, lacks a structured pedagogical engine and provides variable-quality feedback depending on prompt complexity. TutorLLM introduces prompt templating and some user-level adaptation but does not sustain learner profiling over time, limiting its instructional personalization.

In contrast, the developed framework integrates multiple advanced features into a unified ecosystem. It continuously monitors user progress, dynamically adjusts task difficulty, and generates instructional content tailored to individual learner profiles. The inclusion of collaborative coding further differentiates it, offering peer interaction within a personalized learning space. The system's modular backend and frontend design also ensure greater adaptability to different instructional contexts, making it more scalable than many existing solutions. The comparison confirms that the developed framework effectively addresses key limitations in previous systems, offering a more adaptive, learner-centered, and pedagogically aligned solution for programming instruction.

4.5 Results

This section presents the consolidated results of the study in relation to the three research objectives. The findings are organized into two subsections: theoretical results, which include the outcomes of the systematic literature review and expert validation; and empirical results, derived from the evaluation of the implemented framework through classroom deployment and experimental testing. The goal of this section is to demonstrate how the developed framework operationalizes validated features of personalization in computer programming education, and to provide evidence of its effectiveness from both conceptual and practical standpoints.

4.5.1 Expert Survey Results

The theoretical results of this study are grounded in the outcomes of the systematic literature review and the subsequent expert validation process. These two phases addressed the first research objective of the study. The outcomes presented in this section reflect how personalization in programming education was conceptualized, validated, and embedded into the structure of the developed system.

4.5.1.1 Systematic Review Findings

The SLR involved a multi-stage screening and synthesis process that yielded 42 relevant studies published between 2020 and 2024. These studies were drawn from peer-reviewed journals and conference proceedings and focused on Generative AI applications, LLM-based instruction, and adaptive learning frameworks within programming education. Through thematic analysis of these studies, eight core features were identified as critical to supporting personalized learning environments powered by AI. These features include: adaptability, contextual relevance, clarity and coherence, accuracy and reliability, prompt engineering effectiveness, practical usability, interactivity and collaboration, and creativity and uniqueness. Adaptability emerged as the most emphasized feature, with many studies highlighting the ability of AI tools to tailor task difficulty, feedback, and pacing to the learner's performance. Contextual relevance together with clarity and coherence were also prominent, underscoring the need for AI-generated content to align with real-world programming scenarios and follow logical explanatory structures. Accuracy, particularly in code generation and syntax validation, was repeatedly noted as essential for maintaining credibility in programming education.

Prompt engineering was identified as a foundational mechanism that underpins the quality of AI output, determining the relevance and coherence of responses. Several studies also emphasized the importance of feedback that is responsive to learner inputs, thereby enabling iterative learning. Additionally, frameworks that support creativity such as through open-ended problem-solving and multiple solution paths, were noted to enhance learner engagement. Finally, studies underscored the importance of delivering all these features through platforms that are practically usable: accessible via web browsers, intuitively navigable, and minimally dependent on technical setup. This synthesis formed the theoretical basis for the framework, guiding the specification of system features and influencing architectural decisions presented in Section 4.3.

4.5.1.2 Expert Validation of Identified Features

To ensure that the features identified in the review were pedagogically relevant and practically applicable, an expert validation exercise was conducted involving 20 participants. These included university lecturers, ICT industry professionals, senior-

level programming students, and recent graduates. The validation process used a structured questionnaire, composed of Likert-scale items and open-ended questions, to assess expert agreement with the relevance and applicability of each feature. The results revealed strong expert endorsement across all eight features. In particular, there was high agreement on the importance of adaptability and prompt engineering in improving AI-generated instructional content.

Experts emphasized the need for content that could adapt to the learner's pace and prior knowledge, and for AI outputs that were logically structured, technically accurate, and pedagogically aligned. Ratings on items related to relevance, accuracy, and structured explanation consistently scored above a mean of 4.0 on the five-point Likert scale, while the feature of personalized feedback was frequently mentioned in open-ended responses as essential for promoting learner autonomy and reducing frustration. These validation results confirmed that the framework's design not only reflected theoretical best practices but was also aligned with stakeholder expectations and practical classroom realities. This strengthened the internal validity of the features and provided further justification for their integration into the system.

4.5.1.3 Feature Integration in the Framework Design

The theoretical features validated through the SLR and expert survey were subsequently embedded into the design of the framework through dedicated functional components. The adaptive learning tool operationalized adaptability by adjusting task difficulty based on student performance metrics. The LLM integration module was configured with prompt templates to improve contextual relevance, coherence, and accuracy of responses. Personalized feedback was enabled through dynamic hint generation and iterative task support. The platform's web-based interface and seamless access to an online coding environment ensured practical usability, while creative exploration was facilitated by open-ended prompts and opportunities for learners to submit diverse solution paths. In directly aligning each personalization feature with a system component, the framework translated theoretical insights into a cohesive and functional architecture. This implementation validated the framework as a theory-informed, practically accessible tool for enhancing personalized programming education.

4.5.1.4 Learner Reflections on Personalization and Learning Experience

The qualitative responses collected through the open-ended section of the framework survey provided further insight into how students experienced the personalization capabilities of the framework. Thematic analysis of the responses revealed recurring patterns in how learners described their interactions with the system. A majority of students emphasized the value of real-time, context-specific feedback. Many noted that the tool offered step-by-step guidance and relevant code explanations that helped them correct their mistakes without external assistance. Several students described the tool as "functioning like a tutor" that adapted to their level of understanding and provided support when they encountered difficulties.

Learners also appreciated the ability to progress at their own pace, revisiting prompts or reattempting exercises based on the feedback they received. Others highlighted the motivational aspect of being able to solve problems independently with the support of the AI, stating that the system made programming feel less intimidating and more engaging. The personalization experience, as described by students, aligned closely with the system features designed to promote adaptability, feedback, contextual relevance, and practical usability. The LLM-generated responses, supported by prompt engineering and retrieval mechanisms, contributed to a guided and supportive learning environment.

4.5.1.5 Challenges and Areas for Improvement

Despite the overwhelmingly positive reception, the participants identified a few limitations that warrant further attention. Some respondents reported occasional slow response times, likely due to internet connectivity or system resource constraints. A few learners expressed concerns over the consistency of AI-generated responses, noting that certain explanations or feedback items appeared overly generic or lacked clarity when dealing with more complex programming logic. Although the platform allowed for real-time messaging and interaction, students felt that opportunities for peer-to-peer learning and shared problem-solving could be improved. These findings suggest that future iterations of the framework should place greater emphasis on building collaborative coding scenarios and refining the LLM's ability to handle nuanced coding problems.

4.5.2 Experimental Results

The empirical results of this study provide evidence of the practical impact of the developed LLM-based learning framework when deployed in a real educational context. This section presents findings derived from the classroom implementation of the framework, structured around the third research objective, which sought to evaluate the system's effectiveness in improving student engagement, learning outcomes, and content quality in computer programming education. To obtain robust empirical insights, the evaluation adopted a mixed-methods approach involving quantitative data collection through a structured Likert-scale questionnaire, qualitative data through open-ended survey items, and experimental comparison of learning outcomes using a pretest-posttest design. The framework was implemented in a classroom setup comprised of 75 students enrolled in an introductory course on the C programming language.

4.5.2.1 Evaluation Through Survey Responses

The primary source of quantitative feedback was a 15-item Likert-scale questionnaire administered after students interacted with the framework. The items were grouped under five thematic areas: ease of use, learning enhancement, engagement and motivation, personalization, and overall satisfaction. Each item was rated on a five-point scale ranging from 1 (strongly disagree) to 5 (strongly agree). The results revealed strong positive perceptions across all five evaluation categories. Among these, the highest mean scores were recorded under learning enhancement with a mean of 4.14 out of 5 and a standard deviation of 0.91. Ease of use had a mean of 4.13 and a standard deviation of 0.89, suggesting that students found the system effective in supporting their learning and intuitively navigable. The dimension of personalization recorded a mean score of 3.86 and standard deviation of 0.91, which, while comparatively lower than other categories, still indicated a positive reception of the system's adaptive capabilities.

The participant's responses to items within the personalization category further underscored the system's impact. Many students agreed that the content and feedback were tailored to their level of understanding and performance. Specifically, students reported that the tool adjusted the difficulty of exercises based on their previous responses and offered contextual hints that facilitated independent problem-solving.

These findings support the claim that the personalization mechanisms embedded in the framework were both noticeable and meaningful to learners.

4.5.2.2 Learning Outcome Comparison Through Pretest-Posttest Design

To assess the effectiveness of the GenAI-powered learning framework in improving programming competencies, a pretest-posttest evaluation was conducted comparing two groups: the experimental group (n = 75), which used the developed framework, and a control group (n = 64), which received conventional instruction without GenAI support. Both groups were administered equivalent assessments before and after the learning intervention. Each test comprised ten programming-based questions totaling 50 marks, covering topics such as C syntax, control structures, functions, arrays, pointers, and debugging. As summarized earlier in table 4.7, the experimental group recorded a pretest mean of 57.16 and improved to a posttest mean of 66.24, yielding a gain of +9.08 marks. The control group began with a higher pretest mean of 60.94 and achieved a posttest mean of 63.08, resulting in a comparatively lower gain of +2.14 marks.

To determine whether the observed performance difference between the groups was statistically significant, a Mann-Whitney U test was conducted on the posttest scores. This non-parametric method was chosen due to the nature of the data and the lack of assumption of normality. The test yielded a U-value of 1,725.5, a Z-score of -2.634, and a p-value of 0.008, indicating that the difference between the experimental and control groups was statistically significant at the $p < 0.01$ level. The median posttest score for the experimental group was 67, compared to 63 for the control group. Table 4.9 illustrates the Mann-Whitney U Test results of posttest score.

Table 4.9: Mann-Whitney U Test Results

Test	U	Z	p-value (2-tailed)	Median (Control)	Median (Experimental)	Interpretation
Post-test Score Comparison	1,725.5	-2.634	0.008	63	67	Statistically significant difference ($p < .01$)

The results demonstrate the practical value of the GenAI-based framework in enhancing student learning outcomes. Despite beginning with a lower average performance, the experimental group surpassed the control group in posttest scores. This outcome highlights the framework's effectiveness in providing adaptive feedback, structured task sequencing, and context-aware guidance that supports learner progress, particularly for those who may start with weaker programming skills. The statistically significant results validate the framework's capacity to deliver personalized, impactful learning experiences in programming education.

4.6 Discussion

This section interprets the theoretical and empirical results of the study, providing a critical reflection on how the findings respond to the research objectives and contribute to the field of computer programming education. The discussion is structured around the dual focus of the study: (i) identifying and validating features of personalization in AI-assisted learning, and (ii) implementing and evaluating a working framework that integrates these features into actual practice. Sub-section 4.6.1 focuses on theoretical implications, while 4.6.2 addresses empirical insights derived from the deployment of the framework.

4.6.1 Implication of Expert Review Results

The theoretical results of this study offer several significant implications for the ongoing discourse on the use of Generative AI in personalized education, particularly within the domain of computer programming instruction. Through a systematic literature review and expert validation, the study identified eight core features that are

foundational to effective personalization. The integration of these features into a single cohesive framework marks a departure from earlier models that focused on isolated capabilities such as adaptive quizzes or AI-assisted grading. Instead, this study presents a holistic vision of personalized programming education, wherein every stage of the learning process from task generation and instructional delivery to feedback and revision is informed by principles grounded in empirical research and expert consensus.

One of the key theoretical contributions of this study lies in its formulation of a feature-to-function mapping approach. Rather than treating personalization as an abstract concept, the study demonstrates how each feature can be operationalized through concrete system components. For example, adaptability is implemented through a dynamic learning engine that adjusts content complexity, while prompt engineering is translated into structured interaction patterns with the LLM to improve the coherence and accuracy of responses. This mapping provides a practical blueprint for future systems seeking to embed GenAI technologies within pedagogical models.

Furthermore, the study confirms that prompt engineering is not merely a technical consideration, but a pedagogical design strategy. In treating prompts as instructional scaffolding, the framework leverages LLMs not only to generate content, but to generate content that is tailored, contextualized, and pedagogically appropriate. This reinforces the theoretical positioning of LLMs as cognitive partners in the learning process, rather than static content generators. The validation of the identified features through expert input also adds a layer of credibility and pedagogical relevance to the framework. The alignment between literature-derived themes and expert consensus indicates that the framework is both theoretically sound and practically applicable. It bridges the gap between educational research and technological innovation, contributing to the broader body of knowledge on how artificial intelligence can be responsibly and effectively integrated into formal learning environments.

The qualitative feedback offered by students enriches this perspective, revealing that many learners experienced the system as a form of digital mentorship. Descriptions such as “felt like a tutor” and “guided me step-by-step” highlight the tool’s perceived instructional value beyond traditional content delivery. These experiences validate the

human-centered design of the framework, which emphasized personalization not merely as technical adaptability, but as a pedagogical experience characterized by relevance, guidance, and interactivity.

The theoretical results of this study underscore the importance of design-driven personalization in AI-powered learning systems. They suggest that effective GenAI frameworks for programming education must go beyond novelty and automation, and instead embed educationally grounded principles that support the learner's development through intelligent, responsive, and context-aware interaction. This theoretical foundation not only informed the design of the current framework but also offers a replicable model for future research and system development in AI-supported personalized learning.

4.6.2 Implication of Experimental Results

The empirical results of this study offer compelling evidence regarding the practical impact of the developed LLM-based framework on student engagement, content understanding, and the overall learning experience in programming education. These findings not only reinforce the theoretical design decisions that shaped the framework but also provide valuable insights into how such technologies can be effectively deployed in real instructional settings to enhance personalization.

One of the most significant implications arising from the empirical evaluation is the confirmation that students are not only receptive to AI-powered educational tools but can also benefit meaningfully from their adaptive capabilities. The positive ratings recorded across the five evaluation dimensions, particularly learning enhancement and ease of use, indicate that students found the framework intuitive and pedagogically effective. Importantly, the personalization category reflected strong agreement that the system responded to individual learning needs through contextual feedback, scaffolded assistance, and adaptive content delivery.

These findings suggest that students are increasingly prepared to engage with intelligent educational technologies, provided that these tools are thoughtfully designed to meet pedagogical objectives. The data shows that learners recognized the value of step-by-step feedback, dynamic exercise adjustments, and immediate

clarification of coding errors, all of which are hallmarks of personalized learning experiences. Such outcomes imply that LLMs, when properly configured, can serve as powerful mediators of individualized learning, particularly in domains that require iterative practice and conceptual reinforcement like programming.

The experimental comparison further underscores the framework's educational value. Although both the control and experimental groups showed improvement in posttest scores, the Mann-Whitney U test results confirmed a statistically significant difference in favor of the experimental group ($U = 1{,}725.5$, $p = 0.008$). Despite starting with a slightly lower pretest mean (57.16 vs. 60.94), the experimental group achieved a higher posttest median score (67 vs. 63) and a greater mean gain (+9.08 marks vs. +2.14 marks). These results validate that the improvement experienced by students in the experimental group was not due to chance but can be attributed to the personalized instruction provided by the framework.

Notably, the framework was particularly beneficial for students who entered the course with lower baseline scores, helping to close achievement gaps through targeted feedback and adaptive support. This has broader implications for equity in education, as AI-driven personalization can support differentiated instruction in diverse learning environments. However, the empirical results also point to important considerations for future development. While most students appreciated the system's features, a minority cited issues such as technical lag and limited support for peer collaboration. These concerns highlight the need for ongoing enhancement of collaborative tools and infrastructure optimization to ensure reliable access and response time. Addressing these limitations is essential to ensure that the personalization experience is not only rich and adaptive but also dependable and scalable.

The empirical findings of this study contribute to the growing body of evidence that supports the integration of large language models into instructional practice. The study demonstrates that personalization is not merely a theoretical affordance of AI but a lived experience that can positively shape learner outcomes when designed with care. The success of this framework provides a model for how LLMs can be harnessed to create intelligent educational environments that adapt in real time, offer targeted support, and empower learners to take ownership of their learning process. The

empirical results establish that the developed framework fulfills its intended purpose: it personalizes the programming learning experience, improves learner engagement, supports differentiated instruction, and contributes to measurable academic progress. These implications underscore the transformative potential of LLM-based systems in reshaping how computer programming is taught and learned, especially in contexts where scalability, feedback, and learner-centered design are critical.

Furthermore, the findings align with the Self-Determination Theory (SDT), particularly in satisfying the psychological needs of competence and autonomy. The adaptive feedback mechanism of the GenAI tool gave learners immediate, personalized responses that helped them feel more capable in handling programming tasks, directly supporting the competence need. Similarly, the tool's on-demand nature and ability to scaffold learning without constant instructor intervention enabled students to exercise greater autonomy in navigating their learning paths. These outcomes are consistent with prior studies (e.g., Chen et al., 2020; Kulik & Fletcher, 2016), which reported that intelligent tutoring systems supporting autonomy and mastery led to increased learner motivation and better retention. However, other studies (e.g., Cope et al., 2021) caution that over-reliance on AI guidance can sometimes reduce critical thinking if learners are not encouraged to independently reflect or experiment. Unlike these systems, the current framework encouraged exploration by providing multiple solution paths and explanatory feedback, which several students described as "mentorship" rather than "answers." This distinction demonstrates that when GenAI systems are designed with pedagogical awareness, they can balance guidance with learner independence, resulting in more effective and equitable learning environments. These connections not only reinforce the practical outcomes observed but also situate the results within broader pedagogical theory and discourse.

Although the results strongly suggest that the GenAI-powered framework contributed to improved programming outcomes, alternative explanations must be considered. For instance, the novelty effect associated with using advanced AI tools may have temporarily boosted student engagement and motivation, independent of the framework's instructional design. Additionally, learners in the experimental group may have possessed unmeasured motivational or digital literacy advantages, which

could have contributed to their improved performance. Another possibility is that the instructor's familiarity with the experimental tool influenced the quality or clarity of guidance received, thereby affecting learning gains. These alternative explanations underscore the importance of conducting follow-up studies with randomized assignment and multi-site trials to rule out context-specific biases.

Regarding scalability, while the framework has shown promising results in a controlled environment and within the English-speaking Kenyan context, deploying it in other educational settings or languages requires careful adaptation. The personalization engine, for example, relies heavily on natural language understanding; thus, linguistic and cultural nuances must be considered when localizing prompts and responses. Furthermore, differences in curriculum structure, assessment formats, and digital infrastructure across institutions may affect how well the system performs elsewhere. Despite these challenges, the modular and language-agnostic design of the framework, including its customizable prompt templates and open-source architecture, offers a scalable foundation for adaptation across diverse learning contexts and languages.

4.7 Summary

This chapter presented the results of evaluating the GenAI-based personalized programming framework. Data were collected from students and experts through structured questionnaires and pre- and post-tests. Quantitative results showed a statistically significant improvement in learning outcomes for the experimental group, with a mean gain of 9.08 points compared to 2.14 points in the control group. The Mann-Whitney U test confirmed that this difference was statistically significant ($p = 0.008$), supporting the conclusion that the improvement was attributable to the personalized instruction provided by the framework. Student feedback highlighted high satisfaction with the tool's usability, relevance, and real-time feedback. The framework's effectiveness was also evaluated through a detailed comparative analysis, contrasting it with other existing AI-powered instructional systems. This analysis demonstrated that the developed framework offered superior adaptability, personalized guidance, and real-time support, especially due to its integrated components such as the adaptive learning tool, collaborative coding platform, and LLM module. An illustrated walkthrough of the framework's operation, from user

login to personalized content delivery, was provided to explain how each component contributes to the overall learning experience. A sample code output generated through the system was also included to demonstrate how learner-specific prompts yielded context-aware, customized programming support. These findings confirm the framework's potential to support personalized, engaging, and effective learning experiences in programming education.

CHAPTER FIVE

CONCLUSION, RECOMMENDATIONS AND FUTURE WORK

5.1 Conclusion

This research set out to design and evaluate a large language model-based framework aimed at enhancing personalized learning in computer programming. Through conducting a comprehensive systematic literature review, validating the findings through expert input, and developing and evaluating a fully functional learning framework, the study successfully met its three research objectives. This section summarizes how each of these objectives was successfully achieved and reflects on the overall performance and significance of the developed framework.

First, the study identified eight critical features of personalization in AI-enhanced education through an extensive literature search. These features were rigorously extracted from 42 peer-reviewed sources and validated by a group of experienced educators, industry practitioners, and advanced learners. The features directly informed the framework's design, ensuring that it addressed specific gaps such as prompt personalization, continuous profiling, and adaptive content delivery. Second, the research operationalized these features into a functional web-based framework. The framework integrated key components such as an LLM-based interaction module, adaptive learning engine, collaborative coding platform, and intuitive user interface. Each of these modules was carefully integrated to work cohesively, with learner data flowing across components to enable real-time decision-making. The framework was designed to be scalable, easy to deploy, and capable of continuously refining its instructional responses based on learner behavior.

To address the third objective, the framework was empirically tested in a real classroom environment involving 139 undergraduate students enrolled in introductory programming courses. The participants were divided into control and experimental groups. The experimental group used the developed system, while the control group relied on traditional learning platforms. Analysis of pre- and post-test results revealed a significant improvement in the performance of students who used the personalized framework. This demonstrates that the framework not only functioned as intended but also led to measurable learning gains.

The evaluation demonstrated significant improvements in student engagement, satisfaction, and learning outcomes, supported by both statistical data and thematic insights from learner feedback. Students experienced the tool as responsive, helpful, and effective in supporting their programming skills development. Overall, this research contributes to the growing field of AI in education by presenting a validated, deployable, and pedagogically grounded framework for enhancing personalized learning. The study confirms that when designed with intentionality, LLMs can serve as powerful tools for transforming the way programming is taught and learned in higher education.

5.2 Recommendations

5.2.1 Recommendations for Practice

Higher education institutions, particularly those offering computing and engineering programs, should integrate LLM-powered systems into foundational programming courses. This curriculum integration can help address early learning challenges related to syntax, debugging, and logic comprehension. The actors responsible for initiating and overseeing this recommendation include curriculum committees and department heads. In addition, academic institutions should introduce mandatory AI literacy modules. These modules should help students understand how large language models function, how to engage with them through effective prompt engineering, and how to critically evaluate AI-generated outputs. This responsibility lies with teaching and learning centres within universities.

To ensure successful classroom deployment, instructors must also receive structured training in the use of generative AI. This includes learning how to craft pedagogically aligned prompts and interpret LLM responses to support differentiated instruction. Academic staff development units should take the lead in facilitating such capacity-building programs. Finally, to promote ethical awareness, educational institutions must incorporate specific instruction on AI bias and mitigation strategies. Students should be taught how to identify, interpret, and minimize algorithmic bias in AI-generated educational content. Ethics committees and curriculum designers are the appropriate actors to implement this recommendation.

5.2.2 Recommendation for Policy

In the policy domain, universities must establish clear and enforceable guidelines regarding the use of AI in academic contexts. These policies should address critical issues such as academic integrity, disclosure of AI assistance, and acceptable limits for AI use during assessments. The responsibility for enacting these policies lies with university senates and policy-making boards. Furthermore, institutions should promote transparency in how AI tools are integrated into teaching and learning. This includes documenting how AI systems are deployed, trained, updated, and evaluated. IT services and data governance teams should lead this initiative to ensure accountability and inclusiveness. Equally important is the need to invest in robust, scalable infrastructure to support the use of intelligent learning environments. This includes ensuring that cloud-based systems are secure, accessible, and optimized for classroom use. University ICT departments and finance teams should collaborate to plan and implement these infrastructure improvements.

5.2.3 Recommendation for Industry

From an industry perspective, the framework developed in this study can be extended to corporate training and professional development contexts. Companies can adopt the model to upskill new hires, retrain existing staff, or support employees transitioning into software development roles. This task is most appropriately managed by human resources departments and learning and development (L&D) managers. To enhance real-world relevance, industries are encouraged to form strategic partnerships with academic institutions. These partnerships can facilitate the piloting of AI-assisted learning systems in workplace environments, especially for onboarding and technical training. Tech companies and innovation hubs should take the lead in building and maintaining such collaborations. Finally, there is great potential in customizing the framework for specific industry needs. For example, adapting it for training in embedded systems, data security, or other niche programming domains can increase its practical utility. This recommendation is best executed by industry training consultants who understand the demands of specific technical fields.

5.3 Future Work

Several important directions remain for future research and development. One key area is the scalability of the proposed framework. Future studies should evaluate how

the system performs across various educational institutions and cultural settings to assess its broader applicability. Another area for expansion involves supporting additional programming languages such as Python, Java, or JavaScript. By doing so, the framework can serve a wider range of learners and curricular needs. Similarly, the platform's collaborative features should be strengthened. Enhancing peer-to-peer learning options such as real-time coding, AI-supported group work, and shared feedback channels will improve interactivity and engagement.

Longitudinal studies should also be conducted to track learners over time, examining how the framework affects long-term academic performance, retention, and industry readiness. This will provide deeper insights into the sustained impact of GenAI-enhanced education. Finally, bias mitigation must become a core part of ongoing development. Researchers should explore the integration of algorithmic transparency tools and human-in-the-loop mechanisms to evaluate and minimize bias in AI-generated content. These steps are essential for creating fair, inclusive, and pedagogically sound learning environments.

REFERENCES

[1] J. Nouri, L. Zhang, L. Mannila, and E. Norén, "Development of computational thinking, digital competence and 21 st century skills when learning programming in K-9," *Educ. Inq.*, vol. 11, no. 1, pp. 1–17, Jan. 2020, doi: 10.1080/20004508.2019.1627844.

[2] L. Jia, R. Alizadeh, J. Hao, G. Wang, J. K. Allen, and F. Mistree, "A rule-based method for automated surrogate model selection," *Adv. Eng. Inform.*, vol. 45, p. 101123, Aug. 2020, doi: 10.1016/j.aei.2020.101123.

[3] W. S. Sayed *et al.*, "AI-based adaptive personalized content presentation and exercises navigation for an effective and engaging E-learning platform," *Multimed. Tools Appl.*, vol. 82, no. 3, pp. 3303–3333, Jan. 2023, doi: 10.1007/s11042-022-13076-8.

[4] S. Maghsudi, A. Lan, J. Xu, and M. Van Der Schaar, "Personalized Education in the Artificial Intelligence Era: What to Expect Next," *IEEE Signal Process. Mag.*, vol. 38, no. 3, pp. 37–50, May 2021, doi: 10.1109/MSP.2021.3055032.

[5] S. Sarsa, J. Pettersson, and A. Hellas, "How to Help to Ask for Help? Help Request Prompt Structure Influence on Help Request Quantity and Course Retention," in *2022 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden: IEEE, Oct. 2022, pp. 1–9. doi: 10.1109/FIE56618.2022.9962561.

[6] R. Kadar, N. Abdul Wahab, J. Othman, M. Shamsuddin, and S. B. Mahlan, "A Study of Difficulties in Teaching and Learning Programming: A Systematic Literature Review," *Int. J. Acad. Res. Progress. Educ. Dev.*, vol. 10, no. 3, p. Pages 591-605, Aug. 2021, doi: 10.6007/IJARPED/v10-i3/11100.

[7] C. S. Cheah, "Factors Contributing to the Difficulties in Teaching and Learning of Computer Programming: A Literature Review," *Contemp. Educ. Technol.*, vol. 12, no. 2, p. ep272, May 2020, doi: 10.30935/cedtech/8247.

[8] F. Naseer, M. U. Khalid, N. Ayub, A. Rasool, T. Abbas, and M. W. Afzal, "Automated Assessment and Feedback in Higher Education Using Generative AI:," in *Advances in Educational Technologies and Instructional Design*, R. C.

Sharma and A. Bozkurt, Eds., IGI Global, 2024, pp. 433–461. doi: 10.4018/979-8-3693-1351-0.ch021.

[9] Z. Xiong, Y. Cui, Z. Liu, Y. Zhao, M. Hu, and J. Hu, "Evaluating explorative prediction power of machine learning algorithms for materials discovery using k -fold forward cross-validation," *Comput. Mater. Sci.*, vol. 171, p. 109203, Jan. 2020, doi: 10.1016/j.commatsci.2019.109203.

[10] V. Berezovskiy, A. Gorodilova, E. Trofimova, and A. Ustyuzhanin, "Machine learning code snippets semantic classification," *PeerJ Comput. Sci.*, vol. 9, p. e1654, Nov. 2023, doi: 10.7717/peerj-cs.1654.

[11] T. H. M. Le, H. Chen, and M. A. Babar, "Deep Learning for Source Code Modeling and Generation: Models, Applications, and Challenges," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–38, May 2021, doi: 10.1145/3383458.

[12] E. Kasneci *et al.*, "ChatGPT for good? On opportunities and challenges of large language models for education," *Learn. Individ. Differ.*, vol. 103, p. 102274, Apr. 2023, doi: 10.1016/j.lindif.2023.102274.

[13] Y. Wang *et al.*, "Towards Natural Language-Based Visualization Authoring," *IEEE Trans. Vis. Comput. Graph.*, pp. 1–11, 2022, doi: 10.1109/TVCG.2022.3209357.

[14] B. Min *et al.*, "Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey," *ACM Comput. Surv.*, vol. 56, no. 2, pp. 1–40, Feb. 2024, doi: 10.1145/3605943.

[15] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models," in *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*, Lugano and Virtual Event Switzerland: ACM, Aug. 2022, pp. 27–43. doi: 10.1145/3501385.3543957.

[16] S. Lau and P. Guo, "From 'Ban It Till We Understand It' to 'Resistance is Futile': How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot," in *Proceedings of the 2023 ACM Conference on International*

Computing Education Research V.1, Chicago IL USA: ACM, Aug. 2023, pp. 106–121. doi: 10.1145/3568813.3600138.

[17] L. Yan *et al.*, "Practical and ethical challenges of large language models in education: A systematic scoping review," *Br. J. Educ. Technol.*, vol. 55, no. 1, pp. 90–112, Jan. 2024, doi: 10.1111/bjet.13370.

[18] R. Scherer, F. Siddiq, and B. Sánchez Viveros, "The cognitive benefits of learning computer programming: A meta-analysis of transfer effects.," *J. Educ. Psychol.*, vol. 111, no. 5, pp. 764–792, Jul. 2019, doi: 10.1037/edu0000314.

[19] S. Yang, H. Tian, L. Sun, and X. Yu, "From One-size-fits-all Teaching to Adaptive Learning: The Crisis and Solution of Education in The Era of AI," *J. Phys. Conf. Ser.*, vol. 1237, no. 4, p. 042039, Jun. 2019, doi: 10.1088/1742-6596/1237/4/042039.

[20] R. Weeda, S. Smetsers, and E. Barendsen, "Unraveling novices' code composition difficulties," *Comput. Sci. Educ.*, pp. 1–28, Jan. 2023, doi: 10.1080/08993408.2023.2169067.

[21] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, and J. Prather, "The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming," in *Proceedings of the 24th Australasian Computing Education Conference*, Virtual Event Australia: ACM, Feb. 2022, pp. 10–19. doi: 10.1145/3511861.3511863.

[22] L. Chen, P. Chen, and Z. Lin, "Artificial Intelligence in Education: A Review," *IEEE Access*, vol. 8, pp. 75264–75278, 2020, doi: 10.1109/ACCESS.2020.2988510.

[23] T. Alqahtani *et al.*, "The emergent role of artificial intelligence, natural learning processing, and large language models in higher education and research," *Res. Soc. Adm. Pharm.*, vol. 19, no. 8, pp. 1236–1242, Aug. 2023, doi: 10.1016/j.sapharm.2023.05.016.

[24] C.-H. Lai and C.-Y. Lin, "Analysis of Learning Behaviors and Outcomes for Students with Different Knowledge Levels: A Case Study of Intelligent Tutoring System for Coding and Learning (ITS-CAL)," *Appl. Sci.*, vol. 15, no. 4, p. 1922, Feb. 2025, doi: 10.3390/app15041922.

[25] E. Frankford, C. Sauerwein, P. Bassner, S. Krusche, and R. Breu, “AI-Tutoring in Software Engineering Education,” in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, Lisbon Portugal: ACM, Apr. 2024, pp. 309–319. doi: 10.1145/3639474.3640061.

[26] Z. Li *et al.*, “TutorLLM: Customizing Learning Recommendations with Knowledge Tracing and Retrieval-Augmented Generation,” 2025, *arXiv*. doi: 10.48550/ARXIV.2502.15709.

[27] Y. Chang *et al.*, “A Survey on Evaluation of Large Language Models,” *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, pp. 1–45, Jun. 2024, doi: 10.1145/3641289.

[28] T. R. McIntosh, T. Susnjak, T. Liu, P. Watters, and M. N. Halgamuge, “Inadequacies of Large Language Model Benchmarks in the Era of Generative Artificial Intelligence,” 2024, doi: 10.48550/ARXIV.2402.09880.

[29] K. R. Pradeep, A. S. Manish, A. S. Adithiyaa, N. Sahana, and S. T. Abhishek, “Personalized Adaptive Learning Platform Empowered by Artificial Intelligence,” in *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, Chikkaballapur, India: IEEE, Apr. 2024, pp. 1–8. doi: 10.1109/ICKECS61492.2024.10617075.

[30] J. P. Lantolf, M. E. Poehner, and M. Swain, Eds., *The Routledge handbook of sociocultural theory and second language development*. in Routledge handbooks in applied linguistics. New York London: Routledge, Taylor Francis Group, 2018.

[31] B. Eun, “The zone of proximal development as an overarching concept: A framework for synthesizing Vygotsky’s theories,” *Educ. Philos. Theory*, vol. 51, no. 1, pp. 18–30, Jan. 2019, doi: 10.1080/00131857.2017.1421941.

[32] K. F. Hew, “Promoting engagement in online courses: What strategies can we learn from three highly rated MOOCS,” *Br. J. Educ. Technol.*, vol. 47, no. 2, pp. 320–341, Mar. 2016, doi: 10.1111/bjet.12235.

[33] J. Ryoo and K. Winkelmann, Eds., *Innovative learning environments in STEM higher education: opportunities, challenges, and looking forward*. Cham: Springer, 2021.

[34] *SYSTEMS AND MODELS FOR DEVELOPING PROGRAMS FOR THE GIFTED AND TALENTED*. ABINGDON: ROUTLEDGE, 2023.

[35] Ö. Demir and S. S. Seferoglu, "The Effect of Determining Pair Programming Groups According to Various Individual Difference Variables on Group Compatibility, Flow, and Coding Performance," *J. Educ. Comput. Res.*, vol. 59, no. 1, pp. 41–70, Mar. 2021, doi: 10.1177/0735633120949787.

[36] A. S. Ojong, "Unraveling the Efficacy of Differentiated Instruction in Enhancing Second Language Acquisition: A Comprehensive Review and Future Directions," *Int. J. Linguist. Lit. Transl.*, vol. 6, no. 6, pp. 75–82, Jun. 2023, doi: 10.32996/ijllt.2023.6.6.8.

[37] M. Tang, D. Wang, and A. Guerrien, "A systematic review and meta-analysis on basic psychological need satisfaction, motivation, and well-being in later life: Contributions of self-determination theory," *PsyCh J.*, vol. 9, no. 1, pp. 5–33, Feb. 2020, doi: 10.1002/pchj.293.

[38] S. J. Shelton-Strong, "Advising in language learning and the support of learners' basic psychological needs: A self-determination theory perspective," *Lang. Teach. Res.*, vol. 26, no. 5, pp. 963–985, Sep. 2022, doi: 10.1177/1362168820912355.

[39] I. M. Darwati and I. M. Purana, "Problem Based Learning (PBL) : Suatu Model Pembelajaran Untuk Mengembangkan Cara Berpikir Kritis Peserta Didik," *Widya Accarya*, vol. 12, no. 1, pp. 61–69, Apr. 2021, doi: 10.46650/wa.12.1.1056.61-69.

[40] Y. Wang, H. Le, A. D. Gotmare, N. D. Q. Bui, J. Li, and S. C. H. Hoi, "CodeT5+: Open Code Large Language Models for Code Understanding and Generation," 2023, doi: 10.48550/ARXIV.2305.07922.

[41] C. Jeong, "A Study on the Implementation of Generative AI Services Using an Enterprise Data-Based LLM Application Architecture," 2023, doi: 10.48550/ARXIV.2309.01105.

[42] S. MacNeil *et al.*, “Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book,” in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, Toronto ON Canada: ACM, Mar. 2023, pp. 931–937. doi: 10.1145/3545945.3569785.

[43] W. Wang, Y. Rao, R. Zhi, S. Marwan, G. Gao, and T. W. Price, “Step Tutor: Supporting Students through Step-by-Step Example-Based Feedback,” in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, Trondheim Norway: ACM, Jun. 2020, pp. 391–397. doi: 10.1145/3341525.3387411.

[44] J. Paladines and J. Ramirez, “A Systematic Literature Review of Intelligent Tutoring Systems With Dialogue in Natural Language,” *IEEE Access*, vol. 8, pp. 164246–164267, 2020, doi: 10.1109/ACCESS.2020.3021383.

[45] J. Chen, H. Lin, X. Han, and L. Sun, “Benchmarking Large Language Models in Retrieval-Augmented Generation,” *Proc. AAAI Conf. Artif. Intell.*, vol. 38, no. 16, pp. 17754–17762, Mar. 2024, doi: 10.1609/aaai.v38i16.29728.

[46] C. Piech *et al.*, “Deep Knowledge Tracing,” 2015, *arXiv*. doi: 10.48550/ARXIV.1506.05908.

[47] J. Prather *et al.*, “‘It’s Weird That it Knows What I Want’: Usability and Interactions with Copilot for Novice Programmers,” 2023, doi: 10.48550/ARXIV.2304.02491.

[48] F. Nizamudeen, L. Gatti, N. Bouali, and F. Ahmed, “Investigating the Impact of Code Generation Tools (ChatGPT & Github CoPilot) on Programming Education:,” in *Proceedings of the 16th International Conference on Computer Supported Education*, Angers, France: SCITEPRESS - Science and Technology Publications, 2024, pp. 221–229. doi: 10.5220/0012628000003693.

[49] W. Takerngsaksiri, C. Warusavitarne, C. Yaacoub, M. H. K. Hou, and C. Tantithamthavorn, “Students’ Perspective on AI Code Completion: Benefits and Challenges,” 2023, *arXiv*. doi: 10.48550/ARXIV.2311.00177.

[50] S. Maity and A. Deroy, “Generative AI and Its Impact on Personalized Intelligent Tutoring Systems,” 2024, *arXiv*. doi: 10.48550/ARXIV.2410.10650.

[51] W. Cain, “Prompting Change: Exploring Prompt Engineering in Large Language Model AI and Its Potential to Transform Education,” *TechTrends*, vol. 68, no. 1, pp. 47–57, Jan. 2024, doi: 10.1007/s11528-023-00896-0.

[52] S. Sivarajkumar, M. Kelley, A. Samolyk-Mazzanti, S. Visweswaran, and Y. Wang, “An Empirical Evaluation of Prompting Strategies for Large Language Models in Zero-Shot Clinical Natural Language Processing: Algorithm Development and Validation Study,” *JMIR Med. Inform.*, vol. 12, p. e55318, Apr. 2024, doi: 10.2196/55318.

[53] L. Reynolds and K. McDonell, “Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm,” in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, Yokohama Japan: ACM, May 2021, pp. 1–7. doi: 10.1145/3411763.3451760.

[54] B. Hof, “The turtle and the mouse: how constructivist learning theory shaped artificial intelligence and educational technology in the 1960s,” *Hist. Educ.*, vol. 50, no. 1, pp. 93–111, Jan. 2021, doi: 10.1080/0046760X.2020.1826053.

[55] M. Schmid, E. Brianza, and D. Petko, “Developing a short assessment instrument for Technological Pedagogical Content Knowledge (TPACK.xs) and comparing the factor structure of an integrative and a transformative model,” *Comput. Educ.*, vol. 157, p. 103967, Nov. 2020, doi: 10.1016/j.compedu.2020.103967.

[56] K. Cunningham, Y. Qiao, A. Feng, and E. O’Rourke, “Bringing ‘High-level’ Down to Earth: Gaining Clarity in Conversational Programmer Learning Goals,” in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*, Providence RI USA: ACM, Feb. 2022, pp. 551–557. doi: 10.1145/3478431.3499370.

[57] S. Marwan, N. Lytle, J. J. Williams, and T. Price, “The Impact of Adding Textual Explanations to Next-step Hints in a Novice Programming Environment,” in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, Aberdeen Scotland Uk: ACM, Jul. 2019, pp. 520–526. doi: 10.1145/3304221.3319759.

[58] M. Hafeez, A. Karki, Y. Radwan, A. Saha, and A. Zavaleta Bernuy, “Evaluating the Efficacy and Impacts of Remote Pair Programming for Introductory

Computer Science Students," in *Proceedings of the 25th Western Canadian Conference on Computing Education*, Vancouver BC Canada: ACM, May 2023, pp. 1–7. doi: 10.1145/3593342.3593351.

[59] J. M. Griffin, "Learning by Taking Apart: Deconstructing Code by Reading, Tracing, and Debugging," in *Proceedings of the 17th Annual Conference on Information Technology Education*, Boston Massachusetts USA: ACM, Sep. 2016, pp. 148–153. doi: 10.1145/2978192.2978231.

[60] Z. Ullah, A. Lajis, M. Jamjoom, A. Altalhi, A. Al-Ghamdi, and F. Saleem, "The effect of automatic assessment on novice programming: Strengths and limitations of existing systems," *Comput. Appl. Eng. Educ.*, vol. 26, no. 6, pp. 2328–2341, Nov. 2018, doi: 10.1002/cae.21974.

[61] G.-G. Lee *et al.*, "Multimodality of AI for Education: Towards Artificial General Intelligence," 2023, doi: 10.48550/ARXIV.2312.06037.

[62] S. Wu *et al.*, "A Comprehensive Exploration of Personalized Learning in Smart Education: From Student Modeling to Personalized Recommendations," 2024, doi: 10.48550/ARXIV.2402.01666.

[63] A. Emerson *et al.*, "Cluster-Based Analysis of Novice Coding Misconceptions in Block-Based Programming," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, Portland OR USA: ACM, Feb. 2020, pp. 825–831. doi: 10.1145/3328778.3366924.

[64] S. Liu, X. Guo, X. Hu, and X. Zhao, "Advancing Generative Intelligent Tutoring Systems with GPT-4: Design, Evaluation, and a Modular Framework for Future Learning Platforms," *Electronics*, vol. 13, no. 24, p. 4876, Dec. 2024, doi: 10.3390/electronics13244876.

[65] G. Jošt, V. Taneski, and S. Karakatič, "The Impact of Large Language Models on Programming Education and Student Learning Outcomes," *Appl. Sci.*, vol. 14, no. 10, p. 4115, May 2024, doi: 10.3390/app14104115.

[66] K. K. Maurya, K. A. Srivatsa, K. Petukhova, and E. Kochmar, "Unifying AI Tutor Evaluation: An Evaluation Taxonomy for Pedagogical Ability Assessment of LLM-Powered AI Tutors," 2024, doi: 10.48550/ARXIV.2412.09416.

[67] W.-Y. Chen, “Intelligent Tutor: Leveraging ChatGPT and Microsoft Copilot Studio to Deliver a Generative AI Student Support and Feedback System within Teams,” 2024, *arXiv*. doi: 10.48550/ARXIV.2405.13024.

[68] S. Wang *et al.*, “Large Language Models for Education: A Survey and Outlook,” 2024, *arXiv*. doi: 10.48550/ARXIV.2403.18105.

[69] J. Figueiredo and F. J. García-Peñalvo, “Intelligent Tutoring Systems approach to Introductory Programming Courses,” in *Eighth International Conference on Technological Ecosystems for Enhancing Multiculturality*, Salamanca Spain: ACM, Oct. 2020, pp. 34–39. doi: 10.1145/3434780.3436614.

[70] B. (Zack) Batsaikhan and A.-P. Correia, “The effects of Generative Artificial Intelligence on Intelligent Tutoring Systems in higher education: A systematic review,” *Stud. Technol. Enhanc. Learn.*, vol. 4, no. 1, Oct. 2024, doi: 10.21428/8c225f6e.33570bb1.

[71] P. J. Phillips *et al.*, “Four principles of explainable artificial intelligence,” National Institute of Standards and Technology (U.S.), Gaithersburg, MD, NIST IR 8312, Sep. 2021. doi: 10.6028/NIST.IR.8312.

[72] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜,” in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, Virtual Event Canada: ACM, Mar. 2021, pp. 610–623. doi: 10.1145/3442188.3445922.

[73] Á. Becerra, Z. Mohseni, J. Sanz, and R. Cobos, “A Generative AI-Based Personalized Guidance Tool for Enhancing the Feedback to MOOC Learners,” in *2024 IEEE Global Engineering Education Conference (EDUCON)*, Kos Island, Greece: IEEE, May 2024, pp. 1–8. doi: 10.1109/EDUCON60312.2024.10578809.

[74] S. C. Tan, W. Chen, and B. L. Chua, “Leveraging generative artificial intelligence based on large language models for collaborative learning,” *Learn. Res. Pract.*, vol. 9, no. 2, pp. 125–134, Jul. 2023, doi: 10.1080/23735082.2023.2258895.

[75] S. Shamroukh and T. Johnson, “Using Factor Analysis to Determine the Factors Impacting Learning Python for Non-Technical Business Analytics Graduate Students,” *J. Data Anal. Inf. Process.*, vol. 11, no. 04, pp. 512–535, 2023, doi: 10.4236/jdaip.2023.114026.

[76] J.-C. Liang, G.-J. Hwang, M.-R. A. Chen, and D. Darmawansah, “Roles and research foci of artificial intelligence in language education: an integrated bibliographic analysis and systematic review approach,” *Interact. Learn. Environ.*, vol. 31, no. 7, pp. 4270–4296, Oct. 2023, doi: 10.1080/10494820.2021.1958348.

[77] A. Tack and C. Piech, “The AI Teacher Test: Measuring the Pedagogical Ability of Blender and GPT-3 in Educational Dialogues,” 2022, doi: 10.48550/ARXIV.2205.07540.

[78] H. Liu, Z. Wu, Y. Lu, and L. Zhu, “Exploring the Balance Between Computational Thinking and Learning Motivation in Elementary Programming Education: An Empirical Study With Game-Based Learning,” *IEEE Trans. Games*, vol. 15, no. 1, pp. 95–107, Mar. 2023, doi: 10.1109/TG.2022.3143701.

[79] J. Fagerlund, P. Häkkinen, M. Vesisenaho, and J. Viiri, “Computational thinking in programming with Scratch in primary schools: A systematic review,” *Comput. Appl. Eng. Educ.*, vol. 29, no. 1, pp. 12–28, Jan. 2021, doi: 10.1002/cae.22255.

[80] C.-Y. Tsai, “Improving students’ understanding of basic programming concepts through visual programming language: The role of self-efficacy,” *Comput. Hum. Behav.*, vol. 95, pp. 224–232, Jun. 2019, doi: 10.1016/j.chb.2018.11.038.

[81] C. Tikva and E. Tambouris, “Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature Review,” *Comput. Educ.*, vol. 162, p. 104083, Mar. 2021, doi: 10.1016/j.compedu.2020.104083.

[82] R. Mathew, S. I. Malik, and R. M. Tawafak, “Teaching Problem Solving Skills using an Educational Game in a Computer Programming Course,” *Inform. Educ.*, vol. 18, no. 2, pp. 359–373, Oct. 2019, doi: 10.15388/infedu.2019.17.

[83] P. Vaithilingam, T. Zhang, and E. L. Glassman, “Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language

Models," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, New Orleans LA USA: ACM, Apr. 2022, pp. 1–7. doi: 10.1145/3491101.3519665.

[84] C. K. Lo, "What Is the Impact of ChatGPT on Education? A Rapid Review of the Literature," *Educ. Sci.*, vol. 13, no. 4, p. 410, Apr. 2023, doi: 10.3390/educsci13040410.

[85] P. Denny, V. Kumar, and N. Giacaman, "Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, Toronto ON Canada: ACM, Mar. 2023, pp. 1136–1142. doi: 10.1145/3545945.3569823.

[86] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, "Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, Hamburg Germany: ACM, Apr. 2023, pp. 1–23. doi: 10.1145/3544548.3580919.

[87] Z. A. Pardos and S. Bhandari, "ChatGPT-generated help produces learning gains equivalent to human tutor-authored help on mathematics skills," *PLOS ONE*, vol. 19, no. 5, p. e0304013, May 2024, doi: 10.1371/journal.pone.0304013.

[88] C. Nelson, A. Doupé, and Y. Shoshitaishvili, "SENSAI: Large Language Models as Applied Cybersecurity Tutors," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, Pittsburgh PA USA: ACM, Feb. 2025, pp. 833–839. doi: 10.1145/3641554.3701801.

[89] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review," Sep. 05, 2024, *arXiv*: arXiv:2310.14735. doi: 10.48550/arXiv.2310.14735.

[90] J. Chen *et al.*, "When large language models meet personalization: perspectives of challenges and opportunities," *World Wide Web*, vol. 27, no. 4, p. 42, Jul. 2024, doi: 10.1007/s11280-024-01276-1.

[91] J. Doughty *et al.*, "A Comparative Study of AI-Generated (GPT-4) and Human-crafted MCQs in Programming Education," in *Proceedings of the 26th*

Australasian Computing Education Conference, Sydney NSW Australia: ACM, Jan. 2024, pp. 114–123. doi: 10.1145/3636243.3636256.

[92] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. (Jack) Jiang, "GitHub Copilot AI pair programmer: Asset or Liability?," *J. Syst. Softw.*, vol. 203, p. 111734, Sep. 2023, doi: 10.1016/j.jss.2023.111734.

[93] S. Abukhalaf, M. Hamdaqa, and F. Khomh, "On Codex Prompt Engineering for OCL Generation: An Empirical Study," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, Melbourne, Australia: IEEE, May 2023, pp. 148–157. doi: 10.1109/MSR59073.2023.00033.

[94] X. V. Lin *et al.*, "Few-shot Learning with Multilingual Language Models," 2021, *arXiv*. doi: 10.48550/ARXIV.2112.10668.

[95] R. Makharia *et al.*, "AI Tutor Enhanced with Prompt Engineering and Deep Knowledge Tracing," in *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India: IEEE, Mar. 2024, pp. 1–6. doi: 10.1109/IATMSI60426.2024.10503187.

[96] J. Zhang, P. Nie, J. J. Li, and M. Gligoric, "Multilingual Code Co-evolution using Large Language Models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, San Francisco CA USA: ACM, Nov. 2023, pp. 695–707. doi: 10.1145/3611643.3616350.

[97] S. Sharma, M. Raj, M. M. Ali, K. Sharma, A. Singh, and S. Singh, "Design of Structure using an Adaptive Generative AI System for Customised Eduction for Individuals," in *2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*, Greater Noida, India: IEEE, Feb. 2024, pp. 276–280. doi: 10.1109/IC2PCT60090.2024.10486396.

[98] S. Gupta, R. R. Dharamshi, and V. Kakde, "An Impactful and Revolutionized Educational Ecosystem using Generative AI to Assist and Assess the Teaching and Learning benefits, Fostering the Post-Pandemic Requirements," in *2024 Second International Conference on Emerging Trends in Information*

Technology and Engineering (ICETITE), Vellore, India: IEEE, Feb. 2024, pp. 1–4. doi: 10.1109/ic-ETITE58242.2024.10493370.

[99] J. Ahn, W. Sung, and J. B. Black, "Unplugged Debugging Activities for Developing Young Learners' Debugging Skills," *J. Res. Child. Educ.*, vol. 36, no. 3, pp. 421–437, May 2022, doi: 10.1080/02568543.2021.1981503.

[100] J. Li, P. Chen, and J. Jia, "MoTCoder: Elevating Large Language Models with Modular of Thought for Challenging Programming Tasks," 2023, doi: 10.48550/ARXIV.2312.15960.

[101] K. K. Phokela, S. Sikand, K. Singi, K. Dey, V. S. Sharma, and V. Kaulgud, "Smart Prompt Advisor: Multi-Objective Prompt Framework for Consistency and Best Practices," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Luxembourg, Luxembourg: IEEE, Sep. 2023, pp. 1846–1848. doi: 10.1109/ASE56229.2023.00019.

[102] A. Narasimhan, K. P. A. V. Rao, and V. M. B, "CGEMs: A Metric Model for Automatic Code Generation using GPT-3," 2021, *arXiv*. doi: 10.48550/ARXIV.2108.10168.

[103] J. White *et al.*, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," 2023, doi: 10.48550/ARXIV.2302.11382.

[104] D. Zhou *et al.*, "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models," 2022, *arXiv*. doi: 10.48550/ARXIV.2205.10625.

[105] J. Wei *et al.*, "Finetuned Language Models Are Zero-Shot Learners," 2021, *arXiv*. doi: 10.48550/ARXIV.2109.01652.

[106] A. Narayan, I. Chami, L. Orr, S. Arora, and C. Ré, "Can Foundation Models Wrangle Your Data?," 2022, *arXiv*. doi: 10.48550/ARXIV.2205.09911.

[107] A. Del Carpio Gutierrez, P. Denny, and A. Luxton-Reilly, "Evaluating Automatically Generated Contextualised Programming Exercises," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, Portland OR USA: ACM, Mar. 2024, pp. 289–295. doi: 10.1145/3626252.3630863.

[108] M. Bahrami, M. Mansoorizadeh, and H. Khotanlou, “Few-shot Learning with Prompting Methods,” in *2023 6th International Conference on Pattern Recognition and Image Analysis (IPRIA)*, Qom, Iran, Islamic Republic of: IEEE, Feb. 2023, pp. 1–5. doi: 10.1109/IPRIA59240.2023.10147172.

[109] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, “A systematic evaluation of large language models of code,” in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, San Diego CA USA: ACM, Jun. 2022, pp. 1–10. doi: 10.1145/3520312.3534862.

[110] L. Sun and Z. Shi, “Prompt Learning Under the Large Language Model,” in *2023 International Seminar on Computer Science and Engineering Technology (SCSET)*, New York, NY, USA: IEEE, Apr. 2023, pp. 288–291. doi: 10.1109/SCSET58950.2023.00070.

[111] I. Gligorea, M. Cioca, R. Oancea, A.-T. Gorski, H. Gorski, and P. Tudorache, “Adaptive Learning Using Artificial Intelligence in e-Learning: A Literature Review,” *Educ. Sci.*, vol. 13, no. 12, p. 1216, Dec. 2023, doi: 10.3390/educsci13121216.

[112] H. Tanaka, N. Mori, and M. Okada, “Genetic Algorithm for Prompt Engineering with Novel Genetic Operators,” in *2023 15th International Congress on Advanced Applied Informatics Winter (IIAI-AAI-Winter)*, Bali, Indonesia: IEEE, Dec. 2023, pp. 209–214. doi: 10.1109/IIAI-AAI-Winter61682.2023.00047.

[113] Q. Zheng *et al.*, “CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Benchmarking on HumanEval-X,” 2023, *arXiv*. doi: 10.48550/ARXIV.2303.17568.

[114] P. Lauren and P. Watta, “Work-in-Progress: Integrating Generative AI with Evidence-based Learning Strategies in Computer Science and Engineering Education,” in *2023 IEEE Frontiers in Education Conference (FIE)*, College Station, TX, USA: IEEE, Oct. 2023, pp. 1–5. doi: 10.1109/FIE58773.2023.10342970.

[115] M. Ciniselli, N. Cooper, L. Pascarella, D. Poshyvanyk, M. Di Penta, and G. Bavota, “An Empirical Study on the Usage of BERT Models for Code

Completion," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, Madrid, Spain: IEEE, May 2021, pp. 108–119. doi: 10.1109/MSR52588.2021.00024.

[116] N. Kiesler and D. Schiffner, "Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments," 2023, *arXiv*. doi: 10.48550/ARXIV.2308.08572.

[117] Y. Sasaki, H. Washizaki, J. Li, D. Sander, N. Yoshioka, and Y. Fukazawa, "Systematic Literature Review of Prompt Engineering Patterns in Software Engineering," in *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, Osaka, Japan: IEEE, Jul. 2024, pp. 670–675. doi: 10.1109/COMPSAC61105.2024.00096.

[118] H. Strobelt *et al.*, "Interactive and Visual Prompt Engineering for Ad-hoc Task Adaptation With Large Language Models," *IEEE Trans. Vis. Comput. Graph.*, pp. 1–11, 2022, doi: 10.1109/TVCG.2022.3209479.

[119] X.-Y. Li, J.-T. Xue, Z. Xie, and M. Li, "Think Outside the Code: Brainstorming Boosts Large Language Models in Code Generation," 2023, *arXiv*. doi: 10.48550/ARXIV.2305.10679.

[120] R. Banjade, P. Oli, M. I. Sajib, and V. Rus, "Identifying Gaps in Students' Explanations of Code Using LLMs," in *Artificial Intelligence in Education*, vol. 14830, A. M. Olney, I.-A. Chounta, Z. Liu, O. C. Santos, and I. I. Bittencourt, Eds., in Lecture Notes in Computer Science, vol. 14830. , Cham: Springer Nature Switzerland, 2024, pp. 268–275. doi: 10.1007/978-3-031-64299-9_21.

[121] K. Fahd, S. J. Miah, K. Ahmed, S. Venkatraman, and Y. Miao, "Integrating design science research and design based research frameworks for developing education support systems," *Educ. Inf. Technol.*, vol. 26, no. 4, pp. 4027–4048, Jul. 2021, doi: 10.1007/s10639-021-10442-1.

[122] M. Hirose and J. W. Creswell, "Applying Core Quality Criteria of Mixed Methods Research to an Empirical Study," *J. Mix. Methods Res.*, vol. 17, no. 1, pp. 12–28, Jan. 2023, doi: 10.1177/15586898221086346.

[123] "Qualitative Research Designs, Sample Size and Saturation: Is Enough Always Enough?," *J. Advocacy Res. Educ.*, vol. 8, no. 3, Dec. 2021, doi: 10.13187/jare.2021.3.60.

[124] H. A. Mohamed Shaffril, S. F. Samsuddin, and A. Abu Samah, "The ABC of systematic literature review: the basic methodological guidance for beginners," *Qual. Quant.*, vol. 55, no. 4, pp. 1319–1346, Aug. 2021, doi: 10.1007/s11135-020-01059-6.

[125] U. Mittal, S. Sai, V. Chamola, and D. Sangwan, "A Comprehensive Review on Generative AI for Education," *IEEE Access*, vol. 12, pp. 142733–142759, 2024, doi: 10.1109/ACCESS.2024.3468368.

[126] K. Jafarzade, "The Role of GPT Models in Education: Challenges and Solutions," in *2023 5th International Conference on Problems of Cybernetics and Informatics (PCI)*, Baku, Azerbaijan: IEEE, Aug. 2023, pp. 1–3. doi: 10.1109/PCI60110.2023.10325940.

[127] E. Aftabi, B. N. Shirazi, A. A. Safavi, G. Salimi, and H. Aftabi, "A Framework for Customized Course Design and Personalized Learning with AI," in *2024 11th International and the 17th National Conference on E-Learning and E-Teaching (ICeLeT)*, Isfahan, Iran, Islamic Republic of: IEEE, Feb. 2024, pp. 1–6. doi: 10.1109/ICeLeT62507.2024.10493063.

[128] W. Lyu, Y. Wang, T. (Rachel) Chung, Y. Sun, and Y. Zhang, "Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study," in *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, Atlanta GA USA: ACM, Jul. 2024, pp. 63–74. doi: 10.1145/3657604.3662036.

[129] J. L. Redifer, C. L. Bae, and Q. Zhao, "Self-efficacy and performance feedback: Impacts on cognitive load during creative thinking," *Learn. Instr.*, vol. 71, p. 101395, Feb. 2021, doi: 10.1016/j.learninstruc.2020.101395.

[130] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code," 2021, *arXiv*. doi: 10.48550/ARXIV.2107.03374.

[131] T. Wang and Z. Chen, “Analyzing Code Text Strings for Code Evaluation,” in *2023 IEEE International Conference on Big Data (BigData)*, Sorrento, Italy: IEEE, Dec. 2023, pp. 5619–5628. doi: 10.1109/BigData59044.2023.10386406.

[132] E. Ferrara, “Fairness and Bias in Artificial Intelligence: A Brief Survey of Sources, Impacts, and Mitigation Strategies,” *Sci*, vol. 6, no. 1, p. 3, Dec. 2023, doi: 10.3390/sci6010003.

APPENDICES

Appendix 1: Expert Opinion Questionnaire

Informed Consent Form for Participants

I am Fredrick Muema Mboya, registration number SC401/5825/2022, pursuing Master of Science in Information Technology at the School of Computing and Information Technology, Department of Information Technology of Murang'a University of Technology. The research topic is "A Large Language Model Based Framework for Enhancing Personalized Learning of C Programming".

You are invited to participate in an expert opinion survey aimed at validating the instructional features implemented in a Large Language Model (LLM)-based framework developed to enhance personalized programming education. Your insights will help improve the integration of LLMs in computer programming instruction.

Participation is voluntary, and all responses will remain anonymous and confidential. The data collected will be used solely for academic research purposes.

Please check the consent statements below before proceeding with the questionnaire.

SN	STATEMENT	I GIVE MY CONSENT	I DO NOT GIVE MY CONSENT
1.	I have read and understood the purpose of this project.		
2.	I have been given the opportunity to ask question about my participation and project.		
3.	I voluntarily agree to participate in project.		
4.	The procedure regarding confidentiality and anonymity are clearly explained.		
5.	The use of obtained data in research, publication, sharing and archiving has been clearly explained to me.		
6.	I understand that other researchers can access obtained data if they agree to preserve confidentiality.		
7.	For purpose of project recognition, I understand the data obtained will be used in reports, publications and other research outputs.		
8.	I agree together with the researcher to sign and date this informed consent form.		

Please indicate your consent to participate by signing below.

Signature of Respondent: _____ Date: _____

SECTION A:

Demographic Information

1. Gender [ ] Male [ ] Female

2. Programme of study

[ ] Bsc. Computer Science

[ ] Bsc. Computer Technology

[ ] Bsc. Information Technology

[ ] Bsc. Software Engineering

[ ] Bachelor of Business Information Technology

3. Year of Study

[ ] First year

[ ] Second year

[ ] Third year

[ ] Fourth year

[ ] Recently graduated

4. Have you taken C Programming as part of your academic curriculum?

[] Yes

[] No

If yes, what grade did you score? _____

SECTION B:

Rate use of Generative AI in teaching computer programming guided by questions items below:

On a five-point Likert scale indicate your agreement with the statements given and explain your selection as guided under explanations

1 - Not at All

2 - Very Rarely

3 - To some little Extent

4 - To some Moderate Extent

5 - To some Great Extent

Theme	Question	1	2	3	4	5
Theme 1: Clarity and Coherence	Code explanations are clear and logically structured.					
	All major code components are covered accurately.					
	Step-by-step logic is maintained in walkthroughs.					
	Programming terminology is consistently applied.					
	Explanations address possible code limitations or edge cases.					
Theme 2: Accuracy and Relevance	Code outputs match explanations provided.					
	Generated responses reflect actual language behaviour.					
	Algorithms are explained appropriately, where applicable.					
	Examples are valid and contextually relevant.					
	No major factual or logical inaccuracies are present.					
Theme 3: Interactive Feedback and Collaboration	Learners are guided with actionable real-time feedback.					
	Prompts encourage peer reflection and discussion.					
	Exercises can be adapted for teamwork or pair programming.					
	Students are guided to improve solutions iteratively.					
	Feedback mechanisms reinforce learning effectively.					
	Exercises align with practical programming scenarios.					

Theme 4: Practicality and Usability	Tasks are innovative, not repetitive or template-based. Challenges are suitable for building programming skills. Tasks reflect real-world application of programming knowledge. Prompts support creativity or open-ended problem solving.
Theme 5: Instructional Readiness	Generated tasks are suitable for direct classroom use. Content aligns with beginner-level course objectives. Tasks and explanations require minimal revision. Code samples are easy to understand without extensive edits. Exercises are instructional and outcome-focused.

Appendix 2: Framework Evaluation Questionnaire

SURVEY ON GENAI-BASED FRAMEWORK FOR PROGRAMMING LEARNING

This instrument is designed to collect student feedback on the personalized learning framework powered by Generative AI. Your honest responses will help improve the tool and its effectiveness in supporting programming education.

Instructions:

Please indicate your level of agreement with each statement using the scale below:
1 – Strongly Disagree | 2 – Disagree | 3 – Neutral | 4 – Agree | 5 – Strongly Agree

SECTION A: System Usability and Interaction

No.	Statement	1	2	3	4	5
1	The interface was easy to use and navigate.					
2	The system provided helpful instructions during use.					
3	I was able to complete tasks without unnecessary effort.					
4	The feedback provided was understandable.					
5	I felt confident using the system after the initial guidance.					

SECTION B: Personalization and Feedback

No.	Statement	1	2	3	4	5
6	The tool provided feedback based on my specific responses or mistakes.					
7	The explanations helped me understand C programming concepts better.					
8	The system adjusted the difficulty of questions or tasks according to my progress.					
9	I received step-by-step guidance that supported my learning process.					
10	The tool helped me focus more on areas I needed improvement in.					

SECTION C: Engagement and Learning Outcomes

No.	Statement	1	2	3	4	5
11	Using the tool increased my interest in learning programming.					
12	The tool helped me retain concepts better than traditional lectures.					
13	I felt more confident in solving programming problems after using the tool.					
14	The system helped me identify and correct my programming errors.					
15	I would recommend the tool to other students learning programming.					

SECTION D: Open-Ended Feedback

1. What did you like most about the GenAI-powered learning tool? .....
2. What difficulties or challenges did you face while using the tool? .....
3. In what ways do you feel the tool supported your learning experience? .....
4. What improvements would you suggest for future use of the tool? .....

Appendix 3: Final SLR Selected Articles

S/No	Citation	Year	Title (Shortened)	Methodology	Key Findings	Theme(s)
1	[125]	2024	Review on GenAI for Education	Systematic Review	AI enhances personalization and engagement	Adaptability, Contextual Relevance
2	[29]	2024	Adaptive Learning Platform	Empirical Study	Adaptive tools improve learner intervention	Adaptability
3	[95]	2024	AI Tutor with Prompt Engineering	Framework Development	Enables dynamic feedback	Adaptability, Prompt Engineering
4	[73]	2024	Personalized Guidance Tool	Case Study	Real-time feedback enhances engagement	Adaptability, Contextual Relevance
5	[113]	2023	CodeGeeX Framework	Model Evaluation	Multilingual support in programming tasks	Adaptability, Practical Usability
6	[15]	2022	Auto-Generated Programming Exercises	Empirical Study	Reduces workload and supports personalization	Practical Usability, Creativity/Uniqueness
7	[74]	2023	LEVER	Framework Development	Improves code reliability	Accuracy and Reliability
8	[126]	2023	GPT Models in Education	Review	Highlights integration challenges	Adaptability, Prompt Engineering
9	[96]	2023	Multilingual Code Co-evolution	Empirical Study	Supports diverse learning contexts	Adaptability, Contextual Relevance
10	[101]	2023	Smart Prompt Advisor	Model Development	Improves prompt effectiveness	Prompt Engineering,

S/No	Citation	Year	Title (Shortened)	Methodology	Key Findings	Theme(s)
						Clarity/Coherence
11	[6]	2021	Learning Programming Difficulties	Systematic Review	Identifies barriers and solutions	Contextual Relevance, Interactivity
12	[107]	2024	AI-Generated Feedback	Benchmark Analysis	Reliable AI feedback	Accuracy and Reliability, Usability
13	[119]	2023	Brainstorming Boosts in LLMs	Experimental Study	Enhances solution diversity	Creativity and Uniqueness, Prompt Engineering
14	[102]	2021	Code Generation Metrics	Metric Development	Establishes coherence standards	Clarity and Coherence
15	[118]	2022	Visual Prompt Engineering	Framework Development	Supports prompt evaluation	Prompt Engineering, Interactivity
16	[17]	2024	Ethical GenAI Use	Review	Highlights fairness and bias concerns	Accuracy and Reliability
17	[127]	2024	Adaptive Code Debugging	Framework Development	Enhances debugging and reliability	Adaptability, Accuracy
18	[112]	2023	Prompt Eng. for Advanced Education	Review	Provides expert-level prompting strategies	Prompt Engineering
19	[8]	2024	Real-Time Feedback in Online Learning	Case Study	Improves engagement and outcomes	Adaptability, Interactivity
20	[108]	2023	Few-shot Prompt Engineering	Experimental Study	Boosts learner outcomes	Prompt Engineering, Accuracy

S/No	Citation	Year	Title (Shortened)	Methodology	Key Findings	Theme(s)
21	[128]	2024	AI Tutors in Programing	Empirical Study	Enhances learner performance	Adaptability
22	[61]	2023	Multimodal Frameworks	Framework Development	Combines visual and textual instruction	Practical Usability, Clarity
23	[98]	2024	AI for STEM Education	Case Study	Customizes content for learners	Contextual Relevance, Practical Usability
24	[114]	2023	AI in Curricula	Framework Development	Embeds AI into instructional designs	Practical Usability, Interactivity
25	[91]	2024	AI Code Model Comparison	Benchmark Analysis	Evaluates output accuracy	Accuracy and Reliability, Creativity
26	[51]	2024	Engagement with GenAI	Empirical Study	Increases student motivation	Prompt Engineering, Interactivity
27	[97]	2024	Adaptive AI Tutors	Case Study	Personalizes instruction	Adaptability, Usability
28	[115]	2021	Code Completion Tools	Model Evaluation	Useful for beginner learners	Readily Usable Content, Accuracy
29	[62]	2024	Scaling Personalized Education	Framework Development	Supports large-scale delivery	Adaptability, Practical Usability
30	[99]	2022	Debugging Assistants	Case Study	Offers real-time help	Contextual Relevance, Accuracy
31	[105]	2021	Fine-Tuning LLMs for Education	Experimental Study	Refines alignment with goals	Prompt Eng., Practical Usability

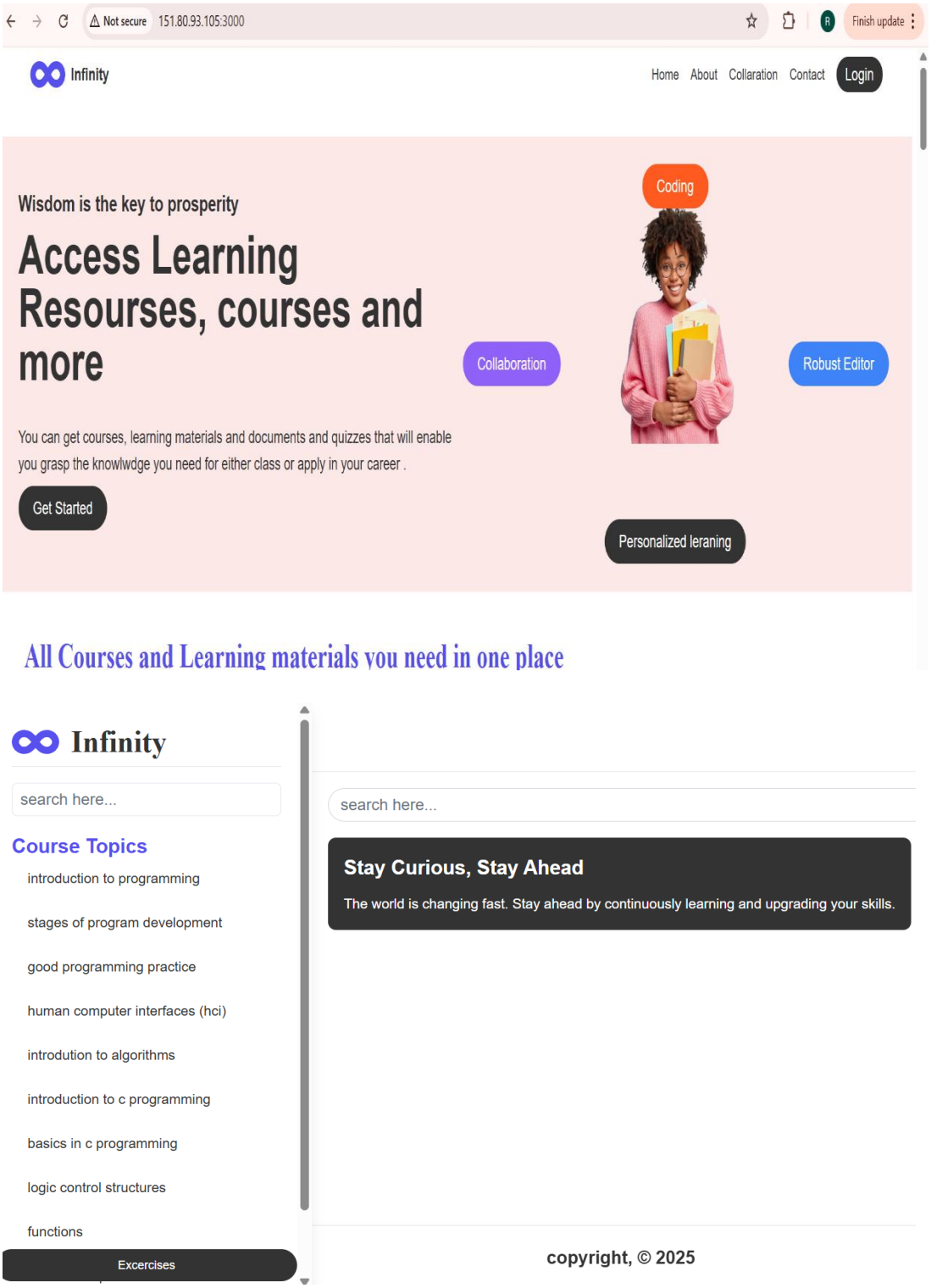
S/No	Citation	Year	Title (Shortened)	Methodology	Key Findings	Theme(s)
32	[129]	2021	Feedback for Online Programming	Model Evaluation	Real-time learner support	Interactivity, Readily Usable
33	[109]	2022	Large-Scale AI Frameworks	Framework Development	Enables institution-wide use	Practical Usability, Context Relevance
34	[130]	2021	LLMs on Code	Benchmark Analysis	Assesses automated code support	Accuracy and Reliability
35	[104]	2022	Chain-of-Thought Prompting	Review	Enhances logic flow	Clarity, Prompt Engineering
36	[116]	2023	AI Scaffolding Tools	Framework Development	Aids abstract programming learning	Adaptability, Interactivity
37	[103]	2021	Prompt Pattern Catalog	Model Development	Improves prompt structure	Prompt Engineering
38	[131]	2023	AI-Based Assessment Tools	Experimental Study	Supports scalable evaluation	Accuracy and Reliability
39	[3]	2023	AI for Programing Colaboration	Model Development	Enables group learning	Interactivity, Creativity
40	[106]	2022	Code Correctness in AI	Benchmark Analysis	Evaluates function & accuracy	Accuracy and Reliability
41	[132]	2023	AI Frameworks at Scale	Framework Development	Facilitates wide deployment	Practical Usability
42	[73]	2023	Feedback Loops in Programing	Case Study	Encourages timely revisions	Interactivity, Prompt Engineering

Appendix 4: Pretest and Posttest Questions

The following table outlines the 10 questions used in both the pretest and posttest to assess programming knowledge before and after the intervention. Each question carried **5 marks**, totaling **50 marks**.

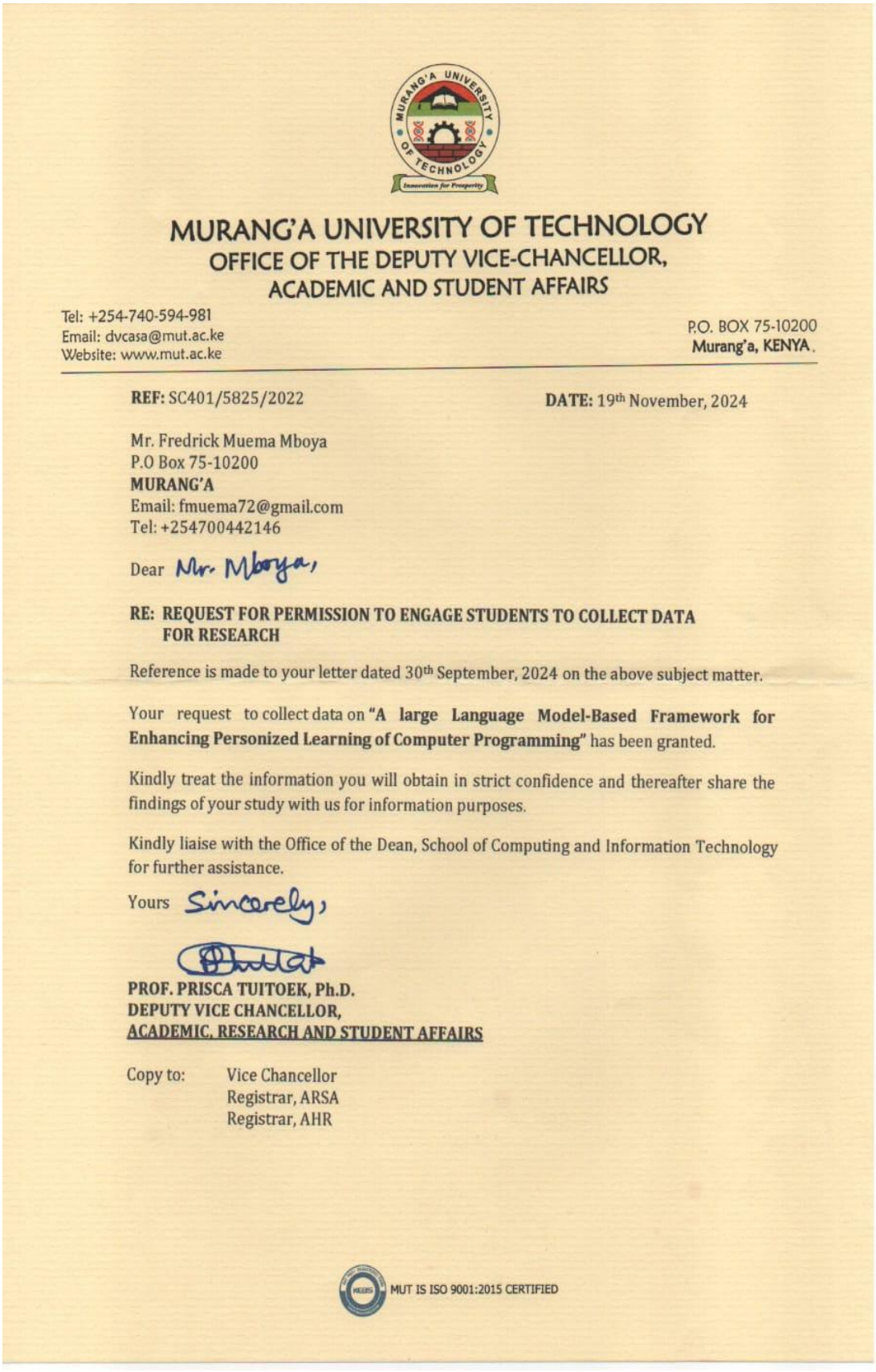
No.	Question	Marks
1	Write a C program to calculate the sum of two numbers.	5
2	Explain the use of loops in C programming with examples.	5
3	Write a C program to find the largest of three numbers using if-else.	5
4	What are functions in C? Write a user-defined function example.	5
5	Debug the following C code segment.	5
6	Explain the difference between call by value and call by reference in C.	5
7	Write a C program using arrays to calculate average marks for 5 students.	5
8	Describe pointers and write a C program that uses a pointer to swap two values.	5
9	Implement a menu-driven program for basic arithmetic operations.	5
10	Explain syntax errors vs logical errors with examples in C.	5

Appendix 5: Framework Interface



Code available at: https://github.com/fredy7274/e-learning-dashboard

Appendix 6: Approval Letter for Data Collection



MURANG'A UNIVERSITY OF TECHNOLOGY
OFFICE OF THE DEPUTY VICE-CHANCELLOR,
ACADEMIC AND STUDENT AFFAIRS

Tel: +254-740-594-981
Email: dvcasa@mut.ac.ke
Website: www.mut.ac.ke

P.O. BOX 75-10200
Murang'a, KENYA.

REF: SC401/5825/2022

DATE: 19th November, 2024

Mr. Fredrick Muema Mboya
P.O Box 75-10200
MURANG'A
Email: fmuema72@gmail.com
Tel: +254700442146

Dear Mr. Mboya,

RE: REQUEST FOR PERMISSION TO ENGAGE STUDENTS TO COLLECT DATA FOR RESEARCH

Reference is made to your letter dated 30th September, 2024 on the above subject matter.

Your request to collect data on **"A large Language Model-Based Framework for Enhancing Personized Learning of Computer Programming"** has been granted.

Kindly treat the information you will obtain in strict confidence and thereafter share the findings of your study with us for information purposes.

Kindly liaise with the Office of the Dean, School of Computing and Information Technology for further assistance.

Yours Sincerely,

PROF. PRISCA TUITOEK, Ph.D.
DEPUTY VICE CHANCELLOR,
ACADEMIC, RESEARCH AND STUDENT AFFAIRS

Copy to: Vice Chancellor
Registrar, ARSA
Registrar, AHR

MUT IS ISO 9001:2015 CERTIFIED

Appendix 7: NACOSTI License

REPUBLIC OF KENYA

NATIONAL COMMISSION FOR SCIENCE,TECHNOLOGY & INNOVATION

Ref No: **160685**

Date of Issue: **13/September/2024**

RESEARCH LICENSE



This is to Certify that Mr.. Fredrick Muema Mboya of Murang'a University of Technology, has been licensed to conduct research as per the provision of the Science, Technology and Innovation Act, 2013 (Rev.2014) in Muranga on the topic: A LARGE LANGUAGE MODEL-BASED FRAMEWORK FOR ENHANCING PERSONALIZED LEARNING OF COMPUTER PROGRAMMING for the period ending : 13/September/2025.

License No: **NACOSTI/P/24/39908**

160685

Applicant Identification Number

Director General
NATIONAL COMMISSION FOR SCIENCE,TECHNOLOGY & INNOVATION

Verification QR Code

NOTE: This is a computer generated License. To verify the authenticity of this document, Scan the QR Code using QR scanner application.

See overleaf for conditions

Appendix 8: Publication

The following paper has been published from this thesis.

[1] Mboya, F.M., Wambugu, G.M., Oirere, A.M., Omuya, E.O., Musyoka, F.M., & Gikandi, J.W. (2025). Enhancing personalized learning in programming education through Generative Artificial Intelligence Frameworks: A systematic literature review. *International Journal of Advanced Trends in Computer Science and Engineering*, 14(2), pp. 61–71. https://doi:10.30534/ijatcse/2025/051422025.