BIOMENTOR - PERSONALIZED E-LEARNING PLATFORM FOR ENGLISH MEDIUM A/L BIOLOGY SUBJECT STUDENTS IN SRILANKA

(LLM BASED ADAPTIVE QUIZ PLATFORM TO IMPROVE MCQ ANSWERING SKILLS IN BIOLOGY FOR A/L STUDENTS)

24-25J-257

Project Final Thesis

Sujitha. S – IT21264634

B.Sc. (Hons) in Information Technology
Specializing in SoftwareEngineering

Department of Computer Science & Software Engineering

Sri Lanka Institute of Information
TechnologySri Lanka

April 2025

BIOMENTOR - PERSONALIZED E-LEARNING PLATFORM FOR ENGLISH MEDIUM A/L BIOLOGY SUBJECT STUDENTS IN SRILANKA

(LLM BASED ADAPTIVE QUIZ PLATFORM TO IMPROVE MCQ ANSWERING SKILLS IN BIOLOGY FOR A/L STUDENTS)

24-25J-257

Project Final Thesis

Sujitha. S – IT21264634

B.Sc. (Hons) in Information Technology
Specializing in SoftwareEngineering

Department of Computer Science & Software Engineering

Sri Lanka Institute of Information
TechnologySri Lanka

April 2025

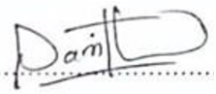
DECLARATION

I declare that this is my own work, and this Thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my Thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Sujitha.S	IT21264634	

The above candidate has carried out this research thesis for the Degree of Bachelor of Science (honors) Information Technology (Specializing in Software Engineering) under my supervision.

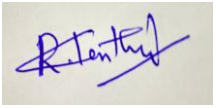


11/04/2025

Signature of the supervisor

Date

(Dr. Sanvitha Kasthuriarachchi)



11/04/2025

Signature of co-supervisor

Date

(Ms. Karthiga Rajendran)

ABSTRACT

The increasing emphasis on personalized learning in modern education has underscored the need for intelligent systems that can adapt to individual student needs. This project presents the design and implementation of an adaptive quiz platform tailored for GCE Advanced Level (A/L) Biology students studying in the English medium. The system is specifically developed to support students in preparing for their final A/L examinations by offering dynamically generated multiple-choice questions that reflect their evolving performance levels.

The platform delivers an engaging and effective self-assessment experience by adjusting the difficulty of questions in real time, based on user accuracy, response time, and historical performance trends. It supports both unit-based quizzes aligned with the official syllabus and adaptive quizzes that evolve according to each student's ability. In addition, it provides a comprehensive dashboard through which students can track progress, identify subject areas requiring revision, and receive personalized feedback aimed at continuous improvement.

The system not only evaluates student performance but also derives analytics such as accuracy trends, time management patterns, and engagement consistency. All content is syllabus-aligned, ensuring that every quiz contributes meaningfully to final exam preparation. The platform is designed for ease of use, enabling students to navigate quizzes, review results, and engage with content without technical complexity.

Through this solution, learners are empowered to take control of their preparation with content that is both relevant and appropriately challenging. By continuously adapting to the learner's pace and capability, the system promotes self-confidence, reduces exam anxiety, and enhances motivation through measurable improvement. Overall, the project demonstrates how adaptive technology can transform assessment into a more responsive, student-centered learning process.

Keywords: Adaptive Learning, MCQ Generation, A/L Biology, Student Performance, Personalized Assessment, Learning Analytics

ACKNOWLEDGEMENT

I wish to express my heartfelt appreciation to all those who have played an instrumental role in the success of this endeavour. My sincere gratitude goes to everyone who provided guidance, encouragement, and support throughout the course of this project.

First and foremost, I extend my deepest thanks to my supervisor, **Dr. Sanvitha Kasthuriarachchi**, whose mentorship, consistent feedback, and unwavering support over the past year were pivotal to the successful completion of this work. I am equally grateful to my co-supervisor, **Ms. Karthiga Rajendran**, for her continued guidance, timely suggestions, and constructive insights that helped shape the direction of the project.

I would also like to acknowledge the valuable contribution of my external supervisor, **Mrs. Nagalatha Thayaparan**, an A/L Biology teacher at Saiva Mangaiyar Vidyalayam, whose expertise and support were instrumental during the dataset preparation phase. Her assistance in collecting biology MCQs and classifying them by difficulty level greatly enriched the foundation of this project.

My heartfelt thanks go out to my family, friends, and seniors, whose constant encouragement and belief in me provided the strength and motivation to overcome every challenge. I am also thankful to the lecturers and colleagues at the **Sri Lanka Institute of Information Technology** for their thoughtful feedback, discussions, and advice throughout this journey.

Lastly, I would like to thank all those who contributed during the requirement gathering stage and supported various aspects of the project. Your collective efforts have been invaluable in bringing this project to life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviations	Description
A/L	Advanced Level (Sri Lankan General Certificate of Education)
AI	Artificial Intelligence
ML	Machine Learning
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment
FAISS	Facebook AI Similarity Search
GCE	General Certificate of Education
GPU	Graphics Processing Unit
CSV	Comma-Separated Values
IDE	Integrated Development Environment
IRT	Item Response Theory
JSON	JavaScript Object Notation
LLM	Large Language Model
LoRA	Low-Rank Adaptation
MCQ	Multiple Choice Question
SDLC	Software Development Life Cycle
NLP	Natural Language Processing
PEFT	Parameter-Efficient Fine-Tuning
RAG	Retrieval-Augmented Generation
JS	Java Script
SLIIT	Sri Lanka Institute of Information Technology
UI/UX	User Interface / User Experience
UAT	User Acceptance Testing
VM	Virtual Machine
VS Code	Visual Studio Code

Table 1: List Of Abbreviations

1. INTRODUCTION

The advancement of digital technologies has revolutionized modern education, enabling the development of learning environments that are more interactive, personalized, and accessible. Among these innovations, adaptive learning systems have gained significant prominence for their ability to cater to the unique learning pace, style, and performance of individual students. These systems have proven to be particularly effective in high-stakes academic contexts, where students often face challenges in mastering a wide range of topics within limited timeframes.

In the context of Sri Lankan education, the General Certificate of Education Advanced Level (GCE A/L) examination serves as a critical academic milestone that determines university admissions and future career pathways. Biology, being one of the most content-heavy and conceptually complex subjects in the A/L curriculum, demands continuous revision and rigorous practice. Traditional methods of learning and assessment, such as printed past papers and static question banks, are often limited in their ability to provide real-time feedback or adjust according to a student's evolving understanding.

This project addresses these limitations by introducing an intelligent and adaptive quiz platform specifically designed for English medium A/L Biology students. The platform not only supports comprehensive syllabus coverage through unit-based quizzes but also dynamically adjusts the difficulty of questions using performance-driven metrics such as accuracy, response time, and user history. The system empowers students with instant feedback and personalized insights via a dashboard, thereby promoting continuous improvement and sustained motivation.

The primary objective of this project is to bridge the gap between conventional assessment methods and the potential of adaptive educational technology. By leveraging the capabilities of artificial intelligence and data-driven evaluation, this

platform aims to enhance exam preparedness, promote learner autonomy, and foster more effective revision practices tailored to each student's learning trajectory.

1.1 Background Study and Literature Review

1.1.1 Background Study

The General Certificate of Education Advanced Level (GCE A/L) examination plays a central role in shaping the academic and professional futures of students in Sri Lanka. It is a highly competitive and rigorous national assessment, and success in this examination is a critical requirement for university admission. Among the science subjects offered at A/L, Biology is widely regarded as one of the most challenging, due to its combination of theoretical concepts, diagram-based content, and memory-intensive structures. The subject demands consistent effort, deep understanding, and the ability to apply learned material to a variety of unfamiliar contexts, particularly in the multiple-choice question (MCQ) section of the exam.

Traditionally, A/L Biology students rely on a combination of printed past paper books, tuition notes, model paper collections, and textbooks for their exam preparation. While these resources remain valuable, they are static and non-interactive, offering the same set of questions irrespective of the student's current knowledge level or learning progression. These methods fail to account for differences in student ability and do not provide real-time feedback or guided improvement. As a result, many students are unable to clearly identify their strengths and weaknesses, or to efficiently focus their revision on the areas where improvement is most needed.

Moreover, traditional systems lack the capacity to adapt dynamically. A student who repeatedly performs well on basic-level questions is not automatically challenged with harder content. Similarly, a student struggling with core concepts may continue to face high-difficulty questions without receiving intermediate support. These gaps often lead to frustration, reduced motivation, and ineffective use of valuable study time—especially as the examination approaches.

Over the last decade, the concept of adaptive learning has gained considerable traction in global education systems. Adaptive learning platforms aim to personalize the learning experience by using real-time data to adjust the complexity, order, and type of content delivered to a student. These systems are capable of analysing a learner's performance history to make informed decisions about what content should be shown next—whether to reinforce a weak area or introduce more advanced material. This not only improves engagement but also helps students learn more efficiently by focusing on what they truly need to master.

Despite the demonstrated benefits of adaptive systems, their application in the Sri Lankan A/L context remains limited. Most available educational tools are either generalized platforms that do not align with the local curriculum or static quiz apps that lack intelligence and personalization. There is a notable absence of intelligent systems tailored specifically for subject-wise, syllabus-aligned, and performance-driven learning in A/L Biology.

To address this critical gap, this project introduces an intelligent and adaptive quiz platform developed exclusively for A/L Biology students. The system goes beyond static question delivery by integrating Large Language Models (LLMs) to generate original, syllabus-relevant MCQs in real time. These models are guided by context-aware inputs and past user data to ensure the questions are appropriate in difficulty, relevant in content, and varied across sessions. The platform also incorporates an adaptive engine based on Item Response Theory (IRT), which assesses student performance across difficulty levels and dynamically adjusts future quizzes to suit the learner's evolving capability.

Furthermore, the platform includes unit-based quiz generation, enabling focused revision on specific chapters, and a performance dashboard that provides detailed analytics on accuracy, timing, improvement trends, and consistency. These features combine to deliver a holistic and personalized revision experience that supports deeper

learning, builds confidence, and prepares students more effectively for the A/L Biology examination.

1.1.2 Literature Review

Over the past decade, education systems worldwide have increasingly integrated intelligent digital solutions to enhance teaching, learning, and assessment. One of the most significant advancements has been the introduction of adaptive learning technologies, which aim to personalize the educational experience based on a learner's unique strengths, weaknesses, and progress. Adaptive learning has proven particularly effective in assessment-driven domains, where feedback and targeted practice are critical to academic success.

In this context, automated multiple-choice question (MCQ) generation has become a key area of interest. Early approaches to MCQ generation involved rule-based systems and static templates, which limited question diversity and adaptability. However, with the emergence of Large Language Models (LLMs) and transformer-based architectures, such as BERT and T5, the process of question generation has undergone a substantial transformation. These models have shown the ability to generate semantically coherent, syllabus-aligned questions from textual data using techniques such as prompt engineering, keyword extraction, and retrieval-based augmentation [1]–[5].

Modern LLM-driven systems are capable of producing MCQs that vary in difficulty and linguistic structure, allowing them to serve a diverse range of learners. They also support curriculum-focused generation by incorporating contextual prompts derived from syllabus content or textbook data. Several research initiatives have proposed methods to automate question generation from educational materials by training or prompting language models with topic-tagged input, thereby improving both relevance and quality [2], [8]. These findings provide foundational support for platforms that aim to use LLMs in localized educational contexts, such as GCE A/L Biology.

Alongside content generation, adaptivity is a critical factor in effective educational systems. Adaptive quiz platforms rely on performance metrics such as accuracy, consistency, and response time to make real-time decisions on question delivery. One

of the most widely adopted frameworks to achieve this is Item Response Theory (IRT). IRT enables systems to dynamically estimate a student's ability level and adjust the difficulty of future questions accordingly. Research has demonstrated that IRT-based models improve learning outcomes by ensuring that learners are challenged appropriately—neither underwhelmed by simple content nor overwhelmed by material beyond their current capabilities [6], [9].

Furthermore, studies have emphasized the role of learner analytics in enhancing the value of adaptive systems. Features such as performance dashboards, time tracking, scoring breakdowns, and progress visualization have been shown to positively influence student motivation and engagement. These features also support educators in identifying learning gaps and personalizing instruction. Such analytical components are commonly embedded in intelligent tutoring systems and form an integral part of user-centric quiz platforms [7], [10].

While the benefits of adaptive assessment and AI-driven MCQ generation are well documented, most existing platforms focus on general education or cater to international curricula. Very few solutions are tailored to localized, syllabus-specific learning environments, such as the Sri Lankan A/L examination system. The majority of existing systems also lack fine-grained control over question difficulty and do not offer dynamic generation capabilities aligned with real-time user performance.

The proposed project builds upon this body of work by combining three key innovations: (i) syllabus-aligned MCQ generation using LLMs, (ii) adaptive delivery logic based on IRT, and (iii) real-time learner analytics and progress tracking. Unlike generic quiz applications, this platform is specifically designed for English medium A/L Biology students, providing both unit-based quiz modules and adaptive quizzes that evolve with the learner. It aims to close the gap between theoretical advancements in educational AI and practical implementation in localized, examination-oriented settings.

In summary, the current literature highlights the effectiveness of AI-powered adaptive systems in education, particularly when driven by language models and performance-

based logic. However, there remains a clear need for domain-specific, curriculum-focused solutions in underrepresented educational contexts. This project aims to address that gap by delivering an intelligent, adaptive learning experience tailored to the real needs of A/L Biology learners in Sri Lanka.

1.2 Research Gap

While significant advancements have been made in educational technology, particularly in the areas of intelligent tutoring systems, adaptive learning, and automated assessment generation, a number of critical gaps remain — especially when it comes to the application of these innovations in localized, syllabus-bound, high-stakes examination contexts such as Sri Lanka's General Certificate of Education Advanced Level (GCE A/L) examination.

Firstly, most existing adaptive learning platforms are either designed for general education purposes or focused on internationally recognized curricula. These platforms often lack alignment with the specific content and structure of the Sri Lankan A/L Biology syllabus. Consequently, they fall short of meeting the needs of local students who require highly targeted practice material that mirrors the depth, scope, and complexity of the actual examination content.

Secondly, although recent research has demonstrated the potential of Large Language Models (LLMs) for automatic MCQ generation, their practical use remains limited to experimental or broad academic domains. Many tools that use LLMs or transformer-based architectures generate questions from general-purpose texts and lack mechanisms to ensure curriculum alignment or content filtering based on subject-specific requirements. There is also limited exploration of how such models can be used for real-time, adaptive quiz generation at scale, particularly in school-level subjects like Biology.

Thirdly, even in systems that offer adaptive quiz experiences, the adaptivity is often static or pre-defined — where difficulty levels are manually set or progress is linear. There is insufficient integration of learner performance data and Item Response Theory (IRT) to dynamically adjust difficulty levels in real time based on a student's evolving proficiency. This results in learning experiences that either do not challenge the student

enough or introduce unnecessary cognitive load, thereby undermining their confidence and engagement.

Furthermore, existing quiz platforms rarely offer integrated analytics and personalized feedback, which are essential components of effective self-assessment. Students using traditional tools are often unaware of their progress over time, topic-wise performance breakdowns, or how their current ability compares across different difficulty levels. Without such insights, learners are unable to fine-tune their revision strategies or monitor their growth effectively.

Lastly, there is a noticeable lack of intelligent learning tools that cater specifically to English medium A/L Biology students in Sri Lanka, a demographic that often lacks access to syllabus-matched, ML-assisted preparation tools. The current ecosystem does not offer solutions that combine adaptive learning, ML-driven MCQ generation, syllabus alignment, and performance analytics in a single, unified platform.

These gaps collectively point to the need for a solution that not only automates question generation using LLMs but also personalizes quiz delivery using adaptive learning techniques grounded in psychometric models like IRT. Such a solution must be curriculum-specific, user-friendly, and capable of providing continuous, data-informed feedback — all while being tailored to the contextual and academic realities of A/L Biology students preparing in the English medium.

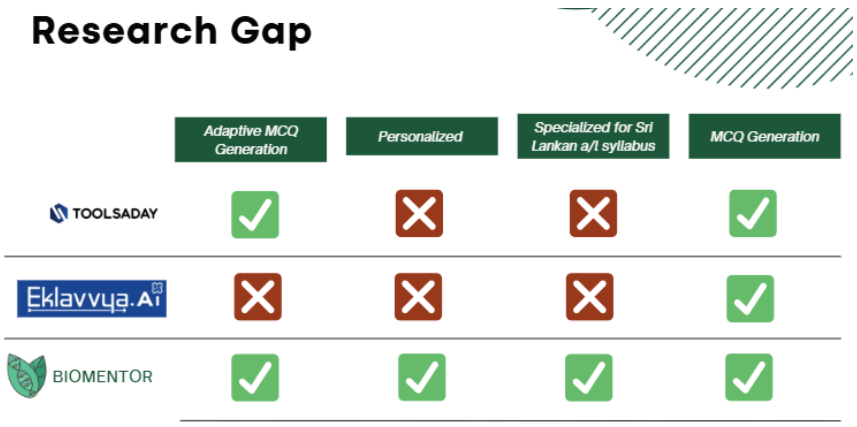


Figure 1: Novel Diagram

1.3 Research Problem

Despite growing advancements in adaptive learning technologies and automated assessment systems, students preparing for the GCE A/L Biology examination in Sri Lanka continue to rely on conventional and non-personalized learning resources. Printed model papers, fixed question banks, and tutor-based materials dominate the current academic landscape, offering little to no interactivity or individualized support. These resources are static in nature and fail to respond to the varying abilities and learning trajectories of individual students. As a result, learners often engage with material that is either repetitive or misaligned with their actual proficiency, leading to inefficient preparation and limited academic growth.

Several adaptive platforms and intelligent tutoring systems have emerged globally, integrating artificial intelligence to improve learner engagement and performance [1]–[5]. However, most of these tools are not designed for syllabus-specific use, and they rarely incorporate real-time performance data to guide question delivery. Although some systems use pre-defined question tiers or progression paths, these approaches do not dynamically adjust to learners using psychometric models like Item Response Theory (IRT), which have proven effective in estimating learner ability and adjusting difficulty accordingly [6], [9].

Moreover, while recent studies have highlighted the capability of Large Language Models (LLMs) to generate high-quality, syntactically sound MCQs [3]–[5], their practical application in curriculum-aligned assessment remains limited. Many current implementations of LLM-based question generation lack mechanisms for ensuring that the output is consistent with local examination requirements, particularly in subject-specific contexts like A/L Biology. The generated questions may appear valid linguistically, but without proper control over content relevance, cognitive level, and syllabus alignment, their educational value remains questionable.

In addition to the lack of personalized content and adaptivity, existing platforms often overlook the importance of learner feedback and analytics. Students are rarely given insight into their performance trends, strengths and weaknesses by topic, or progress over time. Without such feedback loops, learners are unable to revise efficiently or strategically improve their weaker areas [7], [10].

Taken together, these challenges highlight a pressing need for a domain-specific, intelligent assessment system that can not only generate curriculum-aligned MCQs using LLMs, but also adaptively deliver content based on individual performance metrics. The absence of such a platform specifically tailored to the needs of English medium A/L Biology students in Sri Lanka creates a gap in the current edtech landscape that this research seeks to address. By leveraging both LLM-driven generation and IRT-based adaptivity, this project proposes a system that offers dynamic quiz experiences, real-time difficulty adjustment, and personalized performance tracking—ultimately supporting more effective, targeted, and engaging examination preparation.

1.4 Research Objectives

1.4.1 Main Objective

To design and develop an intelligent, adaptive quiz platform specifically tailored for GCE A/L Biology students in Sri Lanka that dynamically generates curriculum-aligned multiple-choice questions using Large Language Models (LLMs), adjusts question difficulty in real time based on student performance using Item Response Theory (IRT), and provides personalized feedback through learner analytics.

1.4.2 Specific Objectives

The following are the sub-objectives of conducting this research.

- **To develop an MCQ generation module** that leverages Large Language Models (LLMs) to create high-quality, syllabus-aligned Biology questions with varying levels of difficulty.
- **To implement adaptive quiz delivery** by integrating Item Response Theory (IRT) to adjust the difficulty of questions based on a student's performance, response accuracy, and time taken.
- **To enable unit-wise quiz functionality**, allowing students to select and practice questions based on specific topics or chapters from the A/L Biology curriculum.
- **To build a performance tracking dashboard** that visualizes student progress over time, highlights topic-wise strengths and weaknesses, and provides actionable feedback to guide revision.
- **To minimize redundancy in question delivery** by employing semantic similarity filtering techniques and question embedding comparisons to avoid repetition and enhance content diversity.
- **To evaluate the effectiveness of the adaptive system** through testing, feedback collection, and performance comparisons between adaptive and non-adaptive quiz sessions.

1.4.3 Business Objectives

- **To provide a cost-effective and scalable digital solution** that enables A/L Biology students to access intelligent, adaptive self-assessment tools without the need for expensive tuition or printed material.
- **To increase student engagement and retention** by offering a personalized learning experience that dynamically adapts to their performance, thereby reducing exam-related stress and improving confidence.
- **To bridge the gap between conventional exam preparation and AI-enhanced learning**, making cutting-edge educational technology accessible to students in both urban and rural areas.
- **To create a platform that supports continuous improvement**, where students can track their progress, set academic goals, and revisit weaker areas using performance analytics and structured feedback.
- **To offer value to educational institutions and tutors**, who can integrate the platform into their teaching models, monitor student performance, and supplement their lessons with AI-generated, syllabus-aligned assessments.
- **To establish a strong foundation for future commercial growth**, by extending the platform to other A/L subjects, multiple language mediums, or educational levels, thereby addressing a broader market segment.

Strengths	Weaknesses
- Domain-specific (A/L Biology) - Uses LLMs for question generation - Real-time adaptivity with IRT - Personalized dashboard and analytics	- Requires internet access - Dependence on LLM API (e.g., latency or cost) - Limited to English medium only in initial version
Opportunities	**Threats**
- Expansion to other subjects or languages - Integration with school LMS - Commercial licensing for schools & tutors	- Competing platforms entering the market - Potential ethical/legal issues with AI content - Syllabus changes by education authorities

Figure 2:SWOT Analysis Diagram

2. METHODOLOGY

2.1 Methodology

Methodology in research refers to the structured process and systematic techniques used to investigate, develop, and validate solutions to defined problems. It encompasses the strategies employed to collect, analyze, and interpret data, ensuring the study is both rigorous and reliable. A well-defined research methodology provides clarity in the direction of the project and supports transparent, repeatable outcomes. In this study, the methodology was carefully designed to support the development of an intelligent, adaptive quiz platform that assists GCE A/L Biology students in exam preparation through personalized assessment. The research approach integrates both academic inquiry and technical development, ensuring that the solution is grounded in educational relevance while supported by empirical and iterative refinement.

To guide this process, the research adopted the Agile methodology in combination with a Seven-Stage Development Framework. Agile methodology, originally rooted in software engineering, has become increasingly valuable in academic and applied research settings due to its emphasis on adaptability, incremental development, and ongoing stakeholder engagement. In the context of this project, Agile allowed for continuous feedback integration, reflection, and iterative improvements to the system's core components — particularly those involving complex, evolving domains such as question generation using large language models and adaptive assessment using psychometric principles.

1. Agile Methodology for Research Development:

Agile methodology was applied as a dynamic and collaborative project management strategy to support the evolving requirements of the research. The platform's goal — to personalize student assessments in a meaningful and syllabus-aligned manner — required constant iteration and responsiveness to emerging challenges and insights. By

dividing the project into smaller, manageable units, the research team was able to incrementally build, test, and refine features such as adaptive question generation, learner performance tracking, and analytics visualization. Regular feedback from peers, supervisor, co-supervisor, external supervisor and a/l biology students guided the refinement of user experience, content quality, and assessment adaptivity. Agile's iterative structure ensured that adjustments could be made quickly in response to unexpected limitations or the discovery of better alternatives.

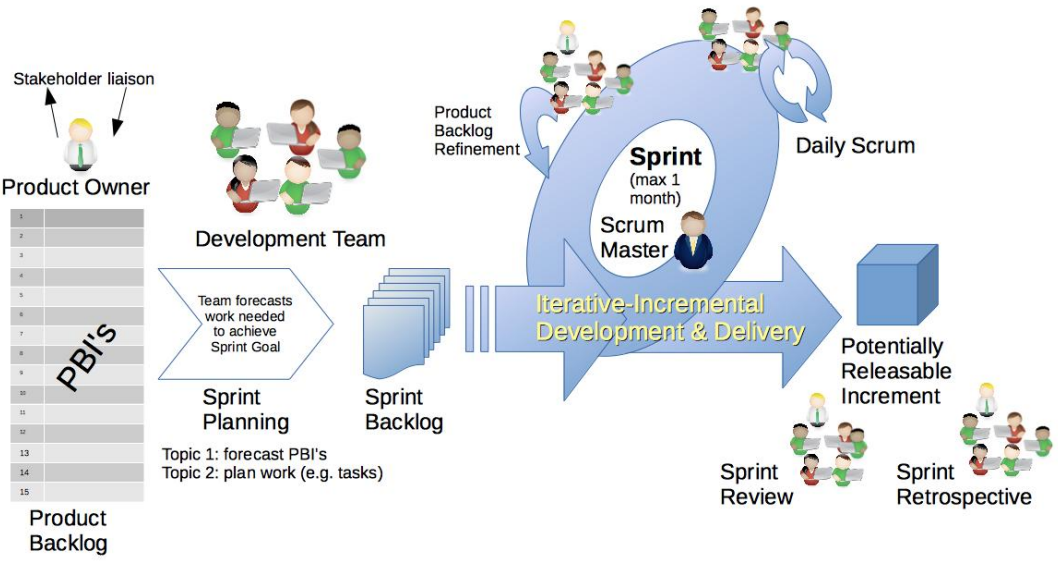


Figure 3: Agile + Scrum Workflow Diagram

2. Seven-Stage Development Framework:

The development process was organized following a Seven-Stage Development Framework, ensuring clarity, structure, and academic rigor across all phases of the research. These stages included: (1) Feasibility Study and Planning, (2) Requirement Gathering and Analysis, (3) System Design, (4) Implementation, (5) Testing, (6) Deployment and Maintenance, and (7) Commercialization. Each stage contributed to a deeper understanding of the problem domain while enabling structured progress toward

the research objectives. The framework provided a comprehensive roadmap for evaluating educational needs, translating them into technical features, and systematically validating their effectiveness through testing and feedback.

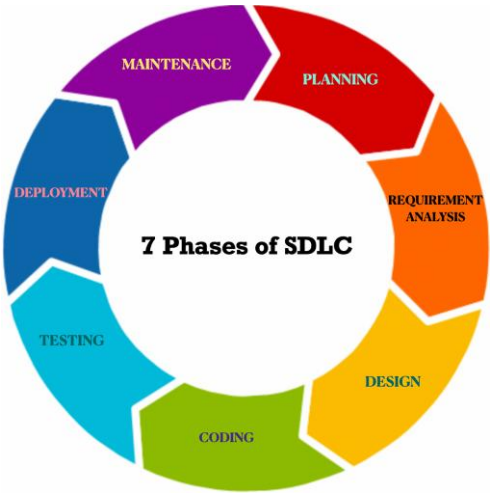


Figure 4: Development Lifecycle Diagram

3. Iterative Development and Collaboration:

A core principle of Agile methodology is incremental and collaborative development, which proved essential for this research. The research was broken down into smaller units aligned with the seven development stages, and each component was iteratively designed and refined. For example, the adaptive logic that determined question difficulty based on user performance underwent several iterations to achieve the desired balance between student challenge and support. Experimental testing with sample users played a central role in guiding development priorities and design decisions. This collaborative and feedback-driven cycle ensured that the system remained aligned with its educational goals throughout the project timeline.

4. Flexibility and Responsiveness:

Agile's flexibility allowed the research to remain responsive to unexpected changes and evolving academic insights. Whether addressing challenges related to the quality

of generated MCQs, or refining the feedback mechanism on the learner dashboard, the ability to pivot quickly and revise methodologies was key to the project's progress. This responsiveness ensured the system remained learner-centered and practical while maintaining a high degree of academic and pedagogical integrity. The adaptive quiz platform's development involved exploration, experimentation, and recalibration — all of which were made possible through Agile's emphasis on flexibility and continuous learning.

In summary, the combination of the Agile methodology and the Seven-Stage Development Framework enabled a systematic yet adaptable research process. It ensured that the final product was not only technically functional but also aligned with the academic, psychological, and practical needs of A/L Biology students. The following sections will describe each development stage in detail, elaborating on how they contributed to the design, validation, and realization of the proposed solution.

2.1.1 Feasibility Study/ Planning

This phase assesses the feasibility of designing and implementing an adaptive quiz platform tailored for GCE A/L Biology students. The system aims to dynamically generate syllabus-aligned multiple-choice questions (MCQs) using Large Language Models (LLMs) and adjusts question difficulty based on student performance using Item Response Theory (IRT). The feasibility study focuses on evaluating the technical, economic, legal, operational, and scheduling viability of the research to ensure practical implementation and long-term sustainability.

1. Technical Feasibility:

Data Availability: A foundational requirement for developing an adaptive assessment platform is access to a diverse and high-quality question dataset. For this research, a dataset of multiple-choice questions is manually compiled from A/L examination papers and school-level test papers, with the direct support of the external supervisor. These questions are reviewed and categorized by difficulty level—ranging from easy to hard—based on pedagogical judgment and curriculum standards. This process ensures that the dataset is syllabus-aligned, representative of real exam formats, and suitable for use in both adaptive delivery and LLM-based generation prompts.

System requirements and Infrastructure: The platform is designed with cloud deployment in mind, using modern web-based technologies that scale efficiently. The MCQ generation module, which uses LLMs for prompt-based content generation, is hosted externally to manage computational demands. The core platform architecture—including adaptive logic, quiz management, and performance tracking—is optimized for minimal resource usage, ensuring cost-effective deployment using standard infrastructure.

Model Integration and Adaptivity: The technical feasibility of building an adaptive system using Item Response Theory (IRT) is evaluated through existing implementations and studies in educational technology. By mapping user

performance metrics (accuracy, speed, consistency) to difficulty levels, the system estimates learner ability and adjusts future quiz content accordingly. The integration of LLMs for content generation is tested using prompt engineering techniques to control difficulty and topic coverage, demonstrating that it is technically viable to generate educationally meaningful MCQs aligned with the local curriculum.

2. Economic Feasibility:

Budgetary Considerations: The financial requirements for this research project remain minimal and within manageable limits. The primary cost involves the use of Google Colab Pro for fine-tuning and executing LLM-based question generation, which provides faster processing and enhanced reliability during development. All other components—including backend development, testing, and analytics—are implemented using free and open-source tools. No additional licenses are required for development.

Resource Allocation: The project uses existing infrastructure and freely available tools for development and testing. Domain-specific support for MCQ classification and syllabus alignment is provided by an experienced A/L Biology teacher, ensuring educational validity without incurring any additional personnel costs. The tools and platforms are compatible with standard computing environments and require no specialized hardware or commercial software.

Return on Investment (ROI): The platform offers strong potential to enhance A/L Biology exam preparation through personalized, adaptive quizzes. Its ability to improve learning efficiency and engagement makes it highly valuable to both students and educational institutions. Considering the low financial investment and the potential for large-scale educational impact, the project is considered economically viable with a high return on value in both academic and practical terms.

Table 1 shows the cost management/ economic feasibility of the research.

Type	Cost
Internet use and web hosting	10,000 LKR
Google Colab Pro Subscription	3,000 LKR
Publication cost	15,000 LKR
Stationery and printing	2,000 LKR
Azure Cloud Hosting (via Student Pack)	0 (Covered by Credits)
TOTAL	***30,000 LKR***

Table 2: Cost Management

3. Legal and Ethical Feasibility:

Data Privacy and Ethics: The platform collects only non-sensitive data such as quiz scores and response times, solely for educational feedback. No personal or sensitive data is used or shared. Ethical considerations were maintained throughout, ensuring data confidentiality and secure usage within the scope of learning.

Intellectual Property: All modules and components were developed using open-source tools with proper licensing. MCQs were sourced from publicly available past papers and school assessments with no copyright conflicts. All system outputs are original and used for educational purposes only.

4. Operational Feasibility:

Data Collection and Classification: MCQs were manually gathered from past A/L papers and school tests, with assistance from a subject expert. Each question was categorized by difficulty to support both adaptive quiz features.

System Usability and Response Time: The platform is designed for ease of use and runs efficiently in standard online environments. A key operational requirement is that questions must be generated on-demand when a user starts a quiz. Therefore, the system ensures fast response times during question generation to

maintain a smooth user experience. All core features, including adaptivity and analytics, operate with minimal delay and no need for high-end hardware.

5. Time/Schedule Feasibility:

Project Timeline: The research project follows a clear timeline, with defined milestones for data collection, design, development, testing, and refinement. Each phase is completed within the academic calendar, with sufficient time allocated for iterative improvements and evaluation. The project remains on track and is completed within the available timeframe without compromise to quality.

Figure 5: Gantt Chart (Schedule Management)

6. Social and Cultural Feasibility:

The platform is socially and culturally feasible as it supports the learning practices commonly followed by A/L students in Sri Lanka. It targets English medium Biology students and aligns strictly with the national syllabus, making it suitable for academic use. The system promotes self-assessment and personalized learning, which are increasingly accepted within local education environments.

The content is neutral and appropriate for school-level learners, avoiding any culturally sensitive or irrelevant material. The interface and language are kept simple and formal to ensure usability and acceptance across different regions and school settings.

Other than these feasibility studies, the risk management plan and communication management plan has been done.

- Risk Management Plan

In addition to feasibility analysis, a comprehensive risk management plan has been formulated to anticipate, address, and mitigate potential issues that may arise during the development and integration of the adaptive quiz component. This plan ensures continuity, minimizes disruptions, and enables proactive response strategies within the broader scope of the group project.

The following table outlines possible risks associated with this component, their triggers, responsible parties, planned responses, and required resources.

Risk	Trigger	Owner	Response	Resource Required
Delay in LLM model output or quiz generation	Model processing time increases;	Component Developer (Adaptive Quiz)	Optimize prompt structure and reduce token usage	Fine-tuned prompts Quiz Fallback Dataset

due to heavy compute load	slow quiz generation		Use fallback static MCQs if model timeout occurs	
Absence of a team member responsible for model integration or backend routes	Illness or other emergencies	Project Leader / Sub-Team Lead	Inform supervisor immediately Reassign backend tasks to another member with GitHub access	Gantt Chart Access to shared repos
API or route failure between quiz generation and frontend	Improper endpoint handling or server downtime	Adaptive Quiz Developer	Debug and redeploy route Notify frontend team Implement route health checks	Postman / FastAPI logs Debugging tools
Inconsistent difficulty progression in adaptive quizzes	Logic error in IRT score application	Component Developer	Conduct additional unit testing Log ability values during generation for review	IRT test scripts Performance logs
Feedback from testing sessions requires urgent changes	UAT feedback from teachers/students	Project Leader	Implement necessary changes Re-test and update documentat	Product Backlog Gantt Chart

			ion if needed	
Conflict in API design or data format between modules	Frontend/backend mismatch in payloads	Backend Developer (Adaptive Quiz)	Schedule sync meeting with frontend team Align API response format to agreed contract	OpenAPI schema Project meetings
Missed progress review due to panel unavailability	Supervisor or panel member absent	Project Leader	Reschedule review or submit written progress update	Meeting log Email documentation
Internal communication gap within the group	Lack of updates or unclear task distribution	Entire Group	Weekly sync-up meetings Update Jira/Gantt and documentation regularly	Communication tools Shared task tracker
Inconsistent code contributions or merge conflicts	Simultaneous development on the same module	Group Member (assigned to repo maintenance)	Follow Git workflow strictly Assign code review roles	GitHub project board Merge policy documentation

Table 3: Risk Management Plan

- Communication Management Plan

The Communications Management Plan helps to ensure that all team members, supervisor, and co-supervisor have the information they need to perform their roles throughout the project. Project success depends on the planning and execution of communication efforts. How to communicate with different stakeholders most effectively and efficiently is decided by the communications management plan. It outlines and documents the audience, communication items' content, format, frequency, and anticipated outcomes. Also, it specifies the various stakeholders' roles in the project, how to assign tasks to them, and the communication strategy for each of them based on their influence on the project, interests, and expectations.

Communication Objectives:

Proactive communication is important on all projects. Communication needs to be:

- Adequate: in the right format and right content.
- Specific: for the targeted audience.
- Sufficient: providing all the necessary information.
- Concise: brief, avoiding repetition and non-important information.
- Timely: addressing points at the right time.

Communication Media:

The communication media that will be used for the project are:

1. Email
2. Document (MS Word and/ PowerPoint)
3. Phone call
4. Meetings (using, meeting rooms, conference phones, Ms Teams)
5. Chats (WhatsApp)

Meeting Type	Attendees	Purpose	Frequency	Agenda Items
Planning Kick-off Meeting	Supervisor, Co-supervisor, All Team Members	Formally launch the project's planning phase. Define project scope, component assignments, team responsibilities, and expectations. Identify potential risks and agree on overall direction and novelty of the components.	Once at the start of the project	- Introduce project overview and research problem - Distribute components and outline responsibilities - Present initial timeline - Discuss expected outcomes and evaluation criteria - Identify high-level risks and mitigation - Confirm communication tools and methods - Recap and record key decisions
Execution Kick-off Meeting	Supervisor, Co-supervisor, All Team Members	Initiate the development phase of the project. Reconfirm roles, finalize milestones, present Communication Management	Once at the start of each major phase (Proposal, PP1, etc.)	- Present the finalized Work Plan - Outline communication flow and escalation paths - Reconfirm Quality

		Plan, and agree on team norms and workflows.		Assurance measures - Introduce issue/change management plan - Set up documentation protocols - Confirm upcoming reviews and evaluations
Internal Project Status Meeting	All Team Members	Review weekly progress of all components, share completed work, address technical challenges, and plan next steps. These meetings ensure continuous alignment between members and timely identification of blockers.	Weekly throughout the project	- Share status updates for each component - Compare actual work vs. planned tasks - Assess milestone progress - Identify new risks or issues - Set action items for the coming week - Record attendance and key outcomes
Actual Project Status Meeting	Supervisor, Co-supervisor, All Team Members	Provide formal progress updates to the supervisor(s). These include demos, presentations, or	Twice per week or as requested by the supervisor	- Present current component status - Highlight key deliverables

		summary reports to ensure each team member is on track with their responsibilities.		and demos - Discuss encountered problems - Get technical and research feedback - Log changes and improvements made
Project Review Meeting	Supervisor, Co-supervisor, All Team Members	Review the entire project's status and preparedness for milestone reviews like Proposal, PP1, PP2, or Final. Discuss potential re-baselining or refinements.	Once per phase (Before Proposal, PP1, PP2, Final)	- Review documentation readiness - Validate milestone achievements - Analyze testing and validation results - Address unresolved issues - Prepare for panel expectations and possible changes - Review research alignment and educational contribution
Project Steering Committee (PSC) Meeting	Supervisor, Co-supervisor, All Team Members	Address official project approvals and permissions. Used when major decisions or milestone	Monthly or at major milestone approval	- Debrief team performance to date - Record accomplish

		achievements are made. Ensures project stays aligned with academic requirements and timeline.		ments and setbacks - Note items pending for next milestone - Review budget, time, or scope constraints - Confirm supervisor approval for milestone progress - Ensure required panel feedback or sign-offs are documented
Change Control Meeting	Supervisor, Co-supervisor, All Team Members	Evaluate and prioritize required changes following evaluations or internal reviews. Commonly used post-panel feedback to initiate necessary modifications in reports, presentations, or implementations.	As needed after panel feedback or scope changes	- Discuss panel feedback - Prioritize changes - Assign responsibility to team members - Set timeline for updated implementation - Approve and document changes made
Project-End	Supervisor, Co-supervisor,	Conduct final evaluation of the project's	Once at the end of the project	- Summarize system performance

Review Meeting	All Team Members	outcomes and performance. Share individual and group reflections, lessons learned, and opportunities for future research or improvement.		and outputs - Review success in achieving goals - Discuss team collaboration experience - Identify technical or logistical issues faced - Share research insights and lessons - Plan long-term contributions (open-source, documentation reuse, etc.)

Table 4: Communication Management Plan

2.1.2 Requirement Gathering & Analysis

The Requirement Gathering and Analysis phase serves as a crucial step in identifying the functional and non-functional expectations of the adaptive quiz platform. This stage helps to define the overall scope of the system and ensures that its design and development align with the actual needs of GCE A/L Biology students. Through a structured survey, data was collected from target users to understand their preferences, challenges, and expectations regarding MCQ-based exam preparation.

2.1.2.1. Functional Requirements

Functional requirements represent the core capabilities and system behaviors necessary to meet the intended learning objectives. These requirements were identified through a survey conducted among A/L Biology students, which aimed to uncover the key issues they face in exam preparation and the type of features they would benefit from in a digital learning tool.

Survey Method: A digital questionnaire was designed and distributed among a selected group of English medium A/L Biology students. The survey included multiple-choice and open-ended questions to gather feedback on their current study practices. The responses were analyzed to extract recurring themes and prioritize system features based on user demand.

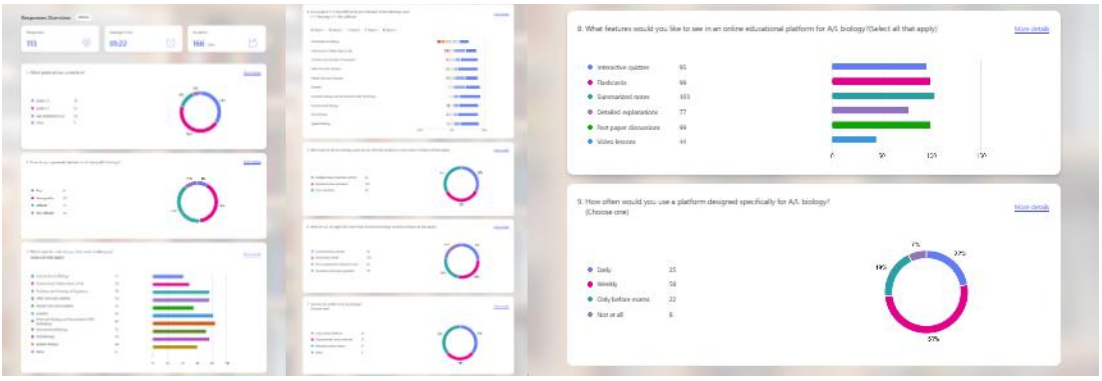
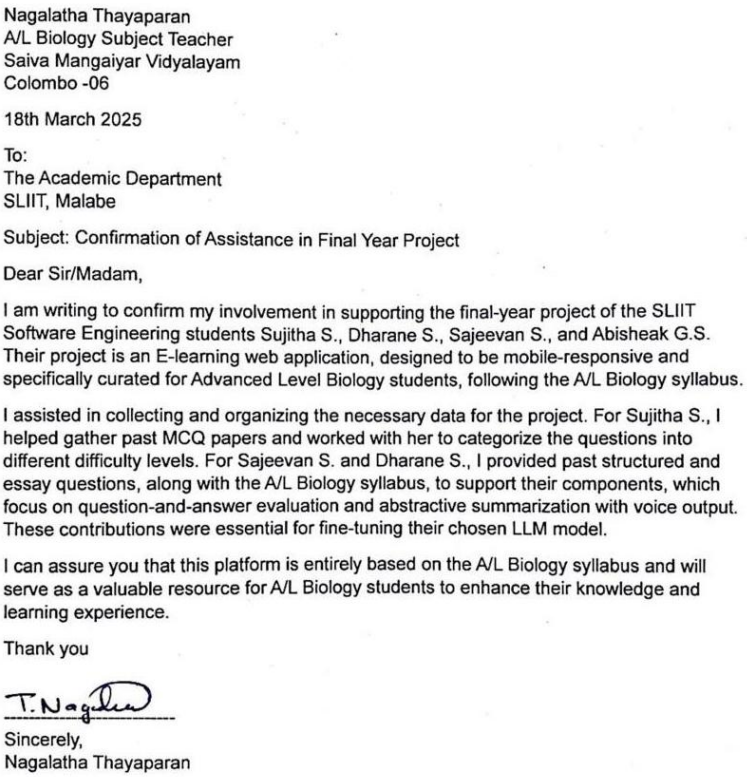


Figure 6: Survey Details

External Supervisor Input: External supervisor was consulted throughout the requirements engineering process. Her feedback guided the refinement of features and confirmed that proposed functionalities were educationally sound and technically feasible within the project scope.



Nagalatha Thayaparan
A/L Biology Subject Teacher
Saiva Mangaiyar Vidyalayam
Colombo -06

18th March 2025

To:
The Academic Department
SLIIT, Malabe

Subject: Confirmation of Assistance in Final Year Project

Dear Sir/Madam,

I am writing to confirm my involvement in supporting the final-year project of the SLIIT Software Engineering students Sujitha S., Dharane S., Sajeevan S., and Abisheak G.S. Their project is an E-learning web application, designed to be mobile-responsive and specifically curated for Advanced Level Biology students, following the A/L Biology syllabus.

I assisted in collecting and organizing the necessary data for the project. For Sujitha S., I helped gather past MCQ papers and worked with her to categorize the questions into different difficulty levels. For Sajeevan S. and Dharane S., I provided past structured and essay questions, along with the A/L Biology syllabus, to support their components, which focus on question-and-answer evaluation and abstractive summarization with voice output. These contributions were essential for fine-tuning their chosen LLM model.

I can assure you that this platform is entirely based on the A/L Biology syllabus and will serve as a valuable resource for A/L Biology students to enhance their knowledge and learning experience.

Thank you

T. Nagalatha

Sincerely,
Nagalatha Thayaparan

Figure 7: Confirmation from external supervisor

Interviews and Mock-up Testing: In addition to the survey, interviews were conducted with students during the UI/UX design phase. These sessions focused on gathering feedback on Figma-based mock-ups, helping refine the interface design and ensuring alignment with user expectations. Students provided valuable suggestions on navigation, quiz flow, and visual clarity, which were incorporated into subsequent design iterations.

Identification of Key Functionalities: Based on the interview results and feedback from external supervisor, the following core functionalities were identified as essential: adaptive quizzes that respond to student performance, unit-wise quizzes for targeted revision, performance tracking to visualize learning progress, and immediate feedback after quiz submission. The system is also expected to generate high-quality, syllabus-aligned MCQs dynamically, and avoid repetition of questions to ensure content variety.

Validation and Prioritization: The feedbacks were reviewed and converted into specific system requirements. Each functionality was categorized based on importance and frequency of mention in the survey. The prioritized features were then mapped into user stories, guiding the development of the platform in successive phases.

After the requirement gathering, the main requirements were mapped as user stories. Figure 6 illustrates the user stories.

Main functional requirements are mentioned below,

- Generate adaptive quizzes based on student performance using difficulty scaling.
- Allow unit-based quizzes aligned with specific chapters in the A/L Biology syllabus.
- Track and display quiz performance metrics over time and by topic.
- Provide real-time feedback upon quiz completion.
- Dynamically generate MCQs using LLM-based prompts aligned with the curriculum.
- Prevent repetition of questions through similarity filtering mechanisms.

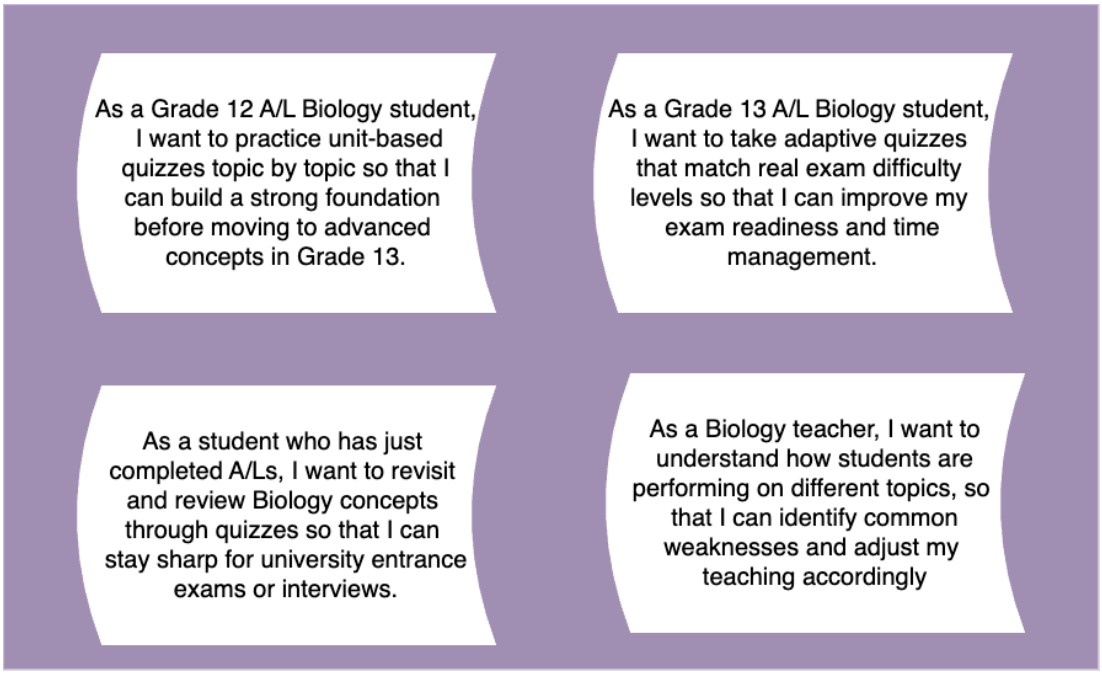


Figure 8: User Stories for Requirements

Mapping the user requirements in the use case diagram and sequence diagram is very important to identify the requirements clearly and it will be used to prioritize the requirements. Figure 6 explains the use case diagram and figure 7 illustrates the sequence diagram.

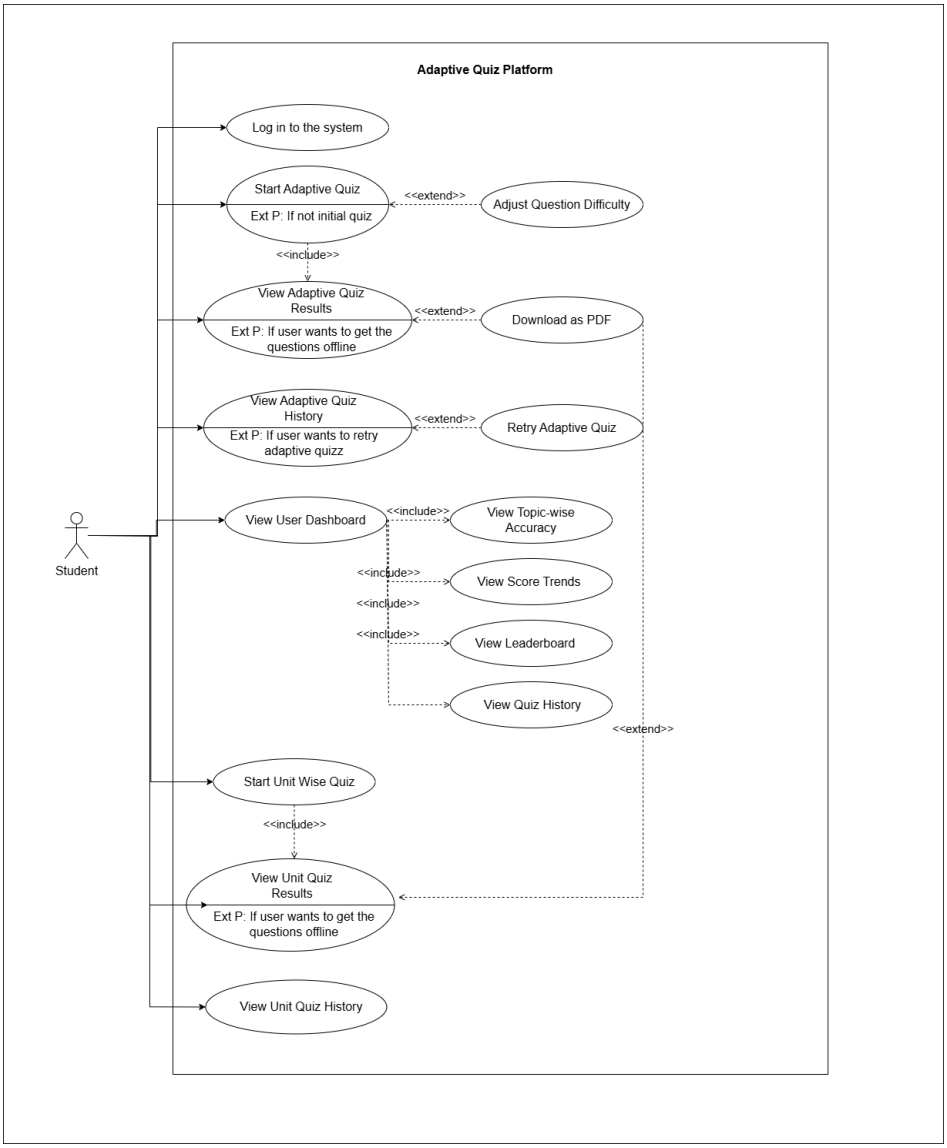


Figure 9: Use Case Diagram

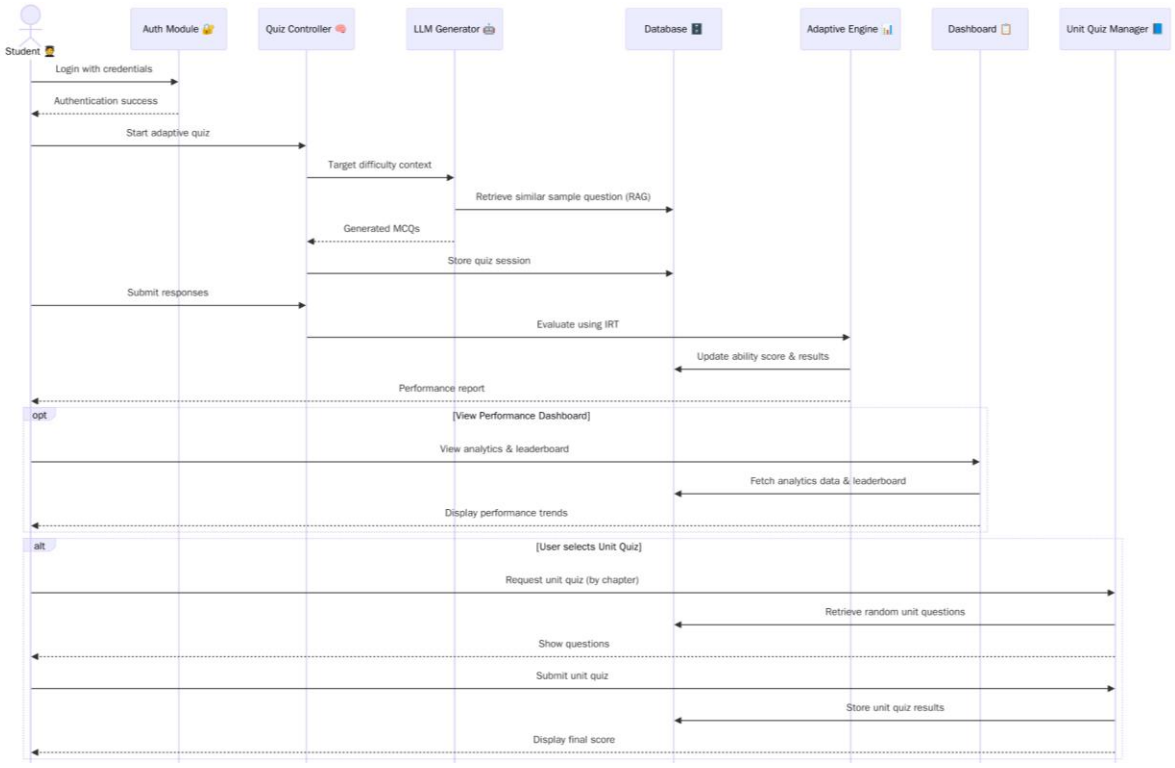


Figure 10: Sequence Diagram

2.1.2.2. Non-Functional Requirements

Non-functional requirements represent the quality attributes and operational constraints that define how the adaptive quiz platform performs under various conditions. Identifying these requirements was essential to ensuring that the system remained efficient, user-friendly, and ethically sound during real-time educational usage.

Performance and Responsiveness: While current quiz generation may involve some delay due to LLM processing, the system is expected to aim for reasonable responsiveness during key operations such as quiz generation, submission, and result display. Ensuring that users are not left waiting for extended periods is considered a critical usability goal.

Accuracy and Relevance: Since MCQs are generated using a fine-tuned LLaMA 2 model with RAG integration, the system is expected to produce syllabus-aligned and contextually appropriate questions. Maintaining question uniqueness and avoiding repetition are important requirements to ensure the relevance and usefulness of the generated content.

Usability and Accessibility: The interface is expected to be simple, clear, and user-friendly across both desktop and mobile devices. Students should be able to navigate quizzes, access results, and view dashboards without needing guidance. Features such as retry options, performance summaries, and visual analytics are considered important for usability.

Reliability and Availability: The system is expected to be consistently accessible to students without frequent downtime. Quiz attempts, performance data, and history must be reliably stored and retrievable without data loss. The platform should maintain functional stability even under multiple concurrent user sessions.

***Security and Ethics**:* Although the system does not collect sensitive personal information, it is expected to handle student quiz and performance data responsibly. Ethical use of LLM-generated content is required to ensure academic appropriateness and fairness. Additionally, no user-identifiable information should be stored or exposed during quiz generation or result handling.

Main Non-Functional Requirements Identified:

- Usability
- Performance (Speed & Responsiveness)
- Accuracy & Relevance of Generated Content
- Reliability
- Availability
- Ethical Handling of Educational Data

2.1.3 Designing

The Designing phase of this research plays a central role in shaping the intelligent structure of the adaptive quiz platform. This phase focuses on the model architecture, component interactions, and logical flows required to deliver a responsive, personalized, and syllabus-aligned assessment system for A/L Biology students. The design integrates LLM-based question generation, semantic retrieval (RAG), and IRT-driven adaptivity into a cohesive and educationally relevant system. Each module has been deliberately structured to address a specific need: contextual question generation, difficulty adjustment, targeted revision, and performance visualization.

At this point in the Software Development Life Cycle (SDLC), abstract ideas about adaptivity and personalization are translated into modular components that interact in a logically coherent architecture. The system's structure is defined with the goal of ensuring quiz variety, reducing cognitive overload, and maintaining alignment with the GCE A/L Biology syllabus. The final architecture not only supports the user journey—from login to quiz submission and performance feedback—but also outlines the internal processing layers that make these experiences intelligent and personalized.

System Architecture Diagram:

The system architecture diagram visually represents the high-level structure of the platform. The sequence begins when the student starts an adaptive quiz. The Adaptive Engine, using past quiz data, applies Item Response Theory (IRT) to estimate the student's current ability level. Based on this estimate, it selects a target difficulty range and triggers the RAG-based LLM pipeline.

The RAG module retrieves a semantically similar MCQs from dataset using FAISS and SentenceTransformers. This retrieved MCQs, along with instructional prompt formatting, is sent to a fine-tuned LLM (llama-2-7b-chat-hf), which generates a

complete quiz containing diverse but syllabus-relevant questions for calculated difficulty ratios using IRT(Item Response Theory).

Once the quiz is delivered and completed by the student, their responses are evaluated. The Adaptive Engine re-calculates ability levels using updated performance data and stores the results. The system also logs quiz metadata, including accuracy, topics covered, and performance trends, into a central data store. This information is later visualized on the student dashboard. For unit-wise quizzes, system retrieves a random set of manually categorized questions from the dataset, enabling focused revision without adaptivity.

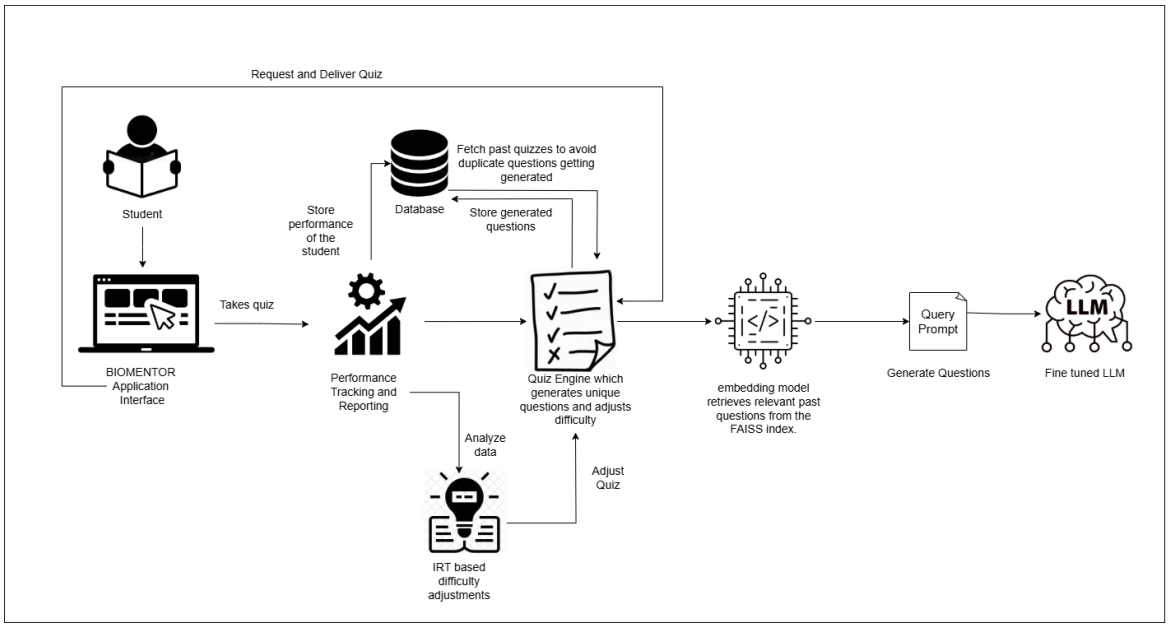


Figure 11: System Diagram

Component Architecture Overview

1. LLM-Based Question Generation (Fine-Tuned llama-2-7b-chat-hf)

This component serves as the backbone of quiz generation. The LLaMA 2 model is fine-tuned on a structured dataset of A/L Biology MCQs. During training, prompts were carefully designed to replicate A/L-style formatting, command the model to vary cognitive levels (e.g., recall, understanding, application), and adhere to syllabus themes. The model is optimized to generate a batch of questions in a single prompt pass.

2. Retrieval-Augmented Generation (RAG) Pipeline

To prevent repetitive or off-topic generation, the RAG module retrieves a semantically related example MCQ from the dataset. Retrieval is conducted using FAISS with vector embeddings generated by SentenceTransformers. The retrieved example is inserted into the LLM prompt as context, enabling the model to mimic the style and complexity of real A/L exam questions while maintaining freshness in every session.

3. Adaptive Engine (IRT-Based Logic)

Adaptivity is driven by a lightweight implementation of Item Response Theory (IRT). Each question has an associated difficulty level, and student responses are analyzed using parameters such as correctness and consistency. After each quiz, the engine updates the learner's estimated ability level (θ), which then guides the difficulty level of the next quiz.

4. Unit-Wise Quiz Retrieval System

The unit-wise quiz component allows users to select a specific chapter or topic from the Biology syllabus. When selected, the system randomly fetches a set of MCQs from a pre-classified dataset, organized manually by the external subject expert. This mode of quiz delivery is non-adaptive but supports focused revision aligned with curriculum coverage.

This dual-mode architecture (adaptive + unit-wise) gives students flexibility to learn either through guided challenge or topic-based repetition.

Flow of Fine-tuning llama-2-7b-chat-hf

Figure 10 illustrates the flow fine tuning the LLM llama-2-7b-chat-hf

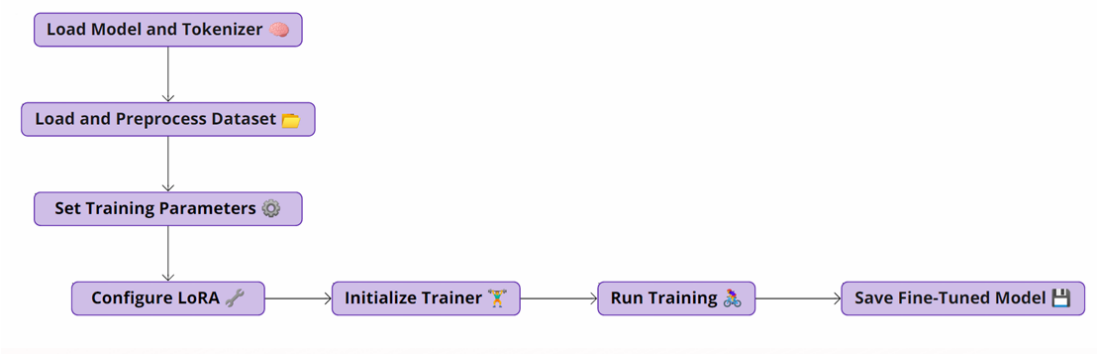


Figure 12: Flow of llama-2-7b-chat-hf fine tuning

2.1.4 Implementation

The Implementation phase of this research marked a critical step in transforming the proposed design into a fully functional adaptive quiz platform tailored for GCE A/L Biology students. This phase involved the collaborative development of all major components, including the LLM-based MCQ generation module, the adaptive engine powered by Item Response Theory (IRT), the Retrieval-Augmented Generation (RAG) pipeline, the unit-wise quiz feature, and the performance analytics dashboard. The implementation process adhered to a modular development approach to ensure clarity, maintainability, and parallel progress across different system segments.

Task Breakdown and Project Management:

To ensure systematic progress and clear visibility during development, project tasks were meticulously planned and managed using Jira, a robust project management and issue-tracking tool. The entire development process was divided into three major sprints, each spanning approximately one month. Jira boards were created to organize epics, tasks, and sub-tasks based on the project modules such as dataset preparation, model integration, adaptive logic, and user interface development.

During Sprint 1, the primary focus was on dataset collection, manual tagging of MCQs by difficulty, and implementing the basic structure of the backend. Sprint 2 was dedicated to integrating and fine-tuning the LLaMA 2 model for MCQ generation, implementing the RAG (Retrieval-Augmented Generation) pipeline, and building out the adaptive quiz logic. Sprint 3 concentrated on implementing IRT-based performance tracking, integrating dashboard analytics, refining the retry quiz feature, and performing extensive testing and optimization across all modules.

Each task in Jira was assigned to specific contributors and labelled with priority and estimated time. This structured sprint-based workflow facilitated steady progress, regular code reviews, and rapid feedback integration, ultimately contributing to the stability and completeness of the final platform. A snapshot of the Jira board and sprint planning is illustrated in Figure 12.

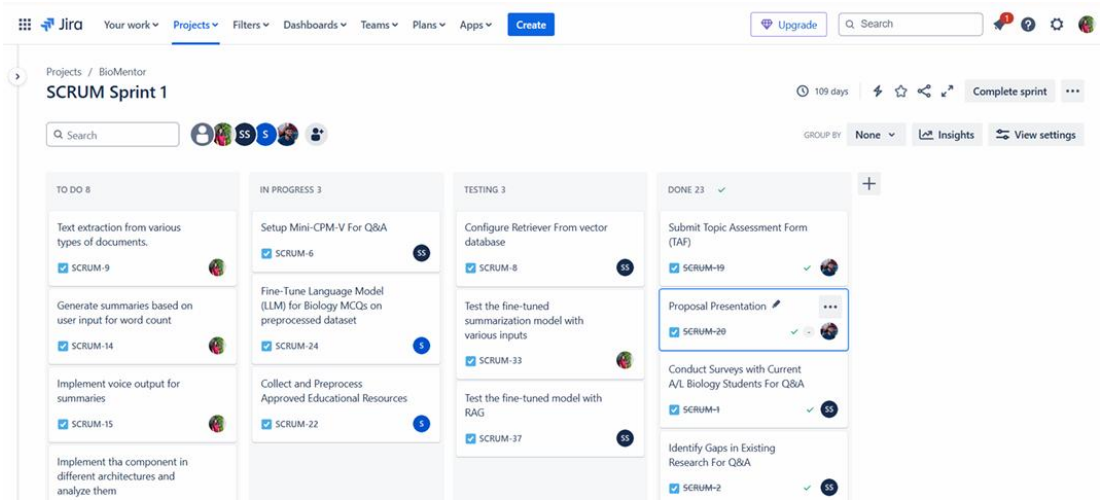


Figure 13: Jira Board

Work Breakdown Structure (WBS)

The Work Breakdown Structure (WBS) was developed to create a hierarchical view of the entire system's development effort. At the highest level, the project was divided into major modules such as Data Preparation, Model Fine-Tuning, Quiz Generation, Adaptive Engine, Unit-Wise Quiz Module, Performance Analytics, Frontend UI Development, and Integration & Testing.

Each of these modules was further broken down into smaller, more manageable subcomponents. For example, the Model Fine-Tuning module was subdivided into data formatting, prompt engineering, fine-tuning the LLaMA 2 model, and output validation. The Adaptive Engine included IRT logic development, student ability estimation, and dynamic difficulty control. Similarly, the Quiz Generation module covered semantic retrieval using FAISS, RAG-based prompt construction, and response processing.

This structured WBS enabled efficient task assignment, dependency mapping, and resource allocation throughout the development process. It also provided a clear view of progress and priorities, helping the team remain aligned with project goals and

deadlines. A visual representation of the WBS used for the adaptive quiz platform is shown in Figure 13.

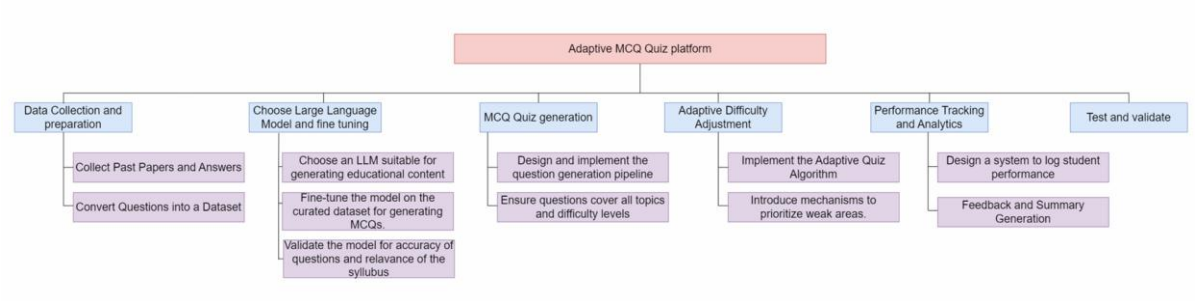


Figure 14: Work Breakdown Structure

Development Environment and Tools:

The implementation of the adaptive quiz platform was carried out using Visual Studio Code (VS Code), a lightweight yet powerful source code editor that offered flexibility and efficiency during development. With its integrated terminal, debugging tools, and extension ecosystem, VS Code provided a seamless environment for coding, testing, and managing both frontend and backend components of the system. Its support for Python and JavaScript was particularly beneficial, given the full-stack nature of the platform.

The backend development was completed using Python, with FastAPI selected as the primary web framework due to its performance, modular structure, and compatibility with asynchronous operations. FastAPI facilitated the creation of well-structured RESTful APIs responsible for key functionalities such as quiz generation, user performance tracking, and analytics delivery.

The core quiz generation logic relied on a fine-tuned LLaMA 2 (llama-2-7b-chat-hf) model, integrated through the Hugging Face Transformers library. This enabled

efficient loading and inference of the language model for dynamic MCQ generation. To support Retrieval-Augmented Generation (RAG), the system used FAISS (Facebook AI Similarity Search) for fast semantic search across a pre-processed MCQ dataset. Sentence Transformers was used in conjunction to generate embedding vectors for meaningful similarity comparisons.

The adaptive quiz engine was driven by a custom Item Response Theory (IRT) implementation, which evaluated student responses and adjusted the difficulty of future quizzes based on performance. All quiz data, performance records, and metadata were stored using a NoSQL-compatible structure, ensuring quick access and flexibility during retrieval for analytics.

On the frontend, React.js was used to build an interactive and responsive user interface. This included components for adaptive and unit-wise quizzes, result views, progress dashboards, and retry options. The frontend communicated with the backend asynchronously via REST APIs, enabling real-time data flow and smooth user interactions across different devices.

Version control was maintained using Git, with all project files and code repositories managed through GitHub. This supported team collaboration, source code tracking, and safe rollbacks during development. The entire tech stack was chosen to balance performance, scalability, and ease of development while maintaining alignment with the project's educational objectives.

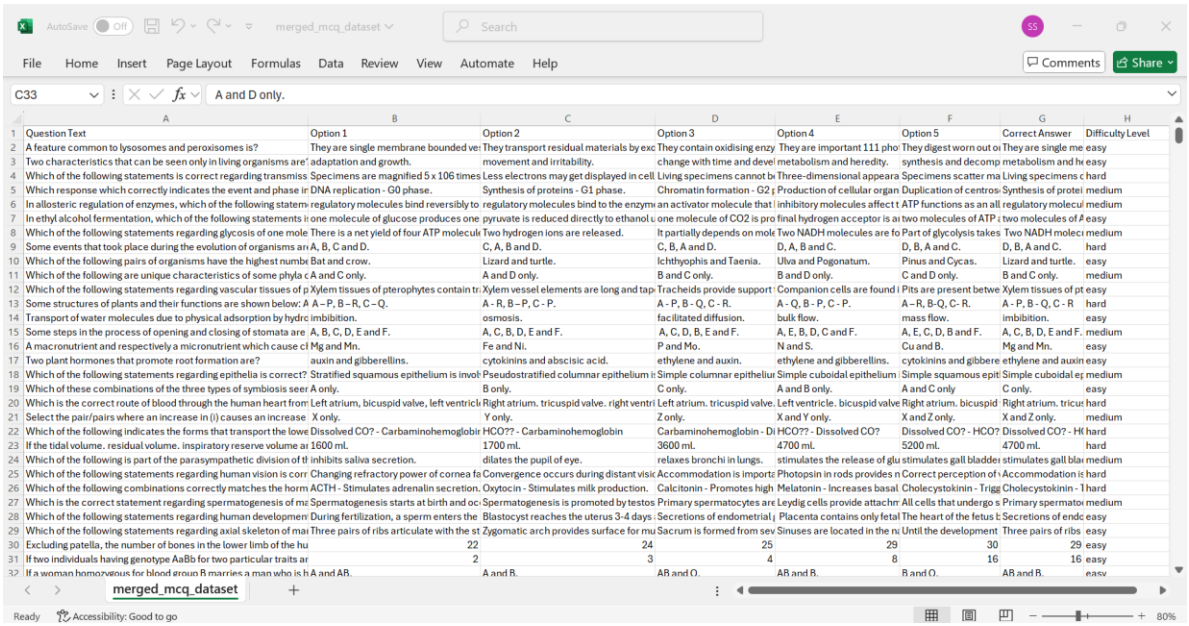
2.1.4.1 Dataset Preparation for MCQ Generation

The dataset preparation process plays a vital role in enabling the adaptive quiz system to deliver syllabus-aligned and contextually appropriate multiple-choice questions for A/L Biology students. The MCQs are collected from reliable academic sources including past GCE A/L examination papers, term test papers from schools, and published model question books commonly used by students and educators across Sri Lanka. This collection approach ensures the inclusion of diverse question formats and difficulty levels reflective of actual exam scenarios.

Each question in the dataset is organized with its correct answer and distractors (incorrect options). To make the dataset more usable for adaptive learning and topic-wise filtering, one additional metadata field is added: difficulty level. The difficulty level of each question—categorized as *easy*, *medium*, or *hard*—is determined with the support of an experienced A/L Biology teacher, who provides expert judgment based on the complexity of the question and its relevance to student comprehension levels. Similarly, questions are sorted by syllabus units, allowing the system to later group and deliver quizzes based on specific chapters.

The dataset is compiled into a structured tabular format, typically as a CSV file, with dedicated fields for the question, answer choices, and difficulty level. Preprocessing steps include formatting corrections, removal of duplicates, and ensuring linguistic clarity and consistency across questions and answers. These preparations ensure that the dataset is not only clean and accurate, but also ready for use in subsequent quiz generation modules.

By maintaining a structured and syllabus-focused dataset with meaningful metadata, this phase lays the foundation for an intelligent and responsive quiz delivery experience that caters to both adaptive and unit-based learning objectives.



	A	B	C	D	E	F	G	H
1	Question Text	Option 1	Option 2	Option 3	Option 4	Option 5	Correct Answer	Difficulty Level
2	A feature common to lysosomes and peroxisomes is?	They are single membrane bounded ves	They transport residual materials by exo	They contain oxidising enzy	They are important 111 phot	They digest worn out or	They are single me	easy
3	Two characteristics that can be seen only in living organisms are?	adaptation and growth.	movement and irritability.	change with time and devel	metabolism and heredity.	synthesis and decomp	metabolism and he	easy
4	Which of the following statements is correct regarding transmiss	Specimens are magnified 5 x 106 times	Less electrons may get displayed in cell	Living specimens cannot be	Three-dimensional appeara	Specimens scatter ma	Living specimens c	hard
5	Which response which correctly indicates the event and phase in	DNA replication - G0 phase.	Synthesis of proteins - G1 phase.	Chromatin formation - G2 p	Production of cellular organ	Duplication of centroso	Synthesis of protei	medium
6	In allosteric regulation of enzymes, which of the following statem	regulatory molecules bind reversibly to	regulatory molecules bind to the enzym	an activator molecule that i	inhibitory molecules affect t	ATP functions as an all	regulatory molecul	medium
7	In ethyl alcohol fermentation, which of the following statements is	one molecule of glucose produces one	pyruvate is reduced directly to ethanol	one molecule of CO2 is pro	final hydrogen acceptor is ac	two molecules of ATP	two molecules of A	easy
8	Which of the following statements regarding glycosis of one mole	There is a net yield of four ATP molecul	Two hydrogen ions are released.	It partially depends on mole	Two NADH molecules are fo	Part of glycolysis takes	Two NADH molecu	medium
9	Some events that took place during the evolution of organisms are	A, B, C and D.	C, A, B and D.	C, B, A and D.	D, A, B and C.	D, B, A and C.	D, B, A and C.	hard
10	Which of the following pairs of organisms have the highest numbe	Bat and crow.	Lizard and turtle.	Ichthyophis and Taenia.	Ulva and Pogonatum.	Pinus and Cycas.	Lizard and turtle.	easy
11	Which of the following are unique characteristics of some phyla c	A and C only.	A and D only.	B and C only.	B and D only.	C and D only.	B and C only.	medium
12	Which of the following statements regarding vascular tissues of p	Xylem tissues of pterophytes contain tr	Xylem vessel elements are long and tap	Tracheids provide support	Companion cells are found i	Pits are present betwe	Xylem tissues of pt	easy
13	Some structures of plants and their functions are shown below: A	A – P, B – R, C – Q.	A - R, B – P, C - P.	A - P, B - Q, C - R.	A - Q, B - P, C - P.	A – R, B-Q, C- R.	A - P, B - Q, C - R	hard
14	Transport of water molecules due to physical adsorption by hydro	imbibition.	osmosis.	facilitated diffusion.	bulk flow.	mass flow.	imbibition.	easy
15	Some steps in the process of opening and closing of stomata are	A, B, C, D, E and F.	A, C, B, D, E and F.	A, C, D, B, E and F.	A, E, B, D, C and F.	A, E, C, D, B and F.	A, C, B, D, E and F.	medium
16	A macronutrient and respectively a micronutrient which cause cl	Mg and Mn.	Fe and Ni.	P and Mo.	N and S.	Cu and B.	Mg and Mn.	easy
17	Two plant hormones that promote root formation are?	auxin and gibberellins.	cytokinins and abscisic acid.	ethylene and auxin.	ethylene and gibberellins.	cytokinins and gibbere	ethylene and auxin	easy
18	Which of the following statements regarding epithelia is correct?	Stratified squamous epithelium is invol	Pseudostratified columnar epithelium is	Simple columnar epitheliur	Simple cuboidal epithelium	Simple squamous epit	Simple cuboidal ep	medium
19	Which of these combinations of the three types of symbiosis seer	A only.	B only.	C only.	A and B only.	A and C only	C only.	easy
20	Which is the correct route of blood through the human heart from	Left atrium, bicuspid valve, left ventricle	Right atrium. tricuspid valve. right ventri	Left atrium. tricuspid valve.	Left ventricle. bicuspid valve	Right atrium. bicuspid	Right atrium. tricu	hard
21	Select the pair/pairs where an increase in (i) causes an increase	X only.	Y only.	Z only.	X and Y only.	X and Z only.	X and Z only.	medium
22	Which of the following indicates the forms that transport the lowe	Dissolved CO? - Carbaminohemoglobin	HCO?? - Carbaminohemoglobin	Carbaminohemoglobin - Di	HCO?? - Dissolved CO?	Dissolved CO? - HCO?	Dissolved CO? - HC	hard
23	If the tidal volume. residual volume. inspiratory reserve volume ar	1600 ml.	1700 ml.	3600 ml.	4700 ml.	5200 ml.	4700 ml.	hard
24	Which of the following is part of the parasympathetic division of th	inhibits saliva secretion.	dilates the pupil of eye.	relaxes bronchi in lungs.	stimulates the release of glu	stimulates gall bladder	stimulates gall bla	medium
25	Which of the following statements regarding human vision is corr	Changing refractory power of cornea fa	Convergence occurs during distant visic	Accommodation is importa	Photopsin in rods provides n	Correct perception of v	Accommodation is	hard
26	Which of the following combinations correctly matches the horm	ACTH - Stimulates adrenalin secretion.	Oxytocin - Stimulates milk production.	Calcitonin - Promotes high	Melatonin - Increases basal	Cholecystokinin - Trigg	Cholecystokinin - T	hard
27	Which is the correct statement regarding spermatogenesis of ma	Spermatogenesis starts at birth and oc	Spermatogenesis is promoted by testos	Primary spermatocytes are	Leydig cells provide attachm	All cells that undergo s	Primary spermato	medium
28	Which of the following statements regarding human developmen	During fertilization, a sperm enters the	Blastocyst reaches the uterus 3-4 days a	Secretions of endometrial g	Placenta contains only fetal	The heart of the fetus b	Secretions of endo	easy
29	Which of the following statements regarding axial skeleton of mai	Three pairs of ribs articulate with the st	Zygomatic arch provides surface for mu	Sacrum is formed from sev	Sinuses are located in the ne	Until the development	Three pairs of ribs	easy
30	Excluding patella, the number of bones in the lower limb of the hu	22	24	25	29	30	29	easy
31	If two individuals having genotype AaBb for two particular traits ar	2	3	4	8	16	16	easy
32	If a woman homozygous for blood group B marries a man who is h	A and B.	AB and O.	AB and B.	B and O.	AB and B.	easy	

Figure 15: A sample section of the curated A/L Biology MCQ dataset showing difficulty classification.

2.1.4.2 Data Preprocessing and Cleaning

Before training the model, the collected dataset of multiple-choice questions undergoes essential preprocessing and cleaning to ensure consistency, completeness, and usability for fine-tuning. The original dataset, compiled from A/L past papers and school resources, is organized with fields such as question text, multiple answer options, correct answer, difficulty level, and syllabus unit.

The preprocessing phase begins by loading the dataset in CSV format using the Pandas library. Duplicate questions and records with missing fields are filtered out to reduce noise and redundancy in the training data. Each valid question is then transformed into a unified prompt format suitable for training a language model. This involves structuring the prompt to include the question, labelled answer options (e.g., A., B., C., etc.), the difficulty tag, and a standard instruction: *"Choose the correct answer."*

The preprocessing function extracts each row, combines the options into a single string, appends the difficulty label, and formats the question into a full instruction-style prompt. This formatted data is then stored in a new DataFrame and converted into a **Hugging Face Dataset** format to be compatible with the Transformers and Trainer APIs used later in model training.

This cleaned and structured data is now ready to be passed into the fine-tuning process. It ensures that the language model receives inputs in a consistent, syllabus-aligned format that reflects real-world exam scenarios, allowing it to learn patterns for question structure and difficulty-specific phrasing.

```python
# Load and preprocess your CSV dataset
df = pd.read_csv(file_path, encoding='ISO-8859-1')

# Define a preprocessing function to format the questions, options, and difficulty
def preprocess_data(row):
    question = row["Question Text"]
    options = [row[f"Option {i}"] for i in range(1, 6) if pd.notna(row[f"Option {i}"])]
    correct_answer = row["Correct Answer"]
    difficulty = row["Difficulty Level"]

    # Format question prompt for training
    options_text = " ".join([f"{chr(65+i)}. {opt.strip()}" for i, opt in enumerate(options)])
    prompt = f"Question ({difficulty}): {question}\nOptions: {options_text}\nChoose the correct answer:"

    return {
        "text": prompt,
        "label": correct_answer
    }

# Apply preprocessing to the dataset and convert to DataFrame
processed_data = df.apply(preprocess_data, axis=1).tolist()
processed_df = pd.DataFrame(processed_data)

# Convert the preprocessed DataFrame into a Hugging Face Dataset format
dataset = Dataset.from_pandas(processed_df)
```

Figure 16: Code snippet showing the preprocessing function that formats MCQs into training prompts.

2.1.4.3 Model Selection and Fine-Tuning

The model selection and fine-tuning phase is a core component of this research, as it enables the system to generate syllabus-aligned, difficulty-specific multiple-choice questions for A/L Biology students. The goal of this phase is to adapt a powerful pre-trained language model to understand the structure, tone, and formatting style of real exam questions, and to generate new ones accordingly.

Model Selection

After evaluating various open-source large language models suitable for instruction-following tasks, the llama-2-7b-chat-hf model was selected. This model, developed by Meta and released via Hugging Face, is optimized for dialogue and structured response generation, making it a strong candidate for MCQ-style content creation. Its instruction-tuned architecture supports generating coherent and contextually relevant outputs when provided with well-structured prompts.

The model is chosen for its performance balance between quality and computational requirements. Its open access and compatibility with fine-tuning libraries such as Transformers, PEFT, and TRL also make it ideal for integration in this educational context.

Training Environment Setup

Fine-tuning was conducted using Google Colab Pro, which offers high-performance GPUs for extended sessions, allowing the full model to be loaded using 4-bit quantization via BitsAndBytes. Libraries such as accelerate, peft, trl, and transformers were installed to support efficient and memory-optimized training.

The model was loaded with quantization configurations to support QLoRA (Quantized Low-Rank Adaptation), enabling the training of specific layers while freezing the base model to reduce memory and time requirements.

Prompt Formatting and Data Alignment

Each question in the dataset is converted into an instruction-style prompt that follows a standard structure. The prompt includes the difficulty level, question, and answer options, followed by the instruction “Choose the correct answer.” This ensures consistency and encourages the model to learn the format and tone used in real A/L question papers.

LoRA Configuration and Training Setup

Low-Rank Adaptation (LoRA) parameters were configured to update a small subset of the model’s weights efficiently. The model was trained for one epoch due to resource limitations, with training arguments including optimizer settings, learning rate, batch size, gradient accumulation, and scheduler type. Training was executed using the SFTTrainer from the trl library, which streamlines supervised fine-tuning on textual instructions.

The training process was monitored using logging intervals and evaluated for output consistency across difficulty levels and topics. Once fine-tuned, the model was saved locally and evaluated using test prompts to ensure that it could generate valid and meaningful MCQs.

```python
# Configure bitsandbytes
compute_dtype = getattr(torch, bnb_4bit_compute_dtype)
bnb_config = BitsAndBytesConfig(
    load_in_4bit=use_4bit,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=use_nested_quant,
)

# GPU compatibility check for bfloat16
if compute_dtype == torch.float16 and use_4bit:
    major, _ = torch.cuda.get_device_capability()
    if major >= 8:
        print("=" * 80)
        print("Your GPU supports bfloat16: accelerate training with bf16=True")
        print("=" * 80)

# Load model and tokenizer
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    device_map=device_map
)
model.config.use_cache = False
model.config.pretraining_tp = 1

# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"

# LoRA configuration
peft_config = LoraConfig(
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    r=lora_r,
    bias="none",
    task_type="CAUSAL_LM",
)
```

Figure 17: Quantization and LoRA configurations used to optimize model training.

```python
# Set training parameters
training_arguments = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=num_train_epochs,
    per_device_train_batch_size=per_device_train_batch_size,
    gradient_accumulation_steps=gradient_accumulation_steps,
    optim=optim,
    save_steps=save_steps,
    logging_steps=logging_steps,
    learning_rate=learning_rate,
    weight_decay=weight_decay,
    fp16=fp16,
    bf16=bf16,
    max_grad_norm=max_grad_norm,
    max_steps=max_steps,
    warmup_ratio=warmup_ratio,
    group_by_length=group_by_length,
    lr_scheduler_type=lr_scheduler_type,
    report_to="tensorboard"
)

# Initialize the trainer with the processed dataset and LoRA configuration
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    dataset_text_field="text",
    max_seq_length=None,  # Or specify max sequence length if needed
    tokenizer=tokenizer,
    args=training_arguments,
    packing=False,
)

# Train the model
trainer.train()
```

Figure 18: Training parameters and supervised fine-tuning setup using SFTTrainer.

2.1.4.4 Embedding Generation and Clustering

Before generating or retrieving questions for a quiz, the system must understand the semantic meaning of each MCQ in the dataset — not just its words, but the concept it tests. To achieve this, the system uses a process called embedding generation, where each question is transformed into a numerical vector that captures its meaning in a mathematical space. This is similar to giving each question a unique "fingerprint" based on what it asks, regardless of wording.

To generate these embeddings, the platform uses a SentenceTransformer model. This model converts each MCQ into a dense vector of fixed dimensions. These vectors act like coordinates on a conceptual map of questions — questions with similar meanings are placed close together, while very different ones are far apart.

Once all the MCQs are embedded, the system applies clustering, using the KMeans algorithm, to group semantically similar questions into clusters. The system shelves them based on the *concepts* they cover. Questions related to “photosynthesis” might fall into one cluster, while those testing “genetics” fall into another.

These clusters are then used in various downstream components. For instance, the quiz engine can sample diverse questions from different clusters to avoid repetition and enhance topic coverage. Clustering also plays a role in unit-wise question selection and in controlling quiz diversity during adaptive delivery.

The embedding vectors are indexed using FAISS (Facebook AI Similarity Search), enabling fast retrieval of semantically similar questions. This combination of vectorization and clustering enhances the system’s ability to generate, filter, and retrieve high-quality, contextually relevant questions.

```python
from sentence_transformers import SentenceTransformer
import pandas as pd
import numpy as np
import faiss
from sklearn.cluster import KMeans

# Load original dataset
dataset = pd.read_csv("merged_mcq_dataset.csv", encoding="latin1").fillna("")

# Ensure required columns exist
required_columns = ["Question Text", "Option 1", "Option 2", "Option 3", "Option 4", "Option 5",
"Correct Answer", "Difficulty Level"]
for col in required_columns:
    if col not in dataset.columns:
        raise ValueError(f"Missing column in dataset: {col}")

# Combine question, options, and correct answer for better context representation
dataset["Combined"] = (
    dataset["Question Text"] + " " +
    dataset["Option 1"] + " " +
    dataset["Option 2"] + " " +
    dataset["Option 3"] + " " +
    dataset["Option 4"] + " " +
    dataset["Option 5"] + " " +
    "Correct Answer: " + dataset["Correct Answer"]
)

# Load Sentence Transformer model
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")

# Generate embeddings
dataset["embeddings"] = dataset["Combined"].apply(lambda x: embedding_model.encode(x).tolist())

# Convert embeddings to NumPy array
embeddings_matrix = np.array(dataset["embeddings"].tolist()).astype("float32")

# Save dataset with embeddings **including Difficulty Level (as separate column)**
dataset.to_csv("question_dataset_with_embeddings.csv", index=False)

# Create FAISS index
index = faiss.IndexFlatL2(embeddings_matrix.shape[1])
index.add(embeddings_matrix)

# Save FAISS index
faiss.write_index(index, "question_embeddings.index")
np.save("question_embeddings.npy", embeddings_matrix)

# **Topic-Based Clustering** (Ensures diverse question selection)
NUM_CLUSTERS = 10
kmeans = KMeans(n_clusters=NUM_CLUSTERS, random_state=42, n_init=10)
dataset["Cluster"] = kmeans.fit_predict(embeddings_matrix)

# Save dataset with Clusters & Difficulty Level for retrieval
dataset.to_csv("question_dataset_with_clusters.csv", index=False)

print(" Embeddings, FAISS index, and Topic Clusters created successfully!")
```

Figure 19: Generating semantic embeddings using SentenceTransformer and clustering them into groups using KMeans.

2.1.4.5 IRT-Based Ability Estimation and Scaling

To create a quiz system that intelligently adapts to individual student performance, this platform incorporates **Item Response Theory (IRT)** to estimate each learner's ability level and adjust future quiz difficulty accordingly. IRT provides a statistical approach to modeling a student's proficiency (θ) based on how they perform on questions of varying complexity.

Each question is represented using three core IRT parameters:

- **a (discrimination):** Reflects how well a question differentiates between students of different ability levels.
- **b (difficulty):** Indicates how challenging the question is.
- **c (guessing):** Represents the probability of answering the question correctly by guessing.

These parameters are systematically assigned based on each question's tagged difficulty level. For example, questions labelled as "easy" have a lower b value and higher c value, while "hard" questions have higher b values and are less likely to be guessed correctly.

The probability of a student correctly answering a question is estimated using the **Three-Parameter Logistic Model**, defined as:

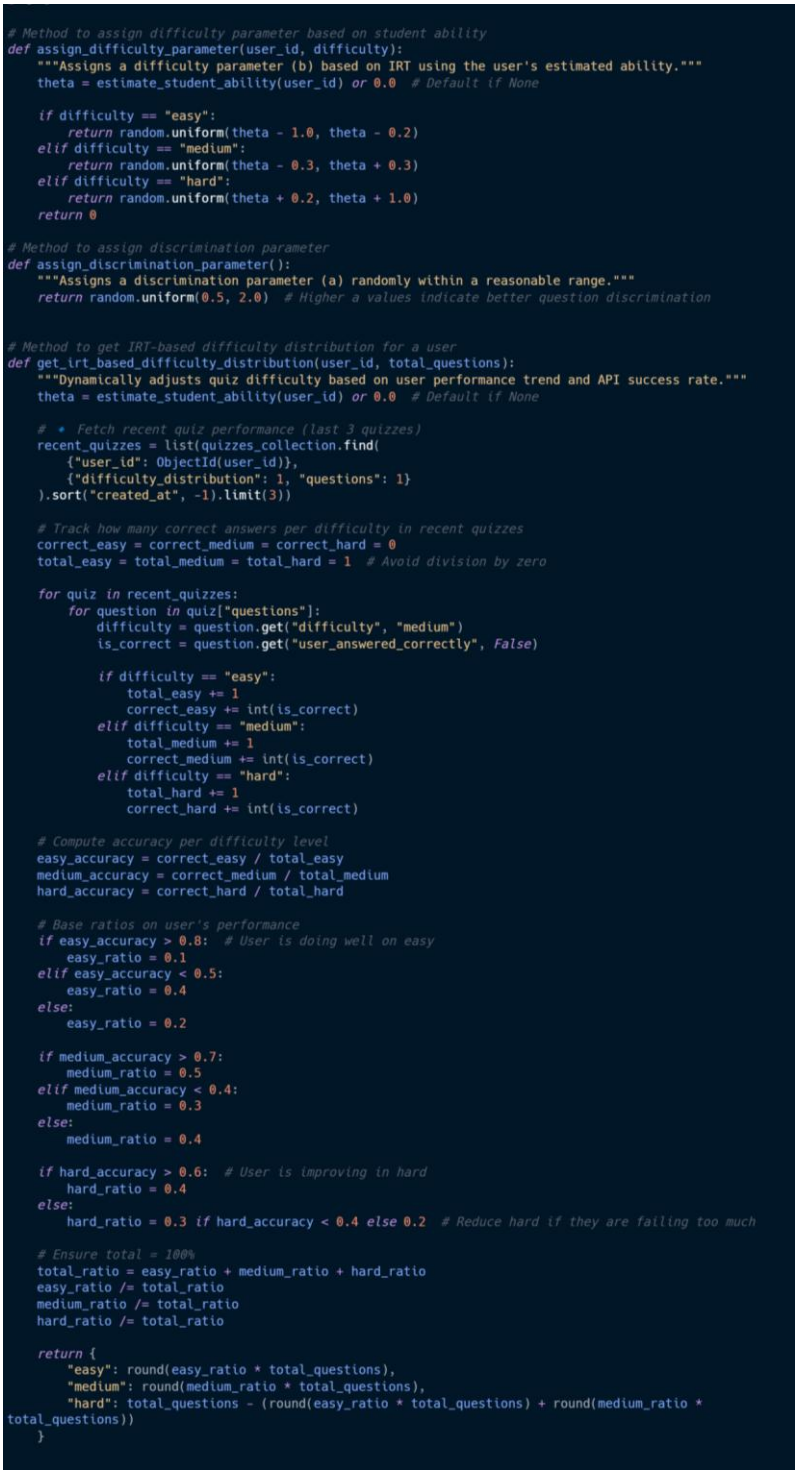
$$P(\theta) = c + \frac{1 - c}{1 + e^{-a(\theta - b)}}$$

Here, θ denotes the student's current ability estimate, and the function predicts their likelihood of answering a question correctly, given its difficulty, discrimination power, and guessing chance.

Once a student completes a quiz, their responses are compared against these theoretical probabilities. The system then updates their θ score using a scoring mechanism that rewards high performance on difficult questions and penalizes incorrect answers on easier ones. This continuous recalibration enables the system to better understand each learner's current level and refine the challenge accordingly.

In subsequent quizzes, the updated ability score is used to dynamically allocate a new set of questions based on a targeted distribution of difficulty levels. For instance, a student with a higher θ may receive a quiz with more hard questions, while a student still developing mastery might receive a greater proportion of easy and medium ones.

This approach transforms the quiz platform into a responsive, evolving environment that supports progressive learning. It balances student engagement and challenge, ensuring that learners remain motivated while advancing at a pace suited to their individual capabilities.



```python
# Method to assign difficulty parameter based on student ability
def assign_difficulty_parameter(user_id, difficulty):
    """Assigns a difficulty parameter (b) based on IRT using the user's estimated ability."""
    theta = estimate_student_ability(user_id) or 0.0  # Default if None

    if difficulty == "easy":
        return random.uniform(theta - 1.0, theta - 0.2)
    elif difficulty == "medium":
        return random.uniform(theta - 0.3, theta + 0.3)
    elif difficulty == "hard":
        return random.uniform(theta + 0.2, theta + 1.0)
    return 0

# Method to assign discrimination parameter
def assign_discrimination_parameter():
    """Assigns a discrimination parameter (a) randomly within a reasonable range."""
    return random.uniform(0.5, 2.0)  # Higher a values indicate better question discrimination


# Method to get IRT-based difficulty distribution for a user
def get_irt_based_difficulty_distribution(user_id, total_questions):
    """Dynamically adjusts quiz difficulty based on user performance trend and API success rate."""
    theta = estimate_student_ability(user_id) or 0.0  # Default if None

    # * Fetch recent quiz performance (last 3 quizzes)
    recent_quizzes = list(quizzes_collection.find(
        {"user_id": ObjectId(user_id)},
        {"difficulty_distribution": 1, "questions": 1}
    ).sort("created_at", -1).limit(3))

    # Track how many correct answers per difficulty in recent quizzes
    correct_easy = correct_medium = correct_hard = 0
    total_easy = total_medium = total_hard = 1  # Avoid division by zero

    for quiz in recent_quizzes:
        for question in quiz["questions"]:
            difficulty = question.get("difficulty", "medium")
            is_correct = question.get("user_answered_correctly", False)

            if difficulty == "easy":
                total_easy += 1
                correct_easy += int(is_correct)
            elif difficulty == "medium":
                total_medium += 1
                correct_medium += int(is_correct)
            elif difficulty == "hard":
                total_hard += 1
                correct_hard += int(is_correct)

    # Compute accuracy per difficulty level
    easy_accuracy = correct_easy / total_easy
    medium_accuracy = correct_medium / total_medium
    hard_accuracy = correct_hard / total_hard

    # Base ratios on user's performance
    if easy_accuracy > 0.8:  # User is doing well on easy
        easy_ratio = 0.1
    elif easy_accuracy < 0.5:
        easy_ratio = 0.4
    else:
        easy_ratio = 0.2

    if medium_accuracy > 0.7:
        medium_ratio = 0.5
    elif medium_accuracy < 0.4:
        medium_ratio = 0.3
    else:
        medium_ratio = 0.4

    if hard_accuracy > 0.6:  # User is improving in hard
        hard_ratio = 0.4
    else:
        hard_ratio = 0.3 if hard_accuracy < 0.4 else 0.2  # Reduce hard if they are failing too much

    # Ensure total = 100%
    total_ratio = easy_ratio + medium_ratio + hard_ratio
    easy_ratio /= total_ratio
    medium_ratio /= total_ratio
    hard_ratio /= total_ratio

    return {
        "easy": round(easy_ratio * total_questions),
        "medium": round(medium_ratio * total_questions),
        "hard": total_questions - (round(easy_ratio * total_questions) + round(medium_ratio *
total_questions))
    }
```

Figure 20: Functions used to assign difficulty and discrimination parameters and scale quiz difficulty based on student ability (θ).

2.1.4.6 Adaptive Quiz Engine with Difficulty Control

The adaptive quiz engine is one of the most critical backend components in the system. It is responsible for generating personalized, syllabus-aligned quizzes that evolve in difficulty based on each student's ongoing performance. This engine orchestrates a multi-step process that begins with quiz initiation, leverages advanced question generation techniques, and delivers a complete quiz back to the user in real-time.

1. **Initial Quiz Delivery (Non-Adaptive First Attempt)**

   When a new user starts the quiz for the first time, the system does not yet have enough performance history to calculate their ability level. Therefore, the platform generates a baseline quiz using a default difficulty ratio. This quiz serves two purposes: to familiarize the user with the format and to capture initial performance data that will be used to seed the adaptive algorithm for future quizzes.

2. **Difficulty Determination via Ability Score**

   Once the user has completed their first quiz, their responses are evaluated, and an initial ability score is calculated using Item Response Theory (IRT). For all subsequent quizzes, this score becomes the basis for selecting the difficulty level of questions. The system maps the student’s current ability to a distribution of easy, medium, and hard questions. For instance, a user with a moderate ability may receive a balanced mix, while a high-performing user may be given more challenging content.

3. **Semantic Context Filtering**

   To initiate adaptive quiz generation, the system selects a contextual MCQ from the pre-tagged question dataset. This selection is done using FAISS, a fast similarity search library, paired with SentenceTransformers. The engine first determines the intended difficulty range, then retrieves a semantically similar question that falls within that difficulty band and has not been recently used for the same student. This context MCQ

is crucial — it is passed into the model as an example to guide the structure and domain of the questions to be generated.

This step enhances the quality and relevance of the LLM's output, anchoring it to real, human-created questions from past A/L papers or school tests.

4. **Prompt Construction and Question Generation**

After identifying the context, the backend constructs a prompt formatted for the fine-tuned LLaMA 2 model (llama-2-7b-chat-hf). The prompt includes the syllabus topic, difficulty level, and example question, followed by an instruction for the model to generate new, similar questions. This is carefully structured to produce multiple MCQs in a single response, formatted consistently and syllabus-aligned.

The prompt is then passed into the LLM, which returns a batch of new questions. These are not tagged, as the system already controls the difficulty through IRT-based selection and semantic filtering prior to generation.

5. **Duplicate Filtering and Cleanup**

To ensure question variety and avoid repetition, the engine performs a duplicate filtering step immediately after receiving the output. It compares each newly generated question to previously attempted questions by the same student using semantic similarity. Any question deemed too similar is discarded or replaced to maintain freshness.

The valid questions are then cleaned, parsed into structured format (question, options, correct answer), and bundled into a quiz object.

6. **Quiz Delivery and Response Logging**

Finally, the structured quiz is returned via API to the frontend interface. The system also creates a session record in the database containing metadata such as the quiz ID, timestamp, difficulty level, question content, and the user's ID. This enables the quiz to be accurately linked to a user's performance history and used for analytics or retries.

When the student completes the quiz, their responses are submitted through a separate API, where scoring and ability score updates are handled. This closes the feedback loop performance on the current quiz influences the content and difficulty of the next.

```python
import re

def extract_mcqs(prompt, raw_output):
    raw_output = raw_output.replace(prompt, "").strip()
    lines = [line.strip() for line in raw_output.split("\n") if line.strip()]

    mcqs = []
    current_mcq = {"question": None, "options": {}, "correct_answer": None}
    waiting_for_question = False

    def flush():
        if (
            current_mcq["question"]
            and len(current_mcq["options"]) == 5
            and current_mcq["correct_answer"] in current_mcq["options"]
        ):
            current_mcq["question"] = re.sub(
                r"^(?:Question\s*)?\d+[\.\:\)]\s*", "", current_mcq["question"]
            ).strip(" .:")
            mcqs.append(current_mcq.copy())

    for i, line in enumerate(lines):
        # Section headers like "Example:", "Easy 1:", etc.
        if re.match(r"^(Example|Easy|Medium|Hard)?\s*\d*[\:\)]\s*$", line, re.IGNORECASE) or
line.lower() in ["example:", "example"]:
            flush()
            current_mcq = {"question": None, "options": {}, "correct_answer": None}
            waiting_for_question = True
            continue

        # Lines like "1:" or "Question 1:"
        if re.match(r"^(?:Question\s*)?\d+[\.\:\)]\s*$", line, re.IGNORECASE):
            flush()
            current_mcq = {"question": None, "options": {}, "correct_answer": None}
            waiting_for_question = True
            continue

        # Waiting for the question after a header
        if waiting_for_question:
            question = re.sub(r"^(?:Question\s*)?\d+[\.\:\)]\s*", "", line.strip(" ?:")).strip(" .:")
            if question:
                current_mcq["question"] = question
                waiting_for_question = False
            continue

        # Inline: "Question 1: What is...?"
        q_match = re.match(r"^Question\s*[:\-]?\s*(.+)", line, re.IGNORECASE)
        if not q_match:
            q_match = re.match(r"^(?:Question\s*)?\d+[\.\:\)]\s*(.+)", line, re.IGNORECASE)
        if q_match:
            flush()
            question = re.sub(r"^(?:Question\s*)?\d+[\.\:\)]\s*", "", q_match.group(1).strip("
?:")).strip(" .:")
            current_mcq = {"question": question, "options": {}, "correct_answer": None}
            continue

        # Bullet-style questions: "- What is...?"
        bullet_q_match = re.match(r"^\-\s*(.+)", line)
        if bullet_q_match and not current_mcq["question"]:
            text = bullet_q_match.group(1).strip(" ?:")
            if not re.match(r"^[A-Ea-e][\)\.\:\-]?\s+", text):
                flush()
                question = re.sub(r"^(?:Question\s*)?\d+[\.\:\)]\s*", "", text).strip(" .:")
                current_mcq = {"question": question, "options": {}, "correct_answer": None}
                continue

        # Option A–E: "A) text"
        opt_match = re.match(r"^([A-Ea-e])[\)\.\:\-]?\s+(.+)", line)
        if opt_match:
            if not current_mcq["question"]:
                # Backtrack to find question
                for j in range(i - 1, -1, -1):
                    prev = lines[j]
                    if not re.match(r"^[A-Ea-e][\)\.\:\-]?\s+", prev) and not
prev.lower().startswith("correct answer"):
                        question = re.sub(r"^(?:Question\s*)?\d+[\.\:\)]\s*", "", prev.strip("
?:")).strip(" .:")
                        if question:
                            current_mcq["question"] = question
                        break
            current_mcq["options"][opt_match.group(1).upper()] = opt_match.group(2).strip()
            continue

        # Numbered options: "(1)", "1.", "1)", etc.
        num_opt_match = re.match(r"^\(?([1-5])\)?[\.\:\-]?\s+(.+)", line)
        if num_opt_match:
            number_to_letter = {"1": "A", "2": "B", "3": "C", "4": "D", "5": "E"}
            letter = number_to_letter.get(num_opt_match.group(1))
            if letter:
                current_mcq["options"][letter] = num_opt_match.group(2).strip()
            continue

        # Answers: "Correct Answer: C" or "Correct Answer: (3)"
        ans_match = re.match(r"^(?:Correct\s*)?Answer\s*[:\-]?\s*\(?([A-Ea-e1-5])\)?(?:[\)\.\:\-]?
\s+.*)?$", line, re.IGNORECASE)
        if ans_match:
            val = ans_match.group(1).upper()
            if val in "12345":
                val = chr(64 + int(val))  # Convert 1–5 to A–E
            current_mcq["correct_answer"] = val
            continue

    flush()
    return mcqs
```

Figure 22: Function to extract and structure multiple-choice questions from the raw output returned by the LLaMA 2 model.

```
def generate_mcq_based_on_performance(user_id, difficulty, max_retries=5, existing_questions=None,
past_embeddings=None):
    """Generate up to 3 MCQs based on user's performance, minimizing retries by accepting partial
results."""
    retries = 0
    batch_generated_questions = set()
    valid_mcqs = []
    existing_questions = existing_questions or set()

    theta = estimate_student_ability(user_id) or 0.0

    while retries < max_retries and len(valid_mcqs) < 3:
        try:
            if dataset.empty:
                logging.error("Dataset is empty. Cannot generate MCQs.")
                return valid_mcqs

            sampled_question = dataset.groupby("Cluster").sample(1)

            if sampled_question.empty:
                logging.error("ERROR: No questions available in dataset. Retrying...")
                retries += 1
                continue

            random_question = sampled_question.iloc[0]["Question Text"]
            context_questions = retrieve_context_questions(random_question, top_k=3)

            context_list = [
                f"- {row['Question Text']} (Correct Answer: {row['Correct Answer']})"
                for _, row in context_questions.iterrows()
            ] if not context_questions.empty else []

            remaining = 3 - len(valid_mcqs)
            prompt = f"""
            Generate {remaining} **{difficulty}** level multiple-choice biology question{'s' if
remaining > 1 else ''}.

            **User's Estimated Ability Level (IRT Theta):** {theta}

            Each question must follow this format:

            Question 1: <Insert your question>
            A) <Option A>
            B) <Option B>
            C) <Option C>
            D) <Option D>
            E) <Option E>
            Correct Answer: <A/B/C/D/E>

            Do not include explanations, numbering, answer keys, or extra text.
            """

            if context_list:
                context_block = "\n".join(context_list)
                prompt = f"""
                Generate a **{difficulty}** level MCQ that is **different** from these:

                {context_block}

                **Follow This Format:**
                {prompt}
                """

            response = requests.post(COLAB_API_URL, json={"prompt": prompt}, timeout=30)
            data = response.json()
            raw_output = data.get("raw_output", "")
            logging.warning(f"\u26a0 RAW API RESPONSE: {raw_output}")

            extracted_mcqs = extract_mcqs(prompt, raw_output)

            for mcq in extracted_mcqs:
                question = mcq.get("question", "").strip()
                options = mcq.get("options", {})
                correct_letters = clean_correct_answer(mcq.get("correct_answer", ""))
                mcq["correct_answer"] = ", ".join(correct_letters)

                if any([
                    not question,
                    len(options) != 5,
                    any(not v.strip() for v in options.values()),
                    len(set(options.values())) < 5,
                    not correct_letters,
                    any(c not in options for c in correct_letters),
                    is_duplicate_faiss(question, index, 0.85),
                    is_similar_to_same_quiz_questions(question, batch_generated_questions,
threshold=0.85),
                    question in batch_generated_questions,
                    question in existing_questions,
                    "error" in question,
                    "<Insert your question>" in question,
                    "Generate a" in question
                ]):
                    continue

                if past_embeddings is not None:
                    new_vec = embedding_model.encode([question]).astype(np.float32)
                    similarities = cosine_similarity(new_vec, past_embeddings)[0]
                    if max(similarities) >= 0.65:
                        logging.warning(f"🚫 Too similar to past quiz questions (cos sim ≥ 0.65):
{question}")
                        continue

                mcq.update({
                    "difficulty": difficulty,
                    "b": assign_difficulty_parameter(user_id, difficulty),
                    "a": assign_discrimination_parameter(),
                    "c": 0.2
                })

                new_vector = embedding_model.encode([question]).astype(np.float32)
                index.add(new_vector)
                valid_mcqs.append(mcq)
                batch_generated_questions.add(question)

                if len(valid_mcqs) >= 3:
                    break

            if len(valid_mcqs) < 3:
                retries += 1
                logging.warning(f"\u26a0 Still need {3 - len(valid_mcqs)} MCQs. Retrying...
({retries}/{max_retries})")

        except Exception as e:
            logging.error(f"\u26a0 Error in MCQ generation: {e}")
            retries += 1
            time.sleep(1)  # slight delay between retries

    return valid_mcqs
```

Figure 23: Prompt construction using topic, difficulty, and context MCQ to guide the LLaMA 2 model in generating targeted multiple-choice questions.

```python
# Method to retrieve diverse context questions to generate new questions
def retrieve_context_questions(query_text, top_k=3):
    """Retrieve diverse MCQs from different clusters and difficulty levels for better generation
context."""
    query_vector = embedding_model.encode([query_text]).astype(np.float32)

    if index.ntotal == 0:
        logging.warning("⚠ FAISS index is empty! No previous questions available.")
        return pd.DataFrame()

    # Retrieve 3x top_k for diversity filtering
    D, I = index.search(query_vector, k=min(top_k * 3, index.ntotal))

    valid_indices = [i for i in I[0] if 0 <= i < len(dataset)]
    retrieved_questions = dataset.iloc[valid_indices]

    unique_clusters = set()
    used_difficulties = set()
    context_questions = []

    for _, row in retrieved_questions.iterrows():
        cluster = row["Cluster"]
        difficulty = row.get("Difficulty Level", "").lower()

        if cluster not in unique_clusters and difficulty not in used_difficulties:
            context_questions.append(row)
            unique_clusters.add(cluster)
            used_difficulties.add(difficulty)

        if len(context_questions) >= top_k:
            break

    if not context_questions:
        logging.warning("⚠ No diverse context questions found.")

    else:
        logging.info("🧠 Context Questions Selected:")
        for row in context_questions:
            logging.info(
                f" - Cluster: {row['Cluster']}, Difficulty: {row.get('Difficulty Level', 'N/A')}, "
                f"Q: {row['Question Text'][:60]}..."
            )

    return pd.DataFrame(context_questions)
```

Figure 24: Semantic retrieval of a context question using SentenceTransformer and FAISS

```python
@router.get("/generate_adaptive_mcqs/{user_id}/{question_count}")
def generate_next_quiz(user_id: str, question_count: int, current_user: str =
Depends(get_current_user)):
    """Generate a new adaptive quiz based on user's previous performance."""
    try:
        logging.info(f"📄 Starting adaptive quiz generation for user {user_id} with {question_count}
questions...")
        sys.stdout.flush()  # Force log flushing

        existing_user = users_collection.find_one({"_id": ObjectId(user_id)})
        if not existing_user:
            logging.error("User not found")
            sys.stdout.flush()
            raise HTTPException(status_code=404, detail="User not found. Please register before
generating a quiz.")

        if current_user != user_id:
            logging.error("Unauthorized access")
            sys.stdout.flush()
            raise HTTPException(status_code=403, detail="Unauthorized access")

        difficulty_distribution = get_irt_based_difficulty_distribution(user_id, question_count)
        logging.info(f"📊 Difficulty distribution: {difficulty_distribution}")
        sys.stdout.flush()

        # 🧠 Cache past questions once
        past_questions = get_seen_questions(user_id)
        past_embeddings = (
            embedding_model.encode(past_questions).astype(np.float32)
            if past_questions else None
        )

        current_quiz_questions = set()
        mcqs = []

        for difficulty, count in difficulty_distribution.items():
            generated = 0
            failed_attempts = 0
            while generated < count and failed_attempts < 5:
                logging.info(f"⚙️ Generating MCQ (Difficulty: {difficulty}) - Attempt {generated +
1}/{count}")
                sys.stdout.flush()

                batch_mcqs = generate_mcq_based_on_performance(user_id, difficulty,
 existing_questions=current_quiz_questions, past_embeddings=past_embeddings)

                logging.info(f"📩 Received MCQ response: {batch_mcqs}")
                sys.stdout.flush()

                if not batch_mcqs:
                    failed_attempts += 1
                    logging.warning(f"⚠️ No MCQs received. Retrying... ({failed_attempts}/5)")
                    continue

                for mcq in batch_mcqs:
                    q_text = mcq.get("question", "")
                    if not q_text or q_text in current_quiz_questions:
                        continue

                    formatted_mcq = {
                        "question_text": q_text,
                        "option1": mcq.get("options", {}).get("A", "N/A"),
                        "option2": mcq.get("options", {}).get("B", "N/A"),
                        "option3": mcq.get("options", {}).get("C", "N/A"),
                        "option4": mcq.get("options", {}).get("D", "N/A"),
                        "option5": mcq.get("options", {}).get("E", "N/A"),
                        "correct_answer": mcq.get("correct_answer", "N/A"),
                        "difficulty": difficulty
                    }

                    current_quiz_questions.add(q_text)
                    mcqs.append(formatted_mcq)
                    generated += 1
                    sys.stdout.flush()

                    if generated >= count:
                        break

        if len(mcqs) < question_count:
            remaining_needed = question_count - len(mcqs)
            logging.warning(f"⚠️ Not enough questions generated. Fetching {remaining_needed} from DB.")
            sys.stdout.flush()

            db_questions = fetch_questions_from_db(remaining_needed)
            mcqs.extend(db_questions)

        quiz_id = str(uuid.uuid4())
        quiz_data = {
            "quiz_id": quiz_id,
            "user_id": user_id,
            "difficulty_distribution": difficulty_distribution,
            "questions": mcqs,
            "created_at": time.time(),
        }

        logging.info("🛠️ Saving quiz to the database...")
        sys.stdout.flush()
        quizzes_collection.insert_one(quiz_data)

        logging.info(f" Quiz generated successfully! Quiz ID: {quiz_id}")
        sys.stdout.flush()

        return {"quiz_id": quiz_id, "total_questions": len(mcqs), "mcqs": mcqs}

    except Exception as e:
        logging.error(f" Error generating adaptive quiz: {str(e)}")
        logging.error(traceback.format_exc())
        sys.stdout.flush()
        raise HTTPException(status_code=500, detail=str(e))
```

Figure 25: API endpoint that handles adaptive quiz generation, session logging, and quiz delivery to the frontend interface.

2.1.4.7 Unit-Based Quiz Generation

In addition to adaptive quizzes, the system offers a Unit-Based Quiz Generation module that allows students to focus on specific chapters or concepts within the A/L Biology syllabus. This feature supports targeted revision, enabling learners to reinforce their understanding in particular areas, either during classroom study or while preparing for exams.

The process begins when the user selects a biology unit from the list of syllabus chapters. Upon receiving this request, the backend filters the curated MCQ dataset to extract only those questions tagged with the specified unit. These questions are manually categorized and labeled during the dataset preparation phase, ensuring precise mapping between each question and its corresponding unit.

To maintain diversity in each quiz attempt, the system performs semantic similarity filtering using SentenceTransformer embeddings. The current request is compared with previously served questions for that student to ensure that duplicate or very similar questions are not reused. This step also accounts for fallback logic — if an insufficient number of new questions is available after filtering, the system supplements the quiz using randomly selected questions from the same unit, ensuring a complete quiz is always delivered.

While the unit-wise quiz module is non-adaptive in structure, it plays a vital complementary role in the platform. It allows students to focus their efforts on weak areas, practice specific topics in isolation, and build confidence before engaging with more adaptive and mixed-difficulty assessments.

```python
@router.get("/unit_quiz/generate/{user_id}")
def generate_unit_quiz(
    user_id: str,
    unit: str = Query(...),
    question_count: int = Query(10, ge=1, le=100),
    current_user: str = Depends(get_current_user)
):
    existing_user = users_collection.find_one({"_id": ObjectId(user_id)})
    if not existing_user:
        raise HTTPException(status_code=404, detail="User not found. Please register before generating
a quiz.")

    if current_user != user_id:
        raise HTTPException(status_code=403, detail="Unauthorized access")

    filtered = unit_df[unit_df["Assigned_Unit"] == unit]
    if filtered.empty:
        raise HTTPException(status_code=404, detail=f"No questions found for {unit}.")

    used_questions = get_previously_seen_questions(current_user, unit)
    filtered = filtered[~filtered["Question Text"].isin(used_questions)]

    sampled_questions = []
    seen = set()
    attempts = 0
    max_attempts = 100
    clusters = list(filtered["Cluster"].unique()) if "Cluster" in filtered.columns else [None]
    random.shuffle(clusters)

    while len(sampled_questions) < question_count and attempts < max_attempts:
        cluster = random.choice(clusters)
        cluster_qs = filtered[filtered["Cluster"] == cluster] if cluster else filtered
        if not cluster_qs.empty:
            q = cluster_qs.sample(1).iloc[0]
            q_text = q["Question Text"]
            if q_text not in seen:
                sampled_questions.append(format_question(q))
                seen.add(q_text)
        attempts += 1

    if len(sampled_questions) < question_count:
        needed = question_count - len(sampled_questions)
        fallback_questions = get_semantically_similar_questions(
            [q["question_text"] for q in sampled_questions],
            excluded_units=[unit],
            used_questions=used_questions.union(seen),
            count=needed
        )
        sampled_questions.extend(fallback_questions)

    quiz_entry = {
        "user_id": current_user,
        "unit_name": unit,
        "questions": sampled_questions,
        "created_at": datetime.utcnow()
    }
    quiz_id = unit_quizzes.insert_one(quiz_entry).inserted_id

    return {
        "message": f"Quiz generated for {unit}",
        "quiz_id": str(quiz_id),
        "questions": sampled_questions
    }
```

Figure 26: Unit-based quiz filtering

2.1.4.8 Performance Analytics and Dashboard Integration

To support a personalized and data-driven learning experience, the system incorporates a comprehensive **Performance Analytics and Dashboard module**. This is responsible for aggregating, storing, and exposing insights related to students' quiz performance.

Each time a quiz (adaptive or unit-based) is completed, the backend records detailed metadata about the session. This includes the user ID, quiz ID, time of submission, individual question responses, accuracy, and the overall score. These records are stored in a structured format within the database, enabling consistent retrieval and historical tracking.

Upon receiving a request from the frontend (e.g., when a student logs in and views their dashboard), the backend fetches all past quiz attempts associated with the user. This data is then processed to compute several key metrics:

- **Topic-wise Accuracy:** By cross-referencing each question with its associated unit tag, the backend calculates average accuracy for each biology unit. This helps students understand which chapters they have mastered and which need further review.

- **Score Trends Over Time:** The system logs performance chronologically, allowing the dashboard to plot a timeline of scores across multiple quiz attempts. This visual trend helps students track progress and recognize patterns in their improvement.

- **Adaptive Ability Tracking:** For users engaging in adaptive quizzes, the backend maintains their evolving IRT-based ability score, which is returned as part of the analytics data. This score reflects the user’s estimated competency level and how it changes over time.

- **Leaderboard Positioning:** The backend also calculates the student's global and classroom rank based on total quizzes taken, average score, and cumulative accuracy. This encourages friendly competition and motivates consistent practice.

These metrics are bundled into a structured API response that the frontend uses to populate the student dashboard. The system ensures that all analytics are computed in real-time and refreshed after each quiz submission, providing an up-to-date view of performance.

By offering detailed backend support for performance analytics, the system ensures that learners receive transparent, actionable feedback, helping them identify strengths, address weaknesses, and stay engaged in their academic progress.

```python
@router.get("/dashboard_data/{user_id}")
def get_dashboard_data(user_id: str, current_user: str =
Depends(get_current_user)):
    """Returns structured performance data for the user dashboard."""
    user_data = users_collection.find_one({"_id": ObjectId(user_id)})
    if not user_data:
            raise HTTPException(status_code=404, detail="User not found.")

    if current_user != user_id:
        logging.error(f" Unauthorized access attempt by {current_user}")
        raise HTTPException(status_code=403, detail="Unauthorized access")

    #  If user has no performance data, return default values
    if not user_data or "performance" not in user_data:
        return {
            "total_quizzes": 0,
            "accuracy_easy": 0,
            "accuracy_medium": 0,
            "accuracy_hard": 0,
            "time_easy": 0,
            "time_medium": 0,
            "time_hard": 0,
            "strongest_area": "N/A",
            "weakest_area": "N/A",
            "consistency_score": 0,
            "last_10_quizzes": [],
            "message": "No quiz data available yet. Start taking quizzes!"
        }
    performance = user_data["performance"]

    return {
        "total_quizzes": performance.get("total_quizzes", 0),
        "accuracy_easy": performance.get("accuracy_easy", 0),
        "accuracy_medium": performance.get("accuracy_medium", 0),
        "accuracy_hard": performance.get("accuracy_hard", 0),
        "time_easy": performance.get("time_easy", 0),
        "time_medium": performance.get("time_medium", 0),
        "time_hard": performance.get("time_hard", 0),
        "strongest_area": performance.get("strongest_area", "N/A"),
        "weakest_area": performance.get("weakest_area", "N/A"),
        "consistency_score": performance.get("consistency_score", 0),
        "last_10_quizzes": performance.get("last_10_quizzes", []),
    }
```

Figure 27:Backend logic for returning student performance analytics.

2.1.4.9 Frontend Development with React and Tailwind CSS

The frontend of the adaptive quiz platform is developed using **React.js** in combination with **Tailwind CSS**, creating a highly responsive and modular user interface. The design prioritizes both functionality and usability, ensuring that students can interact with quizzes, track their performance, and access historical data in a seamless and visually engaging environment. Throughout the development process, a consistent folder and component structure is maintained, allowing for clear separation of concerns and scalable code management.

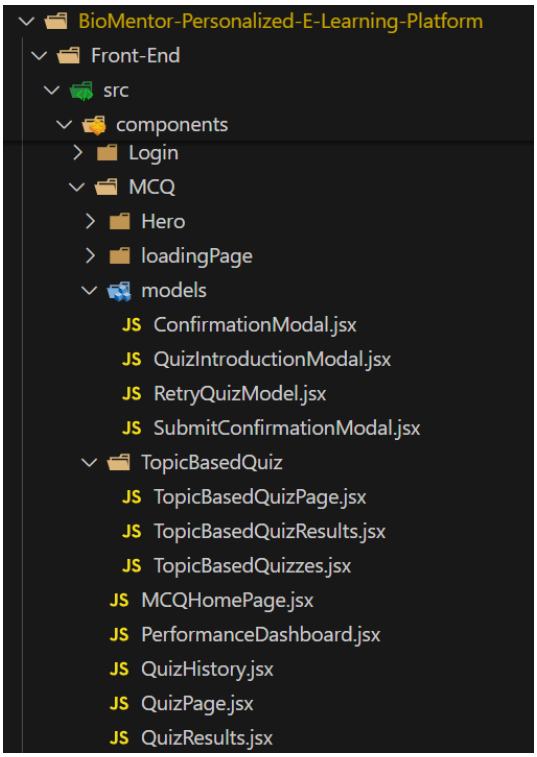


Figure 28: Frontend folder structure maintained in Visual Studio Code.

Component Architecture and Routing

The frontend architecture is component-based, following a modular approach in which each major page and feature is encapsulated in its own file. React Router is used to handle client-side navigation between different views, such as the adaptive

quiz interface, unit-wise quizzes, result displays, the performance dashboard, and quiz history. Each of these routes is connected to its own stateful component, which manages data fetching, interaction logic, and rendering.

The application begins with the MCQ Home Page, which acts as the central hub from which users can choose between adaptive quizzes, topic-based quizzes, reviewing their performance dashboard, or accessing previous quiz attempts. Each section is visually distinguished using interactive cards styled with Tailwind utility classes. These cards feature dynamic icons, hover animations, and responsive layout adaptations to ensure clarity across both desktop and mobile devices.

Figure 29: MCQ homepage

Adaptive Quiz Interface

The adaptive quiz experience is delivered through a dedicated page, which dynamically renders multiple-choice questions one at a time. The quiz page includes several interactive elements: a timer, a progress bar, difficulty level indicators, and selectable answer options. The current question is displayed in a stylized container that also includes the difficulty badge, determined by the user's current ability level, as calculated using IRT. The quiz flow is managed through local React state, enabling smooth transitions between questions and ensuring that user inputs are preserved and validated in real time.

Once all questions are answered and submitted, the platform transitions to the Quiz Results page. This interface provides a comprehensive breakdown of the student's performance, showing overall score, accuracy percentage, per-question correctness, and time taken per question (if recorded). The questions are also color-coded to provide instant visual feedback, using green for correct answers and red for incorrect ones. Difficulty distribution is shown either as text or through a small, embedded chart for better interpretability.

Figure 30:Adaptive quiz result summary

Figure 31: Adaptive quiz page

Topic-Based Quiz Flow

For students seeking focused revision, the system offers a dedicated topic-wise quiz experience. The TopicBasedQuizzes page presents all available syllabus units as visually styled cards. Each unit card includes the unit name, best past performance (if any), and a prompt to begin the quiz. Upon selection, the platform transitions to the TopicBasedQuizPage, where students are presented with a set of static, pre-selected MCQs from the chosen topic.

After completion, students are directed to the TopicBasedQuizResults page, which follows a similar layout to the adaptive result screen. This result view includes detailed feedback for each question and also allows students to compare their latest performance against previous attempts for the same topic. This feature supports incremental progress tracking in specific content areas.

Figure 32: unit-wise quiz cards for focused topic-based practice.

Quiz History and Performance Dashboard

The **Quiz History** page provides a chronological list of all quiz attempts made by the user. Each entry includes the date and time, quiz type, score, and an option to either retry the quiz or view the detailed results. The history is organized in a scrollable layout, optimized for both desktop and mobile devices.

Performance analytics are centralized in the **PerformanceDashboard** component. Upon accessing the dashboard, users are presented with a set of interactive charts and summaries, pulled from the backend's analytics API. The dashboard displays topic-wise accuracy through bar or radar charts, score trends over time using line graphs, and relative performance rankings through a leaderboard view. Tailwind's grid and flex utilities are used to manage layout across viewports, and data visualizations are implemented using Chart.js for rendering quality and clarity.

Figure 33: Quiz history

Styling, Responsiveness, and Interaction Design

Tailwind CSS enables rapid UI development through utility-first class names that directly reflect layout, spacing, color, and interaction properties. All major interface components — including quiz cards, buttons, modals, and feedback blocks — are fully responsive. Breakpoints are applied to ensure the layout adapts across mobile, tablet, and desktop views. Transition effects, hover animations, and scaling behaviors further enhance the user experience.

Framer Motion is used to animate elements such as quiz transitions, score popups, and page navigation. These subtle yet purposeful animations improve user engagement without detracting from clarity or speed.

State Management and API Communication

Frontend state management is handled using React's useState and useEffect hooks. The quiz state, selected answers, timing, and user feedback messages are managed at the component level. Communication with the backend is achieved through Axios, with authentication tokens stored in localStorage to persist login sessions. API calls are made asynchronously, with loading indicators and error messages implemented to provide feedback during interactions such as quiz generation and result submission.

By maintaining a clean separation between UI logic and data fetching, the frontend ensures consistent behavior even in the case of slow or failed network responses. Retry logic and status monitoring are used to prevent incomplete submissions or data loss.

2.1.5 Testing

The testing process undertaken for the adaptive quiz component of the "Bio Mentor" platform followed a structured and layered approach. It included **unit testing**, **integration testing**, **system testing**, and **acceptance testing**, each designed to validate the stability, reliability, and user experience of the component under real-world educational use cases. These phases were critical in ensuring that the adaptive quiz system functions effectively as part of the broader group project.

Unit Testing:

Unit testing formed the foundation of the testing strategy. Python's built-in unittest framework was used to create a suite of tests targeting individual functions and logic blocks within the backend. Key areas tested included IRT-based ability calculation, difficulty distribution logic, MCQ formatting, route responses, and quiz submission workflows. Specific functions such as assign_difficulty_parameter, calculate_probability, generate_mcq, and question parsing were tested against various inputs to verify that they returned accurate and expected results. These tests ensured that the component's core logic remained robust and stable across updates.

Integration Testing:

Integration testing was performed to assess the interaction between the adaptive quiz module and other components such as the quiz history tracker, user performance dashboard, and frontend quiz interface. These tests involved triggering quiz generation from the frontend, submitting answers, and confirming that results were correctly stored and reflected in dashboard metrics. The API endpoints were tested using tools such as Postman and automated test clients to validate proper data flow and endpoint behavior. This ensured seamless communication between the frontend and backend, with consistent response formats and minimal latency.

System Testing:

System testing involved validating the adaptive quiz module within the full deployment of the "Bio Mentor" platform, hosted on Microsoft Azure. During this phase, the complete workflow—from user login to quiz generation, submission, result analysis, and dashboard rendering—was tested as a unified system. The tests confirmed that the adaptive quiz functioned correctly when integrated with authentication, analytics, leaderboard systems, and other modules developed by group members. System testing also helped verify responsiveness across different devices and browsers.

Acceptance Testing:

Acceptance testing involved real students and academic reviewers interacting with the deployed system. Alpha testing was first conducted within a controlled environment, where students explored both the adaptive and unit-wise quiz features, and their feedback was recorded. This helped identify and resolve usability issues, content alignment mismatches, and small bugs. Following this, beta testing was carried out by allowing users to access the system in their own environments. This stage validated the system’s performance in real-world usage scenarios and confirmed that quizzes were relevant, results accurate, and analytics understandable. Feedback from both students and subject experts was positive, particularly regarding the platform’s personalized quiz generation and adaptive difficulty logic.

By following this layered and comprehensive testing strategy, the adaptive quiz component has been thoroughly evaluated for performance, accuracy, and usability. This approach not only ensured the quality of the component itself but also contributed to the seamless operation of the overall "Bio Mentor" platform, reinforcing its readiness for real-world educational deployment.

2.1.5.1 Unit Testing

To validate the correctness and reliability of critical components in the adaptive quiz platform, Python's built-in **unittest framework** and **pytest** were used to write and execute automated unit tests. The unit tests are focused on verifying the logic of MCQ generation, IRT-based ability scoring, quiz routing, and response submission. Test cases were organized by functionality and covered scenarios such as checking IRT parameter ranges, semantic filtering, quiz creation endpoints, and correct answer parsing.

The unit tests also simulate user interactions such as quiz submission and result fetching by mocking request payloads and verifying backend responses. These tests ensure that each individual component operates as expected and contributes to the robustness of the overall system. Testing was performed in isolation with dummy user IDs and mocked model/data interactions.

```python
import sys
import os
from bson import ObjectId

# Dynamically add the absolute path to project root (MCQ) to sys.path
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))
dummy_user_id = str(ObjectId())

from utils.quiz_generation_methods import (
    assign_difficulty_parameter,
    assign_discrimination_parameter,
    get_irt_based_difficulty_distribution
)

def test_assign_difficulty_parameter_easy_range():
    value = assign_difficulty_parameter(dummy_user_id, "easy")
    assert -2.0 <= value <= 1.0

def test_assign_difficulty_parameter_hard_range():
    value = assign_difficulty_parameter(dummy_user_id, "hard")
    assert isinstance(value, float)

def test_assign_discrimination_parameter_range():
    value = assign_discrimination_parameter()
    assert 0.5 <= value <= 2.0

def test_get_irt_based_difficulty_distribution_low_theta():
    distribution = get_irt_based_difficulty_distribution(dummy_user_id, 10)
    assert sum(distribution.values()) == 10
    assert isinstance(distribution, dict)

def test_get_irt_based_difficulty_distribution_high_theta():
    distribution = get_irt_based_difficulty_distribution(dummy_user_id, 10)
    assert sum(distribution.values()) == 10
    assert isinstance(distribution, dict)
```

Figure 34: Unit tests verifying IRT difficulty assignment and distribution logic.

```python
import sys
import os
import pandas as pd
import numpy as np
from unittest.mock import patch
from bson import ObjectId

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

import utils.generate_question as gq
from utils.generate_question import generate_mcq, generate_mcq_based_on_performance

FAKE_RAW_OUTPUT = """
Question 1: What is the powerhouse of the cell?
A) Nucleus
B) Ribosome
C) Mitochondria
D) Golgi apparatus
E) Lysosome
Correct Answer: C
"""

def mock_extracted_mcq():
    return [{
        "question": "What is the powerhouse of the cell?",
        "options": {
            "A": "Nucleus",
            "B": "Ribosome",
            "C": "Mitochondria",
            "D": "Golgi apparatus",
            "E": "Lysosome"
        },
        "correct_answer": "C"
    }]

@patch("utils.generate_question.requests.post")
@patch("utils.generate_question.extract_mcqs", return_value=mock_extracted_mcq())
@patch("utils.generate_question.index.add")
@patch("utils.generate_question.embedding_model.encode", return_value=np.array([[0.1]*384],
dtype=np.float32))
@patch("utils.generate_question.retrieve_context_questions", return_value=pd.DataFrame())
def test_generate_mcq_success(mock_context, mock_encode, mock_add, mock_extract, mock_post):
    mock_df = pd.DataFrame([{
        "Question Text": "What is the powerhouse of the cell?",
        "Correct Answer": "C",
        "Cluster": 1
    }])

    with patch.object(gq, "dataset", mock_df):
        mock_post.return_value.status_code = 200
        mock_post.return_value.json.return_value = {"raw_output": FAKE_RAW_OUTPUT}

        result = generate_mcq("easy", str(ObjectId()), max_retries=1)

        assert isinstance(result, list)
        assert len(result) > 0
        assert result[0]["difficulty"] == "easy"

@patch("utils.generate_question.requests.post")
@patch("utils.generate_question.extract_mcqs", return_value=mock_extracted_mcq())
@patch("utils.generate_question.index.add")
@patch("utils.generate_question.embedding_model.encode", return_value=np.array([[0.1]*384],
dtype=np.float32))
@patch("utils.generate_question.retrieve_context_questions", return_value=pd.DataFrame())
@patch("utils.generate_question.estimate_student_ability", return_value=0.5)
def test_generate_mcq_based_on_performance_success(mock_theta, mock_context, mock_encode, mock_add,
mock_extract, mock_post):
    mock_df = pd.DataFrame([{
        "Question Text": "What is the powerhouse of the cell?",
        "Correct Answer": "C",
        "Cluster": 1
    }])

    with patch.object(gq, "dataset", mock_df):
        mock_post.return_value.status_code = 200
        mock_post.return_value.json.return_value = {"raw_output": FAKE_RAW_OUTPUT}

        result = generate_mcq_based_on_performance(str(ObjectId()), "medium", max_retries=1)

        assert isinstance(result, list)
        assert len(result) > 0
        assert result[0]["difficulty"] == "medium"
        assert "question" in result[0]
```

Figure 35: Unit test using mocks to validate MCQ generation with context and model inference.

```python
import pytest
from fastapi.testclient import TestClient
from main import app
from database.database import users_collection, unit_quizzes, unit_quiz_responses
from bson import ObjectId
from utils.user_mgmt_methods import get_current_user
from datetime import datetime

# Use a known test ID for consistent mock
TEST_USER_ID = "000000000000000000000001"

# Dependency override to simulate authentication
def override_get_current_user():
    return TEST_USER_ID

app.dependency_overrides[get_current_user] = override_get_current_user
client = TestClient(app)

@pytest.fixture(scope="module", autouse=True)
def setup_test_user():
    users_collection.insert_one({
        "_id": ObjectId(TEST_USER_ID),
        "username": "test_user",
        "performance": {
            "total_quizzes": 0,
            "accuracy_easy": 0,
            "accuracy_medium": 0,
            "accuracy_hard": 0,
            "time_easy": 0,
            "time_medium": 0,
            "time_hard": 0,
            "strongest_area": "N/A",
            "weakest_area": "N/A",
            "consistency_score": 0,
            "last_10_quizzes": []
        }
    })
    yield
    users_collection.delete_many({"_id": ObjectId(TEST_USER_ID)})

def test_dashboard_data():
    response = client.get(f"/dashboard_data/{TEST_USER_ID}")
    assert response.status_code in [200, 404]  # Accept default or seeded response

def test_performance_graph():
    response = client.get(f"/performance_graph/{TEST_USER_ID}")
    assert response.status_code in [200, 404]

def test_leaderboard():
    response = client.get("/leaderboard")
    assert response.status_code in [200, 404]

def test_generate_unit_quiz():
    unit = "Photosynthesis"  # Change this to a valid unit in your dataset if needed
    response = client.get(f"/unit_quiz/generate/{TEST_USER_ID}?unit={unit}&question_count=1")
    if response.status_code == 200:
        assert "quiz_id" in response.json()
    else:
        assert response.status_code == 404  # If unit doesn't exist

def test_submit_unit_quiz():
    unit = "Photosynthesis"
    gen_response = client.get(f"/unit_quiz/generate/{TEST_USER_ID}?unit={unit}&question_count=1")
    if gen_response.status_code != 200:
        return  # skip test if no questions

    quiz_id = gen_response.json()["quiz_id"]
    questions = gen_response.json()["questions"]

    payload = {
        "quiz_id": quiz_id,
        "responses": [
            {"question_text": q["question_text"], "selected_answer": "A"}
            for q in questions
        ]
    }

    response = client.post(f"/unit_quiz/submit/{TEST_USER_ID}", json=payload)
    assert response.status_code == 200
    assert "score" in response.json()

def test_unit_quiz_results():
    unit = "Photosynthesis"
    gen_response = client.get(f"/unit_quiz/generate/{TEST_USER_ID}?unit={unit}&question_count=1")
    if gen_response.status_code != 200:
        return

    quiz_id = gen_response.json()["quiz_id"]
    questions = gen_response.json()["questions"]

    client.post(f"/unit_quiz/submit/{TEST_USER_ID}", json={
        "quiz_id": quiz_id,
        "responses": [{"question_text": q["question_text"], "selected_answer": "A"} for q in questions]
    })

    response = client.get(f"/unit_quiz/results/{quiz_id}")
    assert response.status_code == 200
    assert "responses" in response.json()

    def test_generate_mcqs():
    response = client.get(f"/generate_mcqs/{TEST_USER_ID}")
    assert response.status_code in [200, 404]
    if response.status_code == 200:
        data = response.json()
        assert "quiz_id" in data
        assert "mcqs" in data

def test_generate_adaptive_mcqs():
    response = client.get(f"/generate_adaptive_mcqs/{TEST_USER_ID}/5")
    assert response.status_code in [200, 404]
    if response.status_code == 200:
        data = response.json()
        assert "quiz_id" in data
        assert "mcqs" in data


def test_submit_quiz_and_get_quiz():
    # First generate quiz
    gen = client.get(f"/generate_mcqs/{TEST_USER_ID}")
    if gen.status_code != 200:
        pytest.skip("MCQ quiz not available for submission test")

    quiz_data = gen.json()
    quiz_id = quiz_data["quiz_id"]
    questions = quiz_data["mcqs"]

    # Submit fake answers
    submission_payload = {
        "user_id": TEST_USER_ID,
        "quiz_id": quiz_id,
        "responses": [
            {
                "question_text": q["question_text"],
                "selected_answer": "A",
                "time_taken": 3.0
            } for q in questions
        ]
    }
    submit = client.post("/submit_quiz/", json=submission_payload)
    assert submit.status_code == 200
    assert "summary" in submit.json()

    # Fetch quiz using the same quiz_id
    fetch = client.get(f"/get_quiz/{quiz_id}")
    assert fetch.status_code == 200
    assert fetch.json()["quiz_id"] == quiz_id
```

Figure 36: Unit tests for APIs

```python
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))
from utils.text_extraction import extract_mcqs
from utils.quiz_generation_methods import clean_correct_answer

def test_extract_mcqs_parsing_single_question():
    prompt = "Ignore this prompt."
    raw_output = """
    Question 1: What is the powerhouse of the cell?
    A) Nucleus
    B) Ribosome
    C) Mitochondria
    D) Chloroplast
    E) Endoplasmic Reticulum
    Correct Answer: C
    """
    parsed = extract_mcqs(prompt, raw_output)
    assert len(parsed) == 1
    assert parsed[0]["question"].startswith("What is")
    assert parsed[0]["correct_answer"] == "C"
    assert "A" in parsed[0]["options"]
    assert len(parsed[0]["options"]) == 5

def test_clean_correct_answer_letters():
    assert clean_correct_answer("Correct Answer: A") == ["A"]
    assert clean_correct_answer("Answer: C and D") == ["C", "D"]
    assert clean_correct_answer("B, D") == ["B", "D"]
    assert clean_correct_answer("(E)") == ["E"]
    assert clean_correct_answer("A, A, C") == ["A", "C"]
    assert clean_correct_answer("123") == []
    assert clean_correct_answer("") == []
    assert clean_correct_answer("Correct Answer: A, C, E") == ["A", "C", "E"]
```

Figure 37: Tests verifying MCQ extraction and correct answer cleaning from LLM output.

2.1.5.2 Manual Testing:

Manual testing is a critical stage of validating the adaptive quiz platform, ensuring all components function correctly when interacted with by real users. This form of testing involves human testers manually executing various test scenarios without automation, simulating real student behavior across different parts of the system. It serves to catch usability issues, logical errors, and inconsistencies that automated scripts may not detect.

Purpose of Manual Testing

The primary goal of manual testing in this context is to verify the functionality, usability, and consistency of the system through hands-on evaluation. This helps confirm that the system meets the needs of students by adapting intelligently, delivering relevant content, and providing actionable feedback.

Manual testers interact with the application using both desktop and mobile environments, replicating various user journeys such as taking quizzes, checking analytics, and revisiting past attempts.

Test Case Development and Execution

Test cases were derived from the system's core functional requirements and user stories. Each test case includes specific steps, expected outcomes, and actual system behavior, recorded using structured templates. Testing focused on the frontend's interaction with the backend, particularly around quiz generation, submission, and performance tracking. The test cases were executed in multiple rounds to confirm reliability.

Types of Manual Testing Applied

✓ **Functional Testing**: Verifying that quizzes generate, submit, and display correctly.

✓ **Usability Testing**: Ensuring that students can navigate and use the platform without guidance.

✓ **Exploratory Testing**: Allowing testers to freely explore the system and identify unexpected behaviors.

✓ **User Acceptance Testing (UAT)**: Having real A/L students and teachers interact with the platform and provide feedback.

✓ **Compatibility Testing**: Checking consistent behavior across different devices and browsers.

Below are the test cases which were done for the manual testing. Tables 4, 5, 6, 7, 8, 9, 10, 11, 12 and 13 display the manual test cases.

Test Case ID	TC_01
Test Case Objective	Generate Initial Adaptive Quiz
Pre-Requirements	User is logged in
Test Steps	1. Login to the system. 2. Navigate to MCQ Home Page 3. Click "Start Adaptive Quiz" Button
Test Data	User details
Expected Output	Quiz generated with equal easy, medium, hard questions
Actual Output	Quiz generated with mixed difficulty questions
Status	Pass

Table 5: Test case for generating the initial adaptive quiz with balanced difficulty.

Test Case ID	TC_02
Test Case Objective	Submit First Quiz and Record Performance
Pre-Requirements	Quiz completed
Test Steps	1. Select answers 2. Submit quiz
Test Data	Balanced answers
Expected Output	Result page showing score and performance summary
Actual Output	Results displayed with feedback
Status	Pass

Table 6: Test case for submitting the first adaptive quiz and recording performance.

Test Case ID	TC_03
Test Case Objective	Generate Second Adaptive Quiz Based on Performance
Pre-Requirements	At least one adaptive quiz is completed
Test Steps	1. Login to the system. 2. Navigate to MCQ Home Page 3. Start new adaptive quiz
Test Data	Ability score applied
Expected Output	Quiz with adapted difficulty ratio
Actual Output	Difficulty adjusted correctly
Status	Pass

Table 7: Test case for generating an adaptive quiz based on previous performance.

Test Case ID	TC_04
Test Case Objective	Submit Adaptive Quiz and Update Ability Score
Pre-Requirements	Quiz generated from updated ratio
Test Steps	1. Select answers 2. Submit quiz
Test Data	Mixed difficulty responses
Expected Output	Ability score updated in backend
Actual Output	Ability updated and stored
Status	Pass

Table 8: Test case for submitting an adaptive quiz and updating ability score.

Test Case ID	TC_05
Test Case Objective	Start and Complete Unit-Based Quiz
Pre-Requirements	Unit quiz page accessed
Test Steps	1. Select unit 2. Start quiz 3. Select answers for the questions 3. Submit selected answers
Test Data	Unit = “Genetics”
Expected Output	Questions based on chosen unit, result displayed
Actual Output	Correct unit-based quiz delivered
Status	Pass

Table 9: Test case for starting and completing a unit-based quiz.

Test Case ID	TC_06
Test Case Objective	View Detailed Results and Feedback
Pre-Requirements	Quiz attempt completed
Test Steps	1. Open Quiz history 2. Click a quiz 3. View details of the quiz
Test Data	Past quiz data
Expected Output	Displays per-question feedback and score
Actual Output	Results with feedback shown correctly
Status	Pass

Table 10: Test case for viewing detailed quiz results and feedback.

Test Case ID	TC_07
Test Case Objective	Access Performance Dashboard
Pre-Requirements	At least 1 quiz is completed
Test Steps	1. Login to the system. 2. Navigate to Dashboard
Test Data	Quiz performance data
Expected Output	Charts display topic-wise accuracy, leaderboard
Actual Output	Dashboard loaded and updated correctly
Status	Pass

Table 11: Test case for accessing the performance dashboard and metrics.

Test Case ID	TC_08
Test Case Objective	Retry Adaptive Quiz
Pre-Requirements	Quiz exists in history
Test Steps	1. Login to the system. 2. Open Quiz history 3. Click Retry for the particular quiz
Test Data	Quiz data
Expected Output	Retrieve already created quiz for the user
Actual Output	Quiz retried
Status	Pass

Table 12: Test case for retrying an adaptive quiz from quiz history.

Test Case ID	TC_09
Test Case Objective	Take Quiz on Mobile View
Pre-Requirements	Mobile browser
Test Steps	1. Login to the system. 2. Navigate to MCQ Home Page 3. Click "Start Adaptive Quiz" Button
Test Data	Android Chrome
Expected Output	Layout responsive, no overlaps or broken UI
Actual Output	Interface worked on small screen
Status	Pass

Table 13: Test case for taking a quiz using a mobile device.

Test Case ID	TC_10
Test Case Objective	Submit Unit Quiz and View Export Option
Pre-Requirements	Unit quiz completed
Test Steps	1. Login to the system. 2. Go to Unit Quizzes Page 3. View Results of already completed Quiz 4. Click download
Test Data	PDF option
Expected Output	Option to export quiz or results provided
Actual Output	Export functionality worked
Status	Pass

Table 14: Test case for submitting a unit quiz and downloading results.

2.1.6 Deployment & Maintenance

Deployment:

The deployment of the adaptive quiz platform was successfully carried out on Microsoft Azure, enabling full access to both the backend services and the frontend application through a cloud-hosted environment. The deployment process was designed to ensure that the system remains consistently available, secure, and maintainable, providing students and educators with a reliable educational experience.

The backend services were deployed on a Linux-based virtual machine configured through Azure's compute infrastructure. Key services—including the FastAPI application powering quiz generation, IRT-based adaptivity, user analytics, and response processing—were installed within a Python virtual environment and launched as a persistent system service. This setup allows the backend to start automatically upon system boot, ensuring uptime even after instance restarts. Server process management is handled using Systemd, and reverse proxying is configured with Nginx, making API endpoints accessible through standard HTTP protocols.

The frontend application for BioMentor, developed using React.js and styled with Tailwind CSS, was also hosted on Azure, allowing users to interact with the platform through a browser interface. All frontend routes and API integrations were tested post-deployment to verify cross-component communication.

To ensure continuous delivery and smooth update cycles, the project integrates a CI/CD pipeline. This pipeline is connected to the project's GitHub repository and automates tasks such as code linting, dependency checks, test execution, and deployment packaging. Upon each push to the main branch, the pipeline triggers the deployment sequence, reducing manual intervention and ensuring that updates are safely and efficiently rolled out.

Monitoring and service health checks are configured to detect and resolve issues quickly. Logs are maintained for backend service activity, and Nginx access logs are periodically reviewed to track request flow. Scheduled maintenance includes dependency upgrades, database backup validation, and system-level patch updates to ensure platform integrity.

This deployment strategy ensures that the adaptive quiz platform remains stable, responsive, and easy to maintain, fulfilling its intended role in delivering an intelligent, learner-centered assessment experience to A/L Biology students.

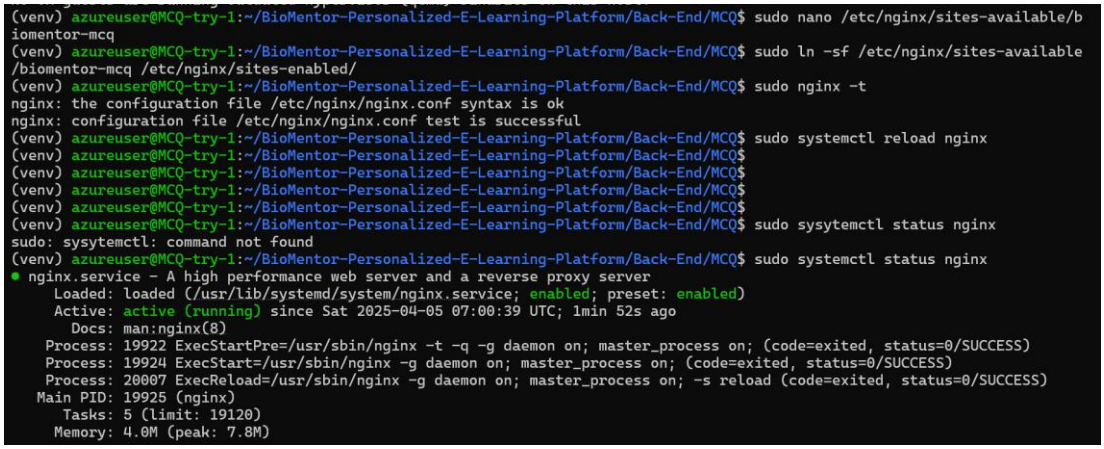


Figure 38: Backend Service Status

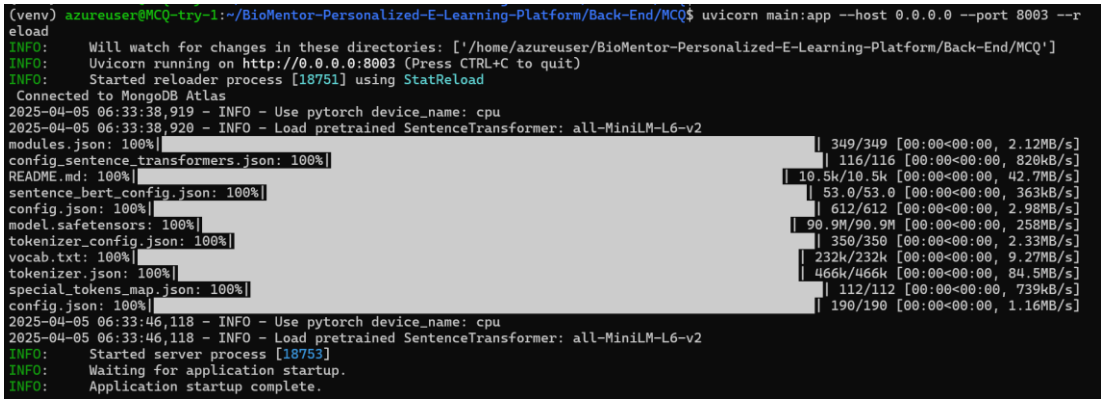


Figure 39: Backend Application Startup logs

Maintenance:

The maintenance phase is a vital component of the software lifecycle that ensures the adaptive quiz platform remains functional, secure, and responsive after its deployment. This phase involves continuous monitoring, prompt issue resolution, iterative improvements, and feature enhancements based on user feedback. It is essential for sustaining the quality and performance of the system in a real-world learning environment.

Bug Fixing and Issue Resolution

One of the primary responsibilities during maintenance is identifying and resolving defects or issues that may surface during day-to-day usage. This includes monitoring backend logs and API responses to detect unexpected behaviors or error traces. Log outputs from both the application layer and the Nginx web server are routinely analyzed to identify and isolate problems quickly.

When issues are reported by users or detected during internal reviews, they are diagnosed based on error messages, logs, or usage patterns. Fixes are developed in isolated branches and tested thoroughly before being merged into the main codebase. Updates are then deployed to the production environment using the existing CI/CD pipeline, ensuring minimal downtime and disruption.

Feature Enhancements and Updates

Beyond resolving issues, the maintenance phase also focuses on improving system functionality based on user input. Feedback is collected through informal testing sessions, survey forms, and direct user communication. Suggestions such as improved feedback presentation, new quiz configurations, or updated analytics views are reviewed and prioritized.

Enhancements are implemented incrementally, ensuring that they align with the educational goals of the platform. All new features undergo unit testing, integration testing, and where applicable, user acceptance testing (UAT) to confirm they deliver value without introducing regressions or usability challenges.

This continuous feedback loop supports the growth and adaptability of the system, ensuring it evolves in response to the needs of students and educators.

Monitoring and Support

The system is regularly monitored for performance stability and availability. System health checks and status reports are reviewed periodically to verify service uptime and response times. Automated restart configurations are in place to recover from service interruptions. Support queries or issues submitted by users are tracked and responded to in a timely manner to maintain user satisfaction and platform trust.

Overall, the maintenance phase plays a central role in ensuring the long-term sustainability of the adaptive quiz platform. It reinforces the platform's reliability and usability while enabling it to grow through continuous improvement and user-centered development.

2.2 Commercialization

The BioMentor platform, developed as a curriculum-aligned intelligent question-answering and evaluation system for A/L Biology, is well-positioned for commercial deployment. With its integration of fine-tuned large language models, adaptive scoring, feedback automation, and personalized academic insights, BioMentor holds significant potential to support the digital transformation of education in Sri Lanka and beyond. Below is a detailed plan outlining the commercialization strategy for BioMentor.

1. User Segmentation

• Students (A/L Biology stream): Individual learners looking for automated revision, feedback, and content recommendations.

• Teachers and Tutors: For evaluating student answers and monitoring performance trends.

• Educational Institutions: Schools, private academies, and tuition centers integrating digital evaluation into their curriculum.

2. Subscription-Based Pricing Model

BioMentor can be offered under a flexible SaaS (Software as a Service) model with role-specific access:

• Free Tier: Limited usage of answer generation and evaluation per day, access to past paper practice.

• Student Premium Plan: Unlimited answer generation, detailed analytics dashboard, and personalized study material recommendations.

• Teacher Dashboard Tier: Access to group analytics, feedback reports, and assignment tools.

• Institutional Tier: Centralized access for faculty and students, integration with LMS platforms, and administrative controls.

3. Authentication & Access Management

• Secure authentication via email and password or federated login (e.g., Google OAuth).

• Role-based access control (RBAC) to restrict or allow feature access for students, teachers, or admins.

4. Granular Permission Sets

• Students: Access to question generation, answer submission, feedback reports, and history.

• Teachers: View student evaluations, assign questions, view class-level analytics.

• Admins: Control user management, billing, and platform configuration.

5. Subscription and Payment Integration

• Integrate Stripe or PayPal for handling secure payments.

• Support monthly/annual plans with billing history, upgrade options, and trial access.

6. Advertising Revenue Model

• For free-tier users, unobtrusive ads can be placed in the dashboard, preferably from academic publishers, EdTech platforms, or local educational partners.

• Ensure ad placement is non-intrusive, complies with GDPR/local data laws, and aligns with student interests.

7. Institutional Partnership Opportunities

• Collaborate with schools to provide classroom-scale deployment.

• Offer training and onboarding packages for teachers and IT staff.

• White-label version of BioMentor for use in tuition centers.

8. Marketing and Outreach

• Target educational forums, social media, and school networks.

• Showcase exam-aligned performance analytics, model answer generation, and feedback automation.

• Highlight the system's alignment with Sri Lankan A/L curriculum to gain trust among local educators.

9. User Feedback and Product Iteration

• Deploy feedback capture tools within the dashboard to collect user experience data.

• Use this to iteratively enhance performance, interface design, and feature relevance (e.g., adding Chemistry/Physics support in future versions).

3. RESULTS & DISCUSSION

The development and deployment of the adaptive quiz platform for A/L Biology students has resulted in a robust, interactive, and performance-driven e-learning system. This section presents the outcomes from various components of the platform and offers a critical discussion of the insights gathered during testing, evaluation, and real-world usage.

1. Adaptive Quiz Engine:

The adaptive quiz module dynamically generates questions based on a student's evolving ability level. The system begins with a fixed mix of easy, medium, and hard questions during the student's first attempt. Subsequent quizzes are generated adaptively based on the student's performance, using Item Response Theory (IRT) to estimate their ability score. The difficulty distribution is then adjusted accordingly, allowing the quiz to better match the student's proficiency level.

This ability-driven adaptation creates a personalized learning experience. During manual and system testing, it was observed that students who scored consistently on medium-difficulty questions were gradually exposed to harder ones, while students who struggled were offered more accessible quizzes to promote learning progression.

2. Unit-Based Quiz Feature:

The unit-wise quiz module supports syllabus-based revision by allowing students to choose specific units from the A/L Biology curriculum. It uses manually categorized questions from the dataset and returns them in a random but unique manner. Semantic filtering is applied to prevent repetition, ensuring each session feels fresh and comprehensive.

This feature proved especially useful in revision settings, with students appreciating the ability to target specific chapters such as Genetics or Cell Biology.

3. Quiz Results and Performance Feedback:

The result generation module provides detailed performance analysis after each quiz attempt. For adaptive quizzes, it shows the number of correct answers, overall accuracy, and ability level updates. For unit quizzes, results include per-question feedback and time spent per question.

This level of detail allows students to identify their strengths and weaknesses, contributing to more effective revision strategies.

4. Dashboard and Leaderboard Analytics

The analytics dashboard aggregates all student activity to provide actionable insights, including topic-wise accuracy, improvement over time, and leaderboard ranking. Charts are dynamically generated and update in real time as quizzes are completed.

This data not only motivates students to improve through visible progress but also helps educators identify struggling learners and intervene early.

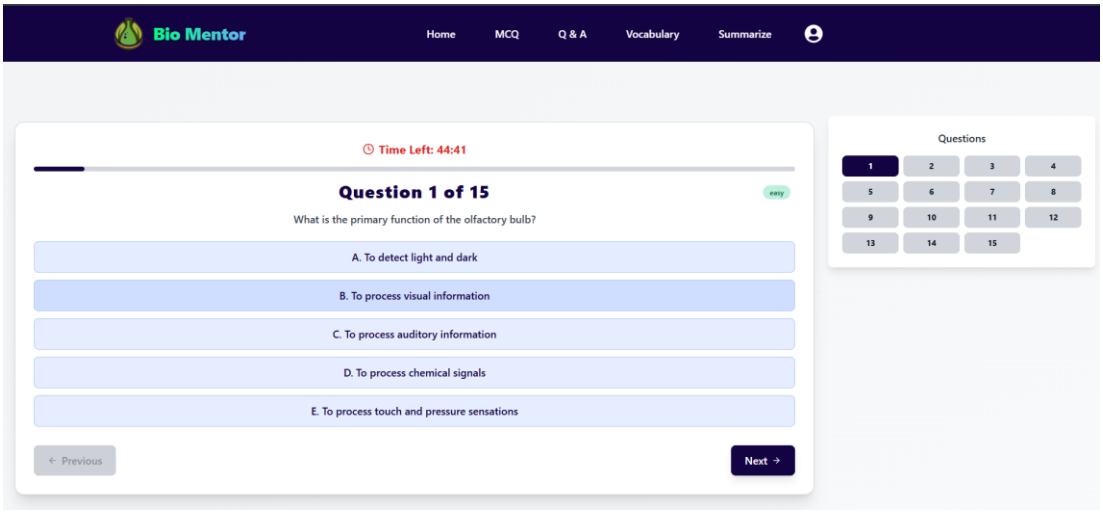


Figure 40: Adaptive quiz interface showing questions dynamically generated based on estimated student ability.

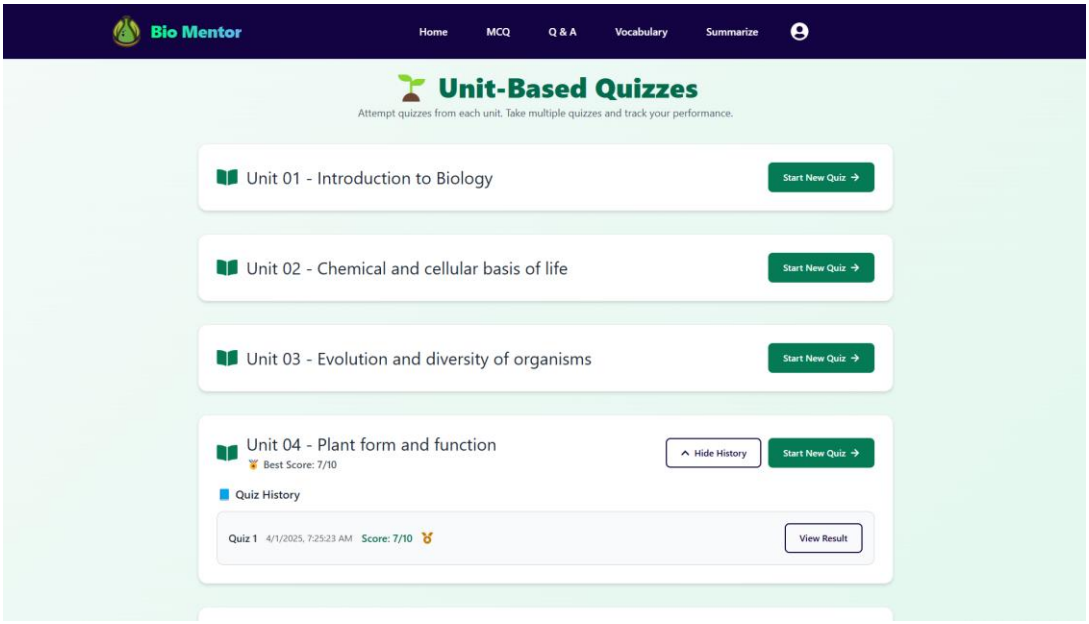


Figure 41: Unit-wise quiz feature allowing students to revise specific chapters from the A/L Biology syllabus.

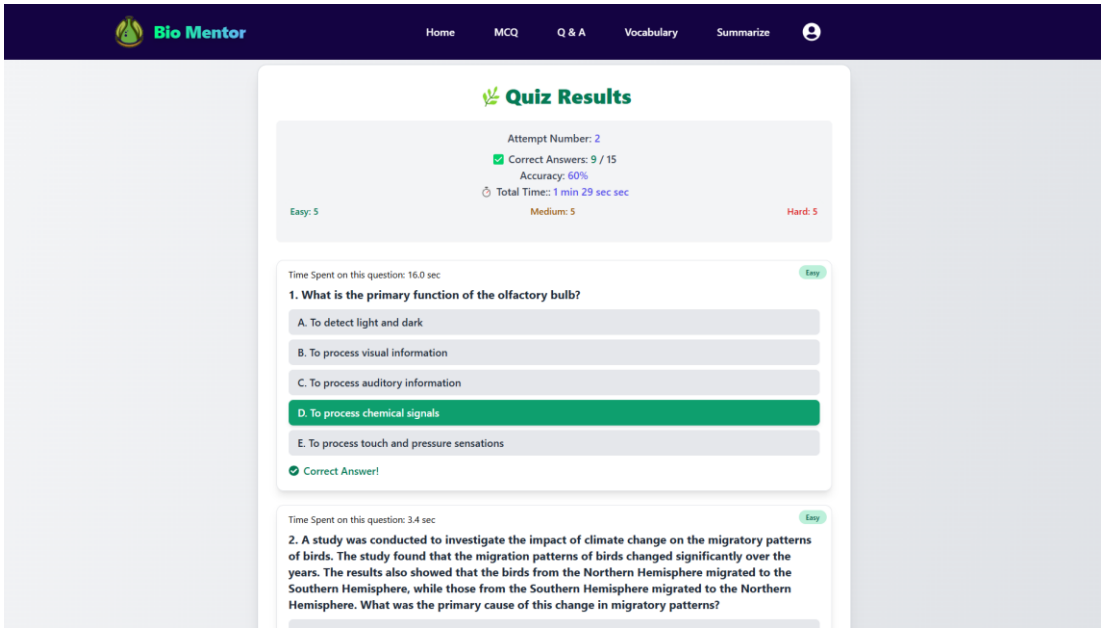


Figure 42: Detailed result screen

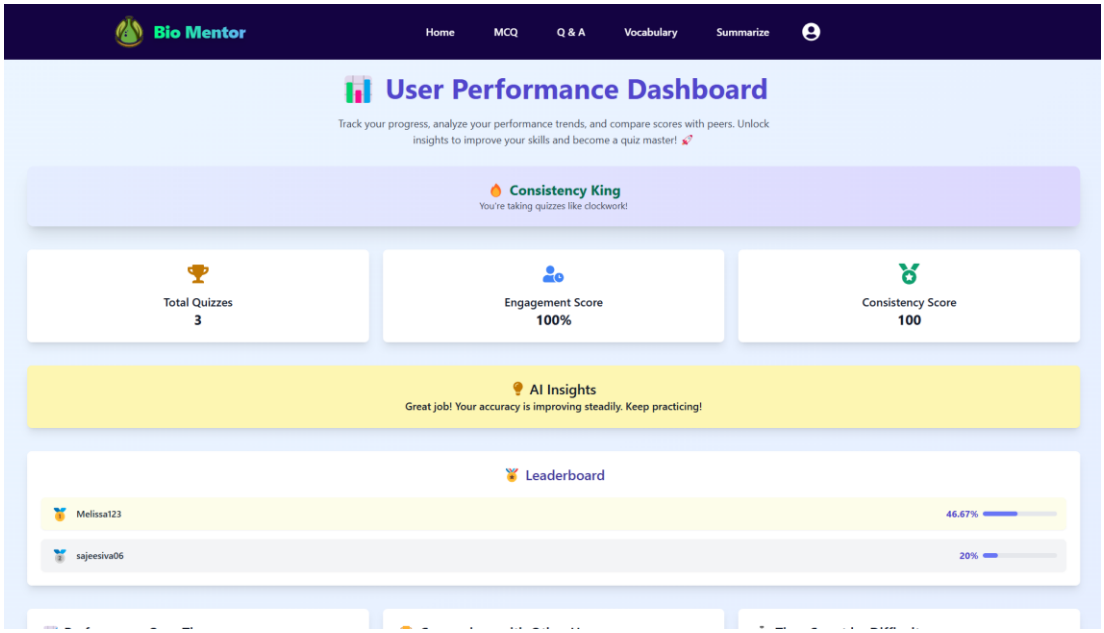


Figure 43: Student Performance Dashboard

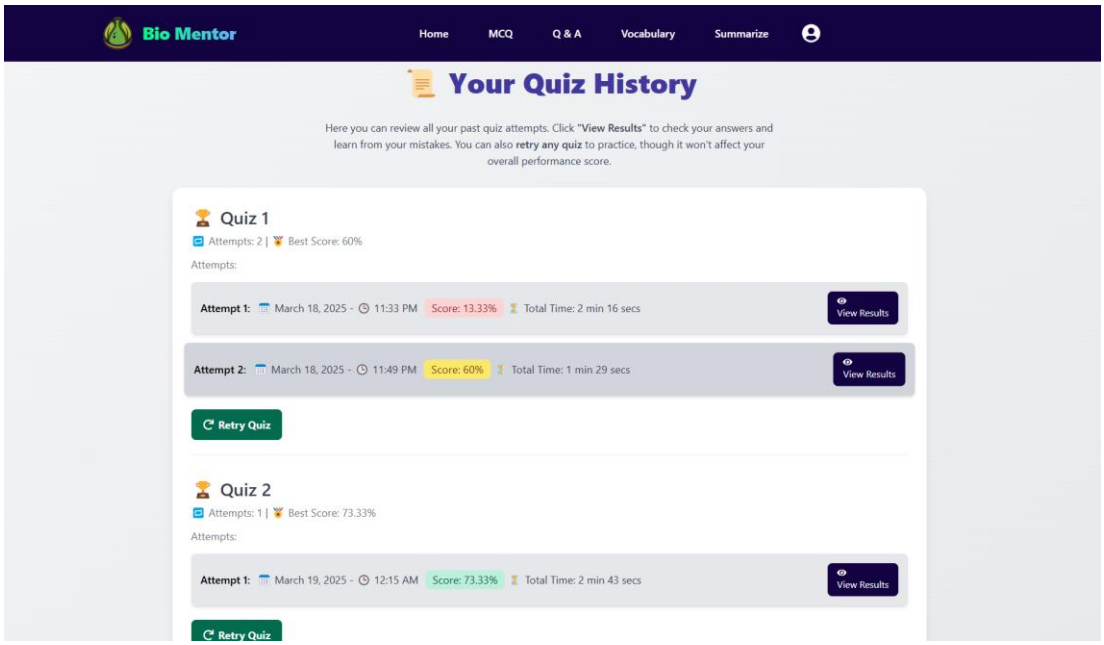


Figure 44: Adaptive Quiz History

Discussion:

The results of the system demonstrate the successful integration of adaptive learning methodologies into a domain-specific quiz platform. The use of IRT to guide difficulty progression ensures that students are consistently challenged at their level, avoiding the pitfalls of static assessment systems.

The unit-based quiz feature complements the adaptive module by allowing intentional and focused revision. Combined with the detailed feedback mechanisms, these modules create a closed feedback loop, wherein students learn from their mistakes, adapt, and reattempt as needed.

From an educational standpoint, this system addresses multiple layers of learning: knowledge recall, concept understanding, self-assessment, and strategic improvement. The dashboard further encourages student autonomy and peer-based motivation through the leaderboard.

Deploying the platform on Azure ensures scalability, uptime, and broad accessibility. Educators can now track student progress in real time, and students can receive data-driven, personalized learning experiences on demand.

4. FUTURE SCOPE

The successful development and deployment of the adaptive quiz platform for A/L Biology students sets the groundwork for various promising extensions and research opportunities in the domain of intelligent educational systems. As digital learning environments continue to evolve, this system can be enhanced across technical, pedagogical, and user experience dimensions. The following subsections outline key areas of potential future work and innovation.

1. Expansion to Other Subjects and Streams

While the current platform is designed specifically for A/L Biology students in the English medium, the underlying architecture and generation logic can be extended to other A/L subjects such as Chemistry, Physics, and Mathematics. Furthermore, support for Sinhala and Tamil medium students can also be integrated by fine-tuning the model on multilingual datasets and updating prompt structures.

2. Adaptive Learning Paths and Content Recommendation

The system currently adapts quiz difficulty based on performance, but future versions can include content-level adaptation, where study materials, video lectures, or targeted revision notes are recommended based on the student's weaknesses. These learning paths could evolve based on real-time performance data, ensuring holistic preparation beyond quizzes alone.

3. Teacher and Class-Level Analytics Dashboards

While the current system focuses on individual performance tracking, future improvements can introduce educator dashboards, allowing teachers to monitor collective classroom performance, identify at-risk students, and assign customized quizzes. Class-based leaderboards, topic heatmaps, and intervention triggers would enhance educator engagement with student progress.

4. Real-Time Quiz Feedback with Concept Remediation

Future iterations can include **r**eal-time corrective feedback during quizzes. When a student answers incorrectly, the system could display a short explanation or a related concept summary to encourage immediate understanding. This would turn assessments into learning opportunities and improve retention.

5. Enhanced IRT Modelling and Dynamic Question Weighting

The IRT model used for ability estimation can be further improved by integrating multi-dimensional IRT (MIRT) or Bayesian networks, which would allow modeling of multiple skill areas simultaneously. In addition, weighting questions based on historical difficulty or past performance trends could make adaptivity more precise.

6. Continuous Model Fine-Tuning with User Data (Federated Learning)

Incorporating federated learning could allow the system to fine-tune the LLM using anonymized student data without exposing sensitive information. This would make the quiz generator smarter over time, improving contextual relevance and diversity without compromising privacy.

7. Offline Access and Mobile App Integration

To increase accessibility, especially in regions with unstable internet connectivity, an offline or mobile app version of the system can be developed. Students could attempt quizzes offline, and results could sync when connectivity is restored. This would widen the reach of the platform and support equitable access.

8. Personalized Study Schedules and Revision Plans

Based on a student's quiz patterns, scores, and exam timelines, the system can be enhanced to generate personalized revision calendars or study plans. These plans could suggest which topics to focus on each day, helping students manage their time more effectively.

9. Integration with National-Level Exam Systems

In the long term, this system can be scaled and integrated into national e-learning frameworks to support state-level online assessments, digital mock exams, or continuous school-based evaluations. Collaboration with education ministries or institutions could standardize adaptive testing in classrooms.

In summary, the adaptive quiz platform holds immense potential to expand into a full-fledged intelligent tutoring system. By leveraging advancements in large language models, personalized learning, and educational data mining, future developments can transform the platform into a complete digital learning assistant that adapts to every student's pace, style, and goals. The combination of real-time assessment, adaptive content delivery, and performance insights positions this system at the forefront of the next generation of student-centered digital education.

5. CONCLUSION

In a rapidly evolving digital education landscape, the need for intelligent, responsive, and student-centered learning solutions has become more pressing than ever. This research focused on addressing this need by designing and developing an adaptive quiz platform specifically tailored for A/L Biology students in Sri Lanka. The system combines the power of large language models, item response theory, and semantic filtering to provide a personalized and effective learning experience that adapts to the learner's performance in real time.

The project began with the careful curation and preprocessing of a syllabus-aligned MCQ dataset, manually annotated with difficulty level tags. This dataset served as the foundation for both unit-wise and adaptive quiz modules. The quiz generation logic leveraged a fine-tuned LLaMA 2-Chat-Hf integrated with a Retrieval-Augmented Generation (RAG) mechanism to generate contextually diverse and syllabus-relevant MCQs. Adaptive behavior was enabled using Item Response Theory (IRT), which estimated learner ability and dynamically adjusted quiz difficulty, ensuring that each quiz challenged the user appropriately.

The system's backend was implemented using FastAPI and deployed on Microsoft Azure, where it was hosted as a continuously running service. The frontend, developed with React and Tailwind CSS, provided a clean, responsive, and interactive user experience across desktop and mobile platforms. Key features included real-time quiz generation, performance dashboards, quiz history, retry options, and leaderboard rankings.

Comprehensive testing — including unit testing, integration testing, and user acceptance testing — validated the robustness and accuracy of the system. Manual testing confirmed that quizzes were generated correctly, analytics were meaningful, and the platform responded well across a range of devices and environments. Feedback from real students highlighted the platform's ease of use, relevance to exam

preparation, and motivational features such as adaptive difficulty and performance tracking.

The successful deployment of the platform not only marked a technical achievement but also demonstrated the educational impact of integrating adaptive learning models into digital assessments. By offering students dynamic, ability-based quizzes and data-driven feedback, the system promotes deeper engagement, encourages continuous improvement, and supports focused revision strategies.

This research also laid a strong foundation for future growth, including subject expansion, personalized study recommendations, teacher dashboards, and broader institutional integration. The scalable and modular architecture ensures that the system can evolve alongside curriculum changes and user demands.

In conclusion, this research presents a significant step forward in enhancing digital learning experiences through intelligent quiz delivery. The integration of adaptive logic, natural language generation, and real-time analytics has created a platform that is not only technically sound but also pedagogically impactful. The outcomes of this work serve as a testament to the power of interdisciplinary collaboration — blending education, machine learning, and user-centered design — to create meaningful tools that support students on their learning journey.

REFERENCES

[1] S. Gopi, D. Sreekanth and N. Dehbozorgi, "Enhancing Engineering Education Through LLM-Driven Adaptive Quiz Generation: A RAG-Based Approach," *2024 IEEE Frontiers in Education Conference (FIE)*, Washington, DC, USA, 2024, pp. 1–8, doi: 10.1109/FIE61694.2024.10893146.

[2] D. R. CH and S. K. Saha, "Generation of Multiple-Choice Questions From Textbook Contents of School-Level Subjects," *IEEE Transactions on Learning Technologies*, vol. 16, no. 1, pp. 40–52, Feb. 2023, doi: 10.1109/TLT.2022.3224232.

[3] C. N. Hang, C. Wei Tan and P. -D. Yu, "MCQGen: A Large Language Model-Driven MCQ Generator for Personalized Learning," *IEEE Access*, vol. 12, pp. 102261–102273, 2024, doi: 10.1109/ACCESS.2024.3420709.

[4] N. M. Dhanya, R. K. Balaji and S. Akash, "AiXAM – AI assisted Online MCQ Generation Platform using Google T5 and Sense2Vec," *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, Coimbatore, India, 2022, pp. 38–44, doi: 10.1109/ICAIS53314.2022.9743027.

[5] P. K. Talupuri, M. G. Kumar, K. Lavanya, G. Chetan and N. P. Challa, "Smart MCQ Generation using KeyBERT and BERT Based Models," *2024 3rd Edition of IEEE Delhi Section Flagship Conference (DELCON)*, New Delhi, India, 2024, pp. 1–5, doi: 10.1109/DELCON64804.2024.10866615.

[6] W. Cui, Z. Xue, J. Shen, G. Sun and J. Li, "The Item Response Theory Model for an AI-based Adaptive Learning System," *2019 18th International Conference on Information Technology Based Higher Education and Training (ITHET)*, Magdeburg, Germany, 2019, pp. 1–6, doi: 10.1109/ITHET46829.2019.8937383.

[7] A. Virani, R. Yadav, P. Sonawane and S. Jawale, "Automatic Question Answer Generation using T5 and NLP," *2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, Coimbatore, India, 2023, pp. 1667–1673, doi: 10.1109/ICSCSS57650.2023.10169726.

[8] A. Saddish, P. Somaiah, V. Gambhir and S. J. Nirmala, "Generation of Multiple Choice Questions from Indian Educational Text," *2023 3rd Asian Conference on Innovation in Technology (ASIANCON)*, Ravet, India, 2023, pp. 1–7, doi: 10.1109/ASIANCON58793.2023.10270551.

[9] Y. L. P. Vega, G. M. F. Nieto, S. M. Baldiris and J. C. G. Guevara Bolaños, "Application of item response theory (IRT) for the generation of adaptive assessments in an introductory course on object-oriented programming," *2012 Frontiers in*

Education Conference Proceedings, Seattle, WA, USA, 2012, pp. 1–4, doi: 10.1109/FIE.2012.6462377.

[10] R. Shah, D. Shah and L. Kurup, "Automatic question generation for intelligent tutoring systems," *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*, Mumbai, India, 2017, pp. 127–132, doi: 10.1109/CSCITA.2017.8066538.

APPENDICES