BIOMENTOR – PERSONALIZED E-LEARNING PLATFORM FOR A/L BIOLOGY SUBJECT STUDENTS IN SRI LANKA

(LLM-based abstractive text summarization tool with voice output implemented in different software architectures)

24-25J-257
Project Final Thesis

Dharane Segar - IT21068478

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 A/L BIOLOGY SUBJECT STUDENTS IN SRI LANKA

(LLM-based abstractive text summarization tool with voice output implemented in different software architectures)

24-25J-257
Project Final Thesis

Dharane Segar - IT21068478

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

Department of Computer Science & Software Engineering

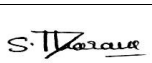
Sri Lanka Institute of Information
TechnologySri Lanka

April 2025

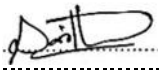
DECLARATION

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
Dharane.S	IT21068478	

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.



10/04/2025

Signature of the supervisor

Date

(Dr. Sanvitha Kasthuriarachchi)

10/04/2025

Signature of co-supervisor

Date

(Ms. Karthiga Rajendran)

ABSTRACT

The adoption of modern technology in education has fundamentally changed how learners consume and interact with content. Still, the sheer volume of unorganized learning materials, especially in scientific disciplines such as biology, complicates matters for learners working towards high stakes exams such as the Sri Lankan G.C.E. Advanced Level (A/L). This report introduces key component of **BioMentor**, an educational assistant designed to automate the creation of short, lecture-ready, structured outlines along with summaries and enable voice output in multiple languages. The system uses a fine-tuned Flan-T5 model with a Retrieval-Augmented Generation (RAG) framework to address the major challenges concerning accessibility, understanding, and personalization of educational content by employing Large Language Models (LLMs).

The two summarization modes that BioMentor offers are document-based summarization, which uses OCR and NLP pipelines to process large texts, such as Word, PDFs, and image-based content, and topic-based summarization, which uses FAISS-based similarity search to find semantically relevant content before producing summaries. The system also offers structured notes, as well as the option to translate them into Tamil and Sinhala using deep translation APIs. By turning generated content into audio, a text-to-speech (TTS) module improves accessibility even more.

The system was designed and developed in both monolithic and microservices architectures and evaluated on performance, deployment efficiency, and speed. Results indicated significant effectiveness in keyword recall, fluency, and structural consistency. The end result is a scalable, open-source, learner-centered assistant for multimodal and multilingual learning, which increases the convenience and effectiveness of online and blended learning environments.

Keywords: Educational Summarization, Large Language Models, Flan-T5, Retrieval-Augmented Generation (RAG), Structured Notes Generation, Multilingual Translation, Text-to-Speech (TTS), FAISS, OCR

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to our **supervisor, Dr. Sanvitha Kasthuriarachchi**, and **co-supervisor, Ms. Karthiga Rajendran**, for their continuous guidance, support, and encouragement throughout the development of this project. Their insightful feedback, patience, and dedication were instrumental in helping us shape and refine our ideas from inception to completion. We are also sincerely thankful to our **external supervisor, Ms. Nagalatha Thayaparan**, for her valuable input, suggestions, and support, which added great value to the progress and success of this project.

Furthermore, we extend our appreciation to the lecturers, assistant lecturers, instructors, and academic and non-academic staff of **SLIIT** for their support and assistance throughout the module. We are also grateful to our group members for their collaboration and teamwork. Lastly, we thank our families and friends for their unwavering support, motivation, and understanding, which helped us persevere through challenges and stay committed to completing this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviations	Description
SLIIT	Sri Lanka Institute of Information Technology
A/L	Advanced Level
GCE	General Certificate of Education
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CSS	Cascading Style Sheet
CSV	Comma-Separated Values
DOCX	Microsoft Word Document Format
FAISS	Facebook AI Similarity Search
gTTS	Google Text-to-Speech
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JS	JavaScript
LLM	Large Language Model
ML	Machine Learning
NLP	Natural Language Processing
OCR	Optical Character Recognition
BART	Bidirectional and Auto-Regressive Transformers
MVP	Minimum Viable Product
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
RAG	Retrieval-Augmented Generation
SDLC	Software Development Life Cycle
WBS	Work Breakdown Structure
TTS	Text-to-Speech

UAT	User Acceptance Testing
ROI	Return on Investment
HBQA	Historical Book Question Answering
PDF	Portable Document Format
DOCX	Document Open XML
PPT	PowerPoint Presentation
TXT	Text File
VS Code	Visual Studio Code
GPU	Graphics Processing Unit
UI	User Interface
UX	User Experience
CTA	Call to Action
MP3	MPEG Audio Layer-3
VM	Virtual Machine
CPU	Central Processing Unit
GB	Gigabyte
RAM	Random Access Memory
NSG	Network Security Group
HTTP	Hyper Text Transfer Protocol
AI	Artificial Intelligence
SSH	Secure Shell
RBAC	Role-Based Access Control
LMS	Learning Management System
CI/CD	Continuous Integration and Continuous Deployment

1 INTRODUCTION

1.1 Background Study and Literature Review

1.1.1 Background Study

The integration of technology in the learning environment has dramatically changed the learning experience, especially in the past decade. Automation, customization, and digital accessibility have enabled a basic paradigm shift in the conventional processes of the classroom for both students and instructors. Notably, machine learning (ML) and natural language processing (NLP) have been leading drivers in the development of intelligent educational systems that address the unique needs and learning styles of different learners.

Such a change is of particular importance in dense subjects full of content, such as biology, where students have to grasp higher-level concepts, technical terms, and extensive technical information. In the G.C.E. Advanced Level (A/L) biology syllabus in Sri Lanka, students experience increased pressure owing to stringent syllabi, examination-driven learning frameworks, and shorter preparation periods. In spite of the availability of online learning content, students commonly encounter issues related to unstructured content, lack of support for diverse languages, and the lack of tools supporting efficient summarization and review.

Summarization is an important element in the discipline of educational information management. It entails the condensation of long texts into brief, well-organized summaries that retain the key meaning and purpose. In such a capacity, it becomes extremely useful in e-learning platforms, where learners must assimilate large volumes of information in short times. Summarization enhances understanding by converting intricate scholarly content such as that in biology into more understandable and manageable formats.

Summarization techniques typically fall into two categories:

- Extractive summarization involves the identification and grouping of important sentences or phrases copied exactly from the source text. While it employs identical words, it may not have coherence in context.
- Abstractive summarization, however, generates new sentences that convey the meaning of the original text. Abstractive approaches leverage complex models such as BART and T5 to generate summaries that are contextually coherent and semantically complete [1]. Abstractive methods find greatest utility in educational settings where clarity, brevity, and flexibility matter most. Figure 1 shows two types of summarization.

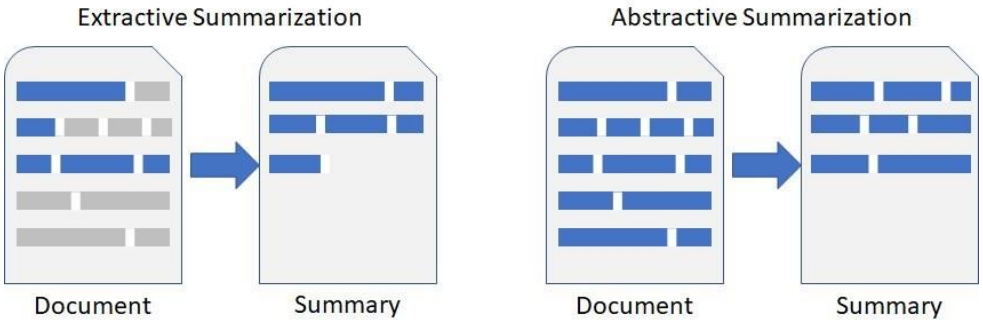


Figure 1: Summarization Techniques

To alleviate the constraints of manual note-taking and generalized study resources, this study presents BioMentor a whole-scale machine learning-based content dissemination system for Sri Lankan Advanced Level biology students. BioMentor produces organized, syllabus-oriented summaries automatically from the topics specified by the user, uploaded material, and graphical material. The system provides functionality for automated note preparation, real-time translation to Sinhala and Tamil, and text-to-speech (TTS) capability, thereby making the content more convenient for students with different linguistic needs and learning requirements.

System architecture utilizes a suite of technologies including document parsing, Optical Character Recognition (OCR), Retrieval-Augmented Generation (RAG) for contextually improved content, and grammar correction for polished outputs. Students can enter a subject or upload material in PDF, DOCX, or JPG formats for automated processing and summary generation.

By incorporating machine learning within the learning process, BioMentor seeks to deepen content comprehension, reduce preparation time, and promote inclusive learning. It is a move towards adaptive, learner-centered, and technology-enabled educational support systems empowering students from diverse socio-economic and linguistic backgrounds.

1.1.2 Literature Review

Abstractive text summarization is a popular topic under the Natural Language Processing (NLP) field, which deals with the contemporary problem of information overload. Unlike extractive methods that simply select sentences from the original text, abstractive summarization models generate new and grammatically correct sentences to summarize the essential meaning of the original content. The method is especially useful in educational contexts where attributes such as clarity, pertinence, and conciseness are of utmost importance.

Transformer-based models such as BART and T5 have taken the field a great distance forward. While BART employs a denoising autoencoder framework, T5 formulates all NLP tasks as text-to-text, providing flexibility and state-of-the-art performance [1], [2]. More recently, Flan-T5, a T5 fine-tuned on instruction-driven data, has been shown to be among the top in domain-specific summarization and factuality consistency both important in education tools [3].

To enhance the contextual relevance and accuracy of summaries, the concept of Retrieval-Augmented Generation (RAG) has been proposed. RAG fuses the generative capabilities of models such as BART and T5 with a retrieval mechanism that extracts semantically similar documents from a knowledge base. The hybrid approach provides improved factual accuracy and is especially helpful in academic and domain summarization, e.g., in the field of biology [4].

One other line of research is long-document summarization, a critical component in managing textbooks and academic texts. Token limits of transformer models are addressed by segmentation approaches, split the document into chunks, summarize them individually, and merge to create a coherent final summary [2], [5].

Topic-based summarization, often driven by BERT and similar retrieval-augmented architectures, has attracted widespread interest due to its capacity for delivering precise and focused summaries against user-specified questions, leveraging a variety of content resources, such as Wikipedia and news datasets [6].

In the bid to enhance accessibility, most educational summarization platforms embrace Text-to-Speech (TTS) technologies. gTTS and pyttsx3 are libraries that provide easy implementations of text to audio conversion, while complex TTS engines like Tacotron 2 and FastSpeech synthesize speech that is very close to human vocalization. This tool is particularly helpful for audio learners or those who have challenges in reading [7].

From a deployment perspective, there is a constant debate regarding the trade-off between monolithic and microservices architectures. Although monolithic systems provide reduced latency and simpler deployment, which is greatly advantageous for real-time applications, microservices provide modular development and improved scalability [8].

Real-world use cases, such as the Historical Book Question Answering (HBQA) project and learning question-answering systems realized on Flan-T5, validate the feasibility of such technologies. Integrating summarization, information retrieval, text-to-speech (TTS), and deployment frameworks into a single platform has been shown to improve user experience as well as learning outcomes [9], [10].

In summary, the literature supports the development of systems like BioMentor that integrate transformer-based models, RAG, long-document summarization, TTS, and large-scale architectures. These components are brought together to create a robust, learner-centric platform aligned with contemporary trends in educational technology.

1.2 Research Gap

Advanced text summarization tools have significant potential to enhance educational technology, particularly within e-learning platforms. However, several research gaps must be addressed to fully exploit their capabilities, especially for complex subjects like biology.

A primary research gap is the need for more effective audible summaries. Although text-to-speech technologies have advanced, many existing summarization tools do not incorporate high-quality, customizable voice output features. There is a need for research into developing voice output capabilities that are tailored to various learning styles, particularly for auditory learners who benefit from audible summaries in educational contexts.

Another challenge is the extraction of data from approved educational resources. Current tools often rely on general data sources, which may not always align with specific educational standards. Research is needed to develop methods for accurately extracting and summarizing content from government-approved resources, ensuring that summaries adhere to educational requirements and provide reliable information. This also extends to the generation of structured notes from these approved sources, enabling students to revise directly from syllabus-compliant material.

Customizable word count is another area requiring attention. Existing summarization tools may not offer sufficient flexibility for users to specify the length of summaries according to their needs. Research should focus on implementing features that allow users to adjust word counts dynamically while maintaining the quality and relevance of the summarized content.

Additionally, the handling of document uploads remains a challenge. While tools can process various document formats, there is a need for improved accuracy in text extraction, especially from diverse and complex document types such as scanned PDFs or Word files. Addressing this involves enhancing OCR (Optical Character Recognition) technologies and summarization algorithms to handle different document formats effectively.

Finally, there is a lack of comprehensive studies comparing different software architectures for implementing summarization tools. The performance, scalability,

and effectiveness of these tools can vary based on the chosen architecture. Research is needed to evaluate and compare different architectural approaches to identify the most effective solutions for deploying advanced summarization technologies.

By addressing these research gaps, this project aims to develop a robust summarization tool that integrates advanced audible summary features, accurate data extraction from approved resources, customizable word counts, and efficient document upload processing. This will enhance the tool's effectiveness in educational contexts and support diverse learning needs, including translation of notes into Sinhala and Tamil for inclusive and localized learning. Figure 2 shows competitive analysis while Figures 3 and 4 pictures the tools Grammarly and QuillBot.

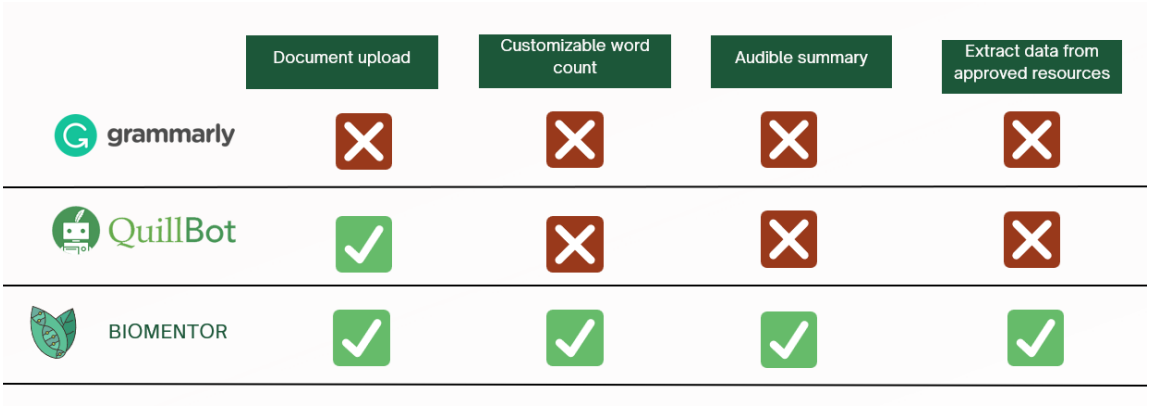


Figure 2: Competitive Analysis

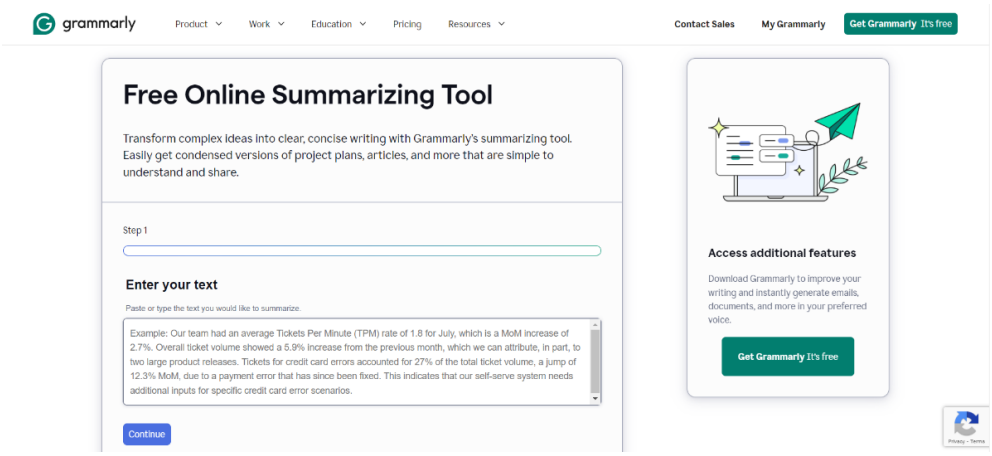


Figure 3: Grammarly – Summarization Tool

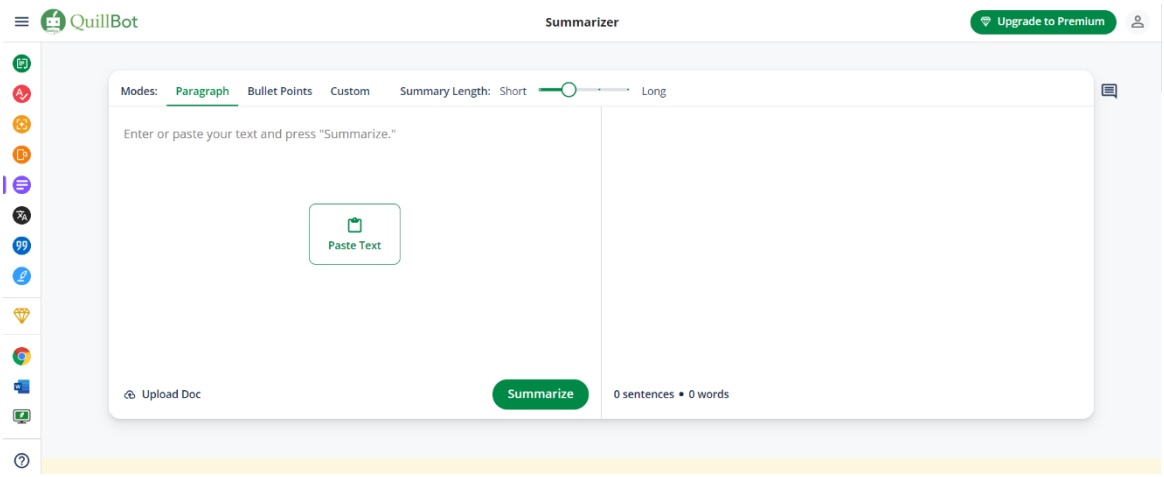


Figure 4: Quill Bot – Summarization Tool

1.3 Research Problem

The prevalence of digital learning has increased, particularly as a result of the COVID-19 pandemic. This shift has had a significant effect on how education is taught in classrooms, colleges, and on tests. Because it can be accessed from any location and gives students flexibility in terms of when and how they learn, online learning has many advantages. But it also has to deal with issues like maintaining student interest, assisting them in understanding difficult concepts, and ensuring that every student has access to the resources they require for learning. Biology and other subjects that require students to comprehend complex details, use visual aids, and thoroughly prepare for significant tests like the Sri Lankan G.C.E. Advanced Level (A/L) exams can be especially challenging.

Current tools are insufficient to meet the unique requirements of exam-oriented, domain-specific education, even with advancements in Natural Language Processing (NLP) and summarization platforms. Popular tools like Grammarly and QuillBot can't generate summaries that match the syllabus. Furthermore, multimodal inputs such as scanned PDFs, and tables embedded documents cannot be processed by these tools. Their practical applicability in educational settings that depend on a variety of intricate learning formats is limited by this gap.

Additionally, current summarization tools do not support audio. Few educational summarization platforms successfully incorporate TTS technologies, despite the fact that they have advanced to the point where solutions like TalkifyPy and neural-based speech synthesis models can now produce extremely natural outputs [7]. Students with visual impairments, reading challenges, or auditory learning preferences are thus still underserved. Rural students and non-English speakers are further marginalized in Sri Lanka and other multilingual nations due to the lack of content generation in their native tongues.

Additionally, a lot of summarization systems provide little control over the level of detail in the summary, resulting in either outputs that are too short and miss important details or summaries that are too wordy and undermine the goal of simplification.

These tools are less helpful for tasks like quick revision, class recaps, or focused study because they can't dynamically define word count thresholds or set summary detail levels.

The potential of transformer-based models, such as BART, T5, and Flan-T5, in educational contexts is still underutilized, despite their impressive performance in producing fluid, coherent summaries [1]–[3]. Notably, few systems use topic-level summarization frameworks [5] or Retrieval-Augmented Generation (RAG) [4] to filter data from reliable, syllabus-aligned academic sources. The majority of tools still extract and summarize generic or open-access content, which may not adhere to formal curriculum standards. This compromises factual accuracy and pedagogical reliability.

The effect of software architecture on the functionality of summarization tools is also poorly understood. Although microservices-based architectures are modular and scalable, they frequently add overhead and latency, which makes them less than ideal for real-time summarization in interactive learning settings. On the other hand, monolithic architectures typically offer tighter component integration and lower latency, despite their lack of flexibility [8], [9]. Comparative analyses of these architectural models in the context of real-time educational applications are still scarce, though.

The creation of a domain-specific, and multimodal educational summarization system that can provide syllabus-aligned, voice-enabled, and user-customizable summaries is thus the main research issue this project attempts to address. This system must support TTS outputs, allow for control over summary length, support a variety of content formats (such as scanned documents, images, and typed text), and guarantee content accuracy by integrating with reputable academic sources. Additionally, an architecture that maximizes real-time performance without sacrificing modularity must be used for deployment. By addressing these interrelated issues, BioMentor, the suggested solution, seeks to close the pedagogical and technological gaps that presently restrict intelligent educational support for A/L Biology students and support a more efficient, inclusive, and context-aware digital learning ecosystem.

1.4 Research Objectives

1.4.1 Main Objective

To develop and deploy an intelligent educational platform that supports local language translation and effective document handling that is suited to the Sri Lankan A/L Biology curriculum, while offering domain-specific, personalized, and accessible content delivery through text-to-speech capabilities, structured note generation, and real-time summarization.

1.4.2 Specific Objectives

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

- To develop an NLP-based summarization engine that can use topic-based and document-based inputs to produce A/L Biology summaries that are in line with the syllabus and pertinent to the context.
- Incorporating a text-to-speech (TTS) module to support visually impaired users and auditory learners by turning text summaries into audio output.
- To include dynamic word count customization, which would let users adjust the summary's length according to their learning goals or test-taking requirements.
- To use trustworthy document parsing and optical character recognition (OCR) methods for extracting content from a variety of formats, including scanned images, pdfs, DOCX, and PPTX.
- Producing clear, well-structured notes that facilitate revision and understanding.
- To improve inclusivity for learners who do not speak English by incorporating real-time translation capabilities for Sinhala and Tamil, thus facilitating multilingual access to notes.

- To guarantee that content is only retrieved and summarized from government-approved educational resources that are in line with the syllabus.
- To evaluate and contrast microservices and monolithic software architectures with regard to deployment complexity, scalability, latency, and appropriateness for real-time educational systems.
- To encourage a learner-centric design methodology while guaranteeing usability, accessibility, and flexibility in a range of Sri Lankan educational contexts.

1.4.3 Business Objectives

- **Boost the wellbeing and engagement of students:** Provide features that promote inclusivity and lessen academic stress, such as structured content delivery, audible summaries, and multilingual support, to improve the virtual learning environment. Encourage long-term retention and academic satisfaction by providing students with a more engaging and personalized learning environment.
- **Facilitate Scalable and Effective Instructional Provision:** Give teachers a platform for automated content creation and delivery to reduce manual labor and enable scalable distribution of curriculum-aligned content. This lessens the workload for teachers and guarantees uniform, superior academic support throughout all institutions.
- **Encourage Learning That Is Differentiated:** By letting students choose their preferred summary lengths, languages, and formats (text/audio), you can support individualized learning paths that cater to a range of cognitive styles and educational backgrounds.
- **Encourage the Development of Educational Technology:** Give educational institutions a competitive edge by implementing state-of-the-art machine learning and natural language processing (NLP) tools that facilitate intelligent voice output, translation, and summarization. Adopting learner-centered and AI-assisted teaching tools will put institutions at the forefront of digital education.
- **Increase Resource Utilization and Operational Efficiency:** Reduce the time and resources typically used for manual academic content preparation by automating document handling, note generation, and translation. This makes it possible for teachers and administrators to plan their classrooms more effectively and have greater access to resources.
- **Encourage Educational Equity:** By offering content in Tamil and Sinhala in addition to English, you can help close the gap for underprivileged and rural student populations by addressing language barriers and access restrictions.

2 METODOLOGY

2.1 Methodology

The methodology, which lays out the methodical processes, instruments, and strategies used to gather, process, and interpret data in accordance with research objectives, is the foundation of both academic and applied research. In projects involving machine learning (ML), natural language processing (NLP), and real-time system interactions, in particular, a clearly defined methodology guarantees that the research process is transparent, reproducible, flexible, and grounded in sound logic. A seven-stage system development lifecycle (SDLC) and Agile software development practices were combined in this study's hybrid methodology to handle the intricate, iterative requirements of creating BioMentor, an intelligent educational platform.

- **Agile Methodology for Research Development:** The main project management framework for this study was Agile, a methodology that was first created for software development but is becoming more and more popular in multidisciplinary research because of its adaptability, iterative structure, and evolution driven by stakeholders. Agile was the perfect choice for a dynamic project with changing pedagogical needs and evolving technical requirements because it places a strong emphasis on collaboration, modularity, and continuous feedback. Core modules like syllabus-aligned summarization, automated note-generation, multilingual translation, document parsing, and text-to-speech (TTS) were able to be seamlessly integrated thanks to the Agile model.

  In particular, the Scrum version of Agile was used, in which the work was broken down into smaller units called sprints, which are usually two to three weeks long. A major system component (such as OCR parsing, summary generation, or Sinhala voice synthesis) was the focus of each sprint. Continuous improvement and prompt issue resolution were made possible by regular sprint reviews, daily stand-up meetings, and retrospectives. The Scrum cycle used in this study is depicted in Figure 5, which shows how tasks moved through iterative feedback loops from planning to delivery.

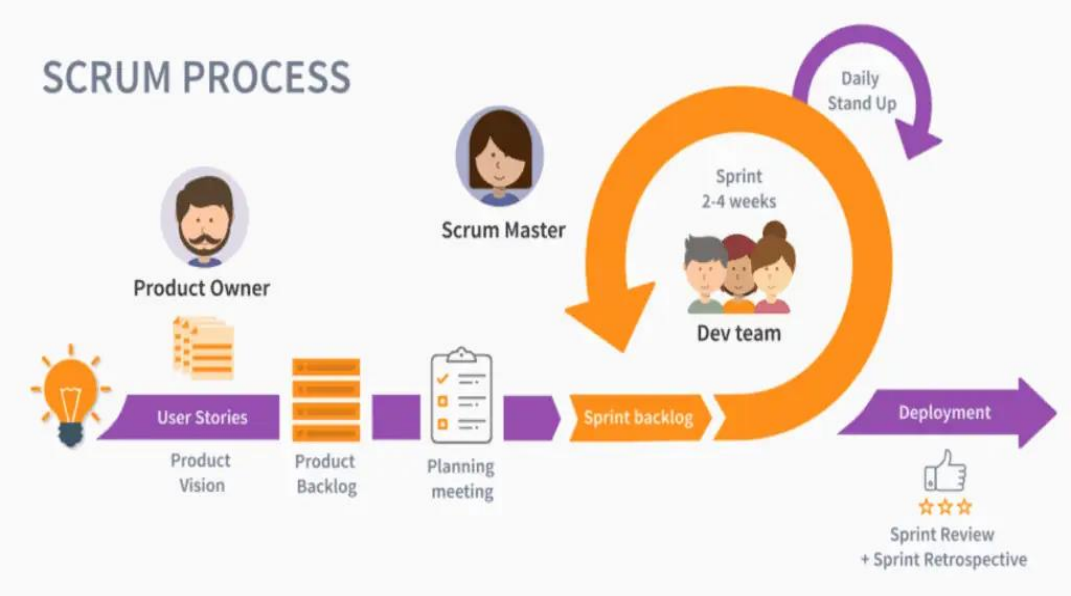


Figure 5: Agile Scrum Framework

- **Seven-Stage Development Framework:** Based on the ideas of Agile methodology, the research process developed over the course of a structured seven-stage process. Requirements collection, system design, deployment, testing, maintenance, and continuous improvement were all phases in this process. The results and input from the previous stage were used to inform each phase, resulting in a dynamic and adaptable development loop. The team was able to address issues as they arose, improve technical elements, and adjust to changing user needs thanks to this iterative structure. Agile's focus on adaptability and stakeholder participation was especially helpful in handling the project's multidisciplinary elements, which integrated cutting-edge machine learning methods with educational requirements. The development cycle encouraged methodical advancement while providing room for creativity, user input, and iterative improvement.

  Figure 6 shows the seven-stage development life cycle, and the rest of the methodology section will explain each of the phases of the life cycle.

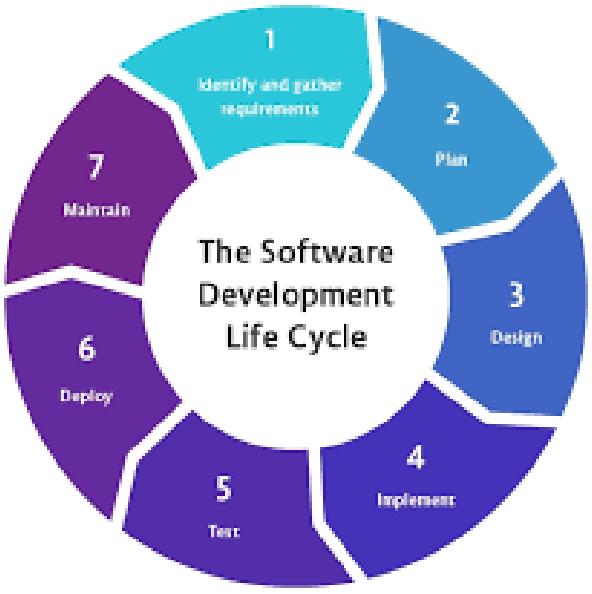


Figure 6: Seven Stage Development Life Cycle

- **Iterative Development and Collaboration**: Cross-functional cooperation and incremental development cycles were key factors in BioMentor' s success. To evaluate module performance and adjust to evolving needs, the interdisciplinary team which included software developers, NLP researchers, and biology subject matter experts convened biweekly review sessions. Jira, and GitHub Projects are examples of agile collaboration tools that made it easier to track issues, documentation, and code changes in real time. This iterative approach proved particularly helpful in improving Sinhala/Tamil translation alignment, resolving OCR noise problems in scanned documents, and fine-tuning NLP outputs.

- **Flexibility and Responsiveness:** Agile methodology's adaptability and responsiveness were essential in overcoming the ever-changing obstacles that arose during the BioMentor platform's development. Unexpected problems like differences in document formats, irregularities in OCR output, and limitations in language translation surfaced as the project progressed.

Because agile is iterative, the team was able to quickly adjust, improving multilingual support, streamlining data pipelines, and improving summarization logic without causing the project's overall timeline to be delayed. This ability to react quickly to changing technical and instructional requirements made sure the system continued to meet user expectations and educational objectives. Eventually, while preserving development continuity and responsiveness to user input, Agile principles made it possible to successfully integrate sophisticated features like notes generation, summarization, text-to-speech, and translation.

2.1.1 Feasibility Study/ Planning

The feasibility study examines the viability of designing and implementing BioMentor, a machine learning-driven system dedicated to summarizing and generating educational content specifically for Sri Lankan A/L Biology students. This system encompasses features such as document summarization, the generation of structured notes, multilingual translation of notes , and text-to-speech capabilities of summaries. The present section provides a comprehensive analysis of the technical, operational, and developmental feasibility of the proposed solution.

1. **Technical Feasibility**:
    - **Data Availability:** The effectiveness of the platform is grounded in its utilization of reliable datasets for the purposes of summarization and the generation of notes. Relevant biological content has been meticulously sourced from government-sanctioned educational materials, subsequently digitized and subjected to preprocessing through advanced natural language processing (NLP) pipelines. Additionally, the system accommodates user-generated files including formats such as PDF, DOCX, PPTX, and scanned images processed using custom-designed optical character recognition (OCR) and content extraction frameworks. To ensure both cultural relevance and alignment with educational curricula, manual data collection from sources within Sri Lanka was conducted.
    - **Hardware and Software Requirements:** System development and testing were performed on devices equipped with a minimum of 8GB RAM, a multi-core CPU, and optional GPU support to ensure optimized performance. The necessary libraries for this project include PyMuPDF, pytesseract, Tabula, gTTS, and FastAPI, all of which are compatible with Linux-based operating systems.
    - **Model Development**: The system utilizes a Retrieval-Augmented Generation (RAG) pipeline, aided by a fine-tuned Flan-T5-Base model and semantic retrieval techniques based on the FAISS framework. Furthermore, existing computational frameworks, including transformers,

SentenceTransformer, and language_tool_python, have been integrated to facilitate grammar correction, content summarization, and the generation of notes.

2. **Economic Feasibility**:

This study considered both development and operational costs. The total cost of the project is estimated at **LKR 25,610**, as detailed in Table 1.

Table 1: Cost Management

Type	Cost
Internet use and web hosting	5000 LKR
Google Colab Pro Subscription for Model Fine-Tuning	3000 LKR
Publication costs	12110 LKR
Stationary	5500 LKR
Azure Cloud Hosting (via student pack)	0 (Covered with student credits)
TOTAL	**25610 LKR**

- **Resource Allocation**: The development team comprised researchers possessing specialized knowledge in machine learning, web development, data preprocessing, and natural language processing. Responsibilities were allocated in accordance with individual strengths to enhance overall efficiency.
- **Return on Investment (ROI)**: The platform provides considerable return on investment by minimizing the manual work involved in note-taking, enhancing accessibility for learners, and fostering academic success through adaptable and inclusive content delivery.

3. **Legal and Ethical Feasibility**:

- **Data Privacy and Ethics**: User information is managed with a strong commitment to privacy regulations. Any data entered or interactions made on the platform do not retain personally identifiable information (PII), guaranteeing user anonymity. Whenever necessary, mechanisms for obtaining user consent are integrated.
- **Intellectual Property**: All third-party libraries and pre-trained models utilized are open-source or are covered by acceptable academic or research licenses. Any original software components created are kept under institutional ownership, with the potential for a future open-source release.

4. **Operational Feasibility**:

- **Data Collection and Processing**: The system effectively analyzes and condenses educational resources uploaded by users in various formats. OCR pipelines and document extraction tools showed strong performance in real-time applications.
- **Model Deployment**: The summarization engine is built with FastAPI and is designed to accommodate both monolithic and microservices architectural approaches. For the purposes of development and testing, a monolithic architecture has been chosen to facilitate smooth integration and reduce latency during real-time summarization and audio generation. Nevertheless, the system has been engineered with scalability considerations and is organized to allow deployment on cloud infrastructure in the future. This will provide greater accessibility, enhanced performance under simultaneous loads, and the capability to scale services independently as user demand increases.

5. Time/Schedule Feasibility:

- **Project Timeline**: A comprehensive timeline for the project was created, including essential milestones like data gathering, model training, feature incorporation, testing, and deployment. The iterative Agile methodology provided room for adaptability and ongoing enhancement. Figure 7 illustrates the Schedule Management plan.

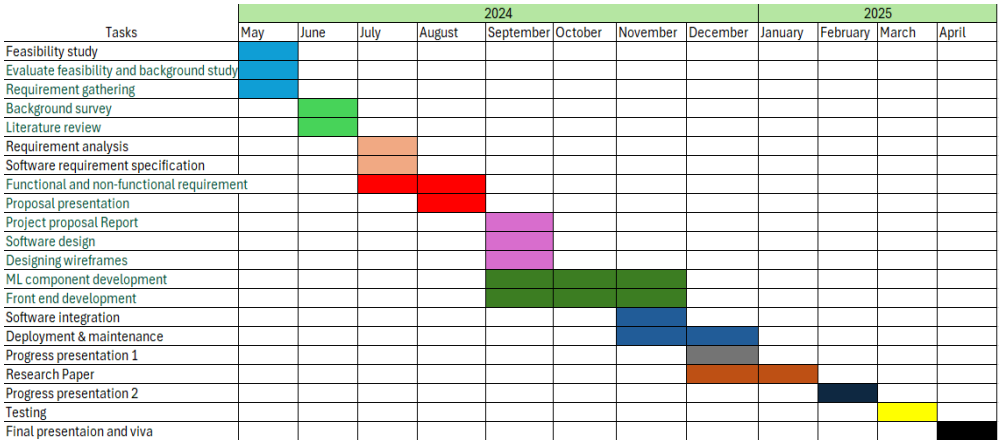


Figure 7: Gantt Chart (Schedule Management)

6. Social and Cultural Feasibility:

- **Acceptance and Impact**: The tool aims to tackle challenges encountered by students in Sri Lanka, particularly those in rural regions. Features such as translations in Sinhala and Tamil, voice summaries, and a user-friendly interface improve usability among local users and help diminish educational disparities.
- **Bias and Fairness**: The system is equipped with filters to prevent inappropriate content and maintains topic relevance. The design seeks to remain neutral both linguistically and academically.

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

Risk Management Plan

An in-depth risk management strategy was developed to foresee and alleviate issues concerning team availability, supervisor input, and stakeholder involvement.

Table 2 shows the risk management plan.

Table 2: Risk Management Plan

Risk	**Trigger**	**Owner**	**Response**	**Resource Required**
Illness or absence of project team members	Illness / Personal emergencies	Project Leader	• Inform supervisor • Redistribute tasks among team	Gantt Chart, Backup resources
Supervisor/Panel requests changes	Dissatisfaction with outcomes	Project Leader	• Apply changes promptly • Update Documents and resubmit	Product Backlog, Meeting Logs
Missed meetings by supervisor/panel	Conflicts, emergencies	Project Leader	• Notify team, reschedule, or conduct via alternate methods	Meeting Logs, Email
Technical failure during development	Software bugs, system crashes	Team	• Run tests regularly • Implement Backups and error handling	Testing Suite, Logs, Backup Mechanisms
Delay in deployment to cloud	Time constraints or configuration issues	Project Leader	• Allocate buffer time • Prepare contingency for demo/testing	Timeline Buffer, Hosting Access

Communication Management Plan

A strong communication strategy was upheld throughout the project to guarantee consistency among team members, supervisors, and stakeholders. Various communication methods such as email, documentation, phone calls, MS Teams, and WhatsApp were utilized for prompt updates.

Key Communication Objectives:

- Specific, adequate, and audience-targeted messaging.
- Timely feedback and decision-making support.
- Documentation of progress, changes, and approvals.

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)

Table 3 illustrates the communication management plan.

Table 3: Communication Management Plan

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,	Once at the start of the project	• Introduce project overview and research problem • Distribute components and outline responsibilities

		and expectations. Identify potential risks and agree on overall direction and novelty of the components.		• 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 Plan, and agree on team norms and workflows.	Once at the start of each major phase (Proposal, PP1, etc.)	• Present the finalized Work Plan • Outline communication flow and escalation paths • Reconfirm Quality 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	Weekly throughout the project	• Share status updates for each component

		technical challenges, and plan next steps. These meetings ensure continuous alignment between members and timely identification of blockers.		• 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 summary reports to ensure each team member is on track with their responsibilities.	Twice per week or as requested by the supervisor	• Present current component status • Highlight key deliverables 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 achievements are made. Ensures project stays aligned with academic requirements and timeline.	Monthly or at major milestone approval	• Debrief team performance to date • Record accomplishments 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 signoffs 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 Review Meeting	Supervisor, Co-supervisor, All Team Members	Conduct final evaluation of the project's outcomes and performance. Share individual and group reflections, lessons learned, and opportunities for future research or improvement.	Once at the end of the project	• Summarize system performance 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.)

2.1.2 Requirement Gathering & Analysis

The Requirement Gathering and Analysis phase constituted a pivotal element in the development of the BioMentor platform. This phase was dedicated to the meticulous identification and refinement of both functional and non-functional requirements essential for the implementation of a comprehensive system. This system aims to facilitate intelligent summarization, audio output of summaries, notes generation and translation of Advanced Level Biology content. Particular attention was given to ensuring alignment with the Sri Lankan G.C.E. A/L curriculum, thereby guaranteeing the relevance of the syllabus, linguistic inclusivity, and accessibility for a diverse range of student populations.

2.1.2.1 Functional Requirements

Functional requirements delineate the precise features and functionalities that a system must incorporate to achieve its designated educational goals. These requirements serve as a fundamental basis for the design, development, and validation processes of the system.

- **Stakeholder Input Approach**: Instead of formal interviews, the requirement identification process relied on two primary sources:
    - Feedback and opinions were gathered informally from current A/L Biology students and recent alumni, offering insights into their learning obstacles, study habits, and content preferences.
    - A brief structured survey was conducted among this group to evaluate the need for summarization tools, note-taking capabilities, and language preferences.
    - Additionally, ongoing consultations were held with an external subject expert, a certified A/L Biology teacher and G.C.E. examination paper marker, who assisted the development team in comprehending the syllabus scope, assessment formats, and requirements for textbook alignment.

Figure 8 illustrates the details of the user survey conducted to gather feedback on summarization needs, preferred features, and language preferences among A/L Biology students.

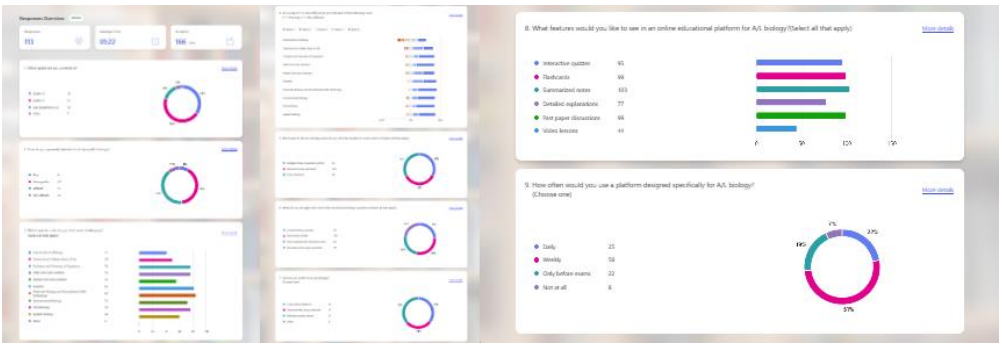


Figure 8: Survey details

In addition to the collection of requirements, this phase encompassed the manual compilation of a subject-specific dataset designed for the training and validation of the summarization system. The educational content was exclusively sourced from government-sanctioned materials, which included:

- Biology teacher guides
- Biology resource books
- Reference Books

All resources were supplied by the external supervisor to ensure complete adherence to the Sri Lankan G.C.E. A/L Biology syllabus, thereby guaranteeing that the generated outputs maintain educational accuracy and relevance. Figure 9,10 and 11 illustrate examples of some of the government-approved biology guidebooks and references utilized during the preparation of the dataset.

- **Identification of Key Functionalities**: Stakeholder discussions revealed the following essential features:
    - **Multi-Format Content Input:** The system should be capable of accepting documents in various formats including PDF, DOCX, PPTX, and image-based documents.
    - **Long Document Summarization**: It is imperative to implement an automatic mechanism that can dissect extensive documents and produce coherent summaries divided into logical sections.
    - **Topic-Based Summarization**: Users should have the ability to input specific biology-related keywords, enabling the retrieval of targeted summaries via a RAG engine enhanced by FAISS technology.
    - **Structured Notes Generation**: The automatic generation of systematically organized notes from the summaries is crucial for the purpose of revision.
    - **Multilingual Translation**: The capability to translate notes outputs into Sinhala and Tamil is necessary to accommodate a diverse range of linguistic requirements.
    - **Text-to-Speech (TTS):** A feature to convert English summaries into auditory format with controls for **play**, **pause**, **speed**, and **volume** will be beneficial for auditory learners.
    - **PDF Export**: Users must be able to download Summaries, English notes in PDF format, while translated notes should be provided in a readable textual format.
    - **Custom Output Settings**: It is essential to allow users flexibility to customize the length of summaries and specify their preferred language for notes output.

- **Architecture Deployment Strategy**: To ensure optimal performance and scalability, the summarization component will be implemented using both monolithic and microservices architectures. Each architecture will be evaluated across key criteria such as response time, resource utilization, and deployment

complexity. Following comparative analysis, the most effective option will be selected for final integration into the overall system.

- **Validation and Prioritization**: The functional requirements that were collected underwent a comprehensive validation and prioritization procedure to ascertain their feasibility, relevance, and alignment with the expectations of end-users. Input from A/L Biology students, educators, and the external supervisor was meticulously analyzed to enhance the functionalities, clarify ambiguities, and eliminate redundancies.

  Subsequently, a priority matrix was created to categorize each requirement into one of three distinct classifications:

  - **Essential** – Fundamental features necessary for the delivery of a minimum viable product (MVP).
  - **Desirable** – Value-enhancing features that contribute to user experience but are not critical to the primary functionality.
  - **Optional**– Advanced features considered for potential inclusion in subsequent iterations.

  This classification facilitated a more efficient development process and optimized resource allocation.

  After validation, the core system functionalities were articulated as user stories, illustrating how each feature would specifically benefit its end users. Figure 12 provides a visual representation of a selection of user stories derived from this exercise.

  The finalized primary functional requirements are delineated below:

  - Compilation and curation of domain-specific content from government-sanctioned A/L Biology resources pertinent to the Sri Lankan context.
  - Summarization of extensive academic documents (PDF, DOCX, images) utilizing Flan-T5 through segment-wise processing and integration.
  - Execution of topic-based summarization employing RAG and FAISS retrieval methodologies based on user-specified keywords.

- Generation of structured and coherent study notes derived from the resource materials.
- Translation of notes into Sinhala and Tamil to enhance accessibility for regional language speakers.
- Provision of text-to-speech (TTS) voice output for English summaries to support auditory learning.
- Facilitation of PDF downloads for English summaries and notes to ensure offline access.

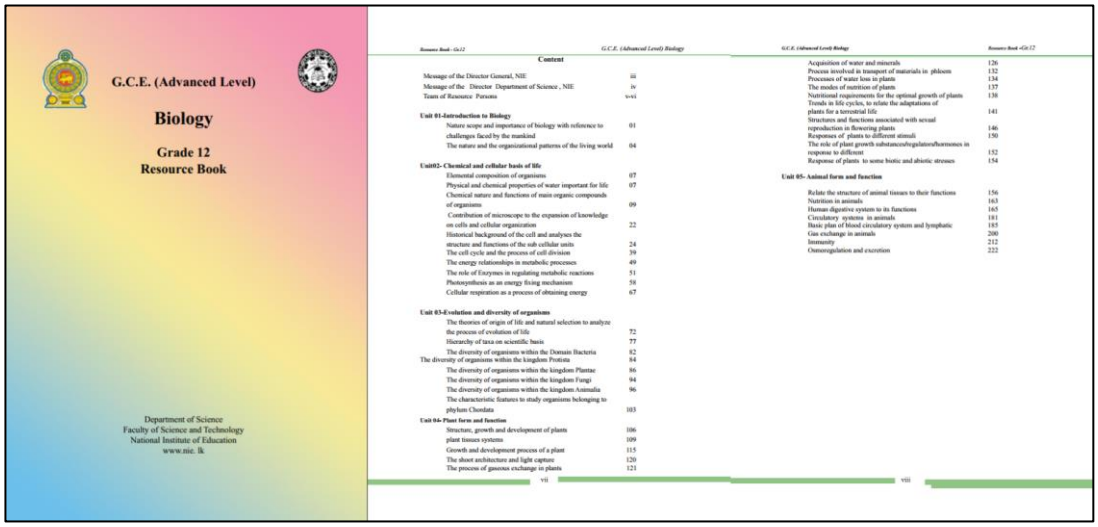


Figure 9: Grade 12 Resource Book

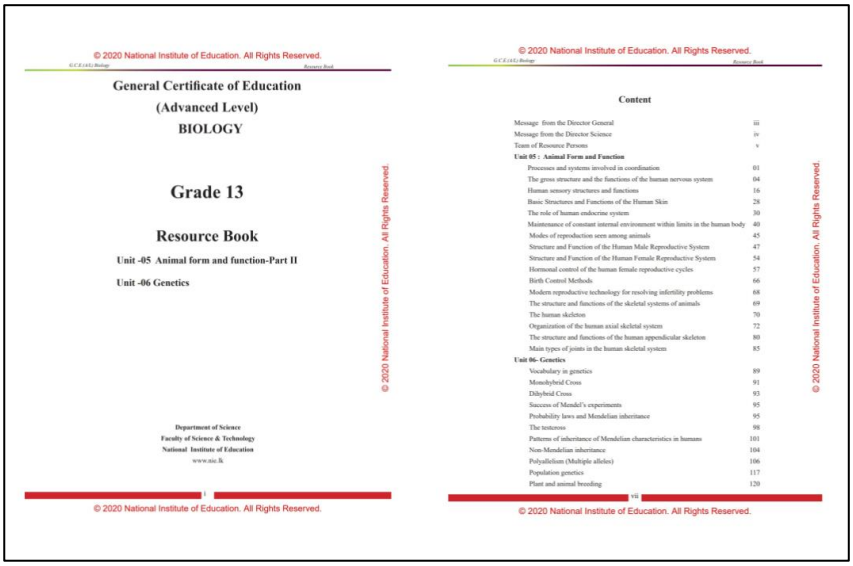


Figure 10: Grade 13 Resource Book

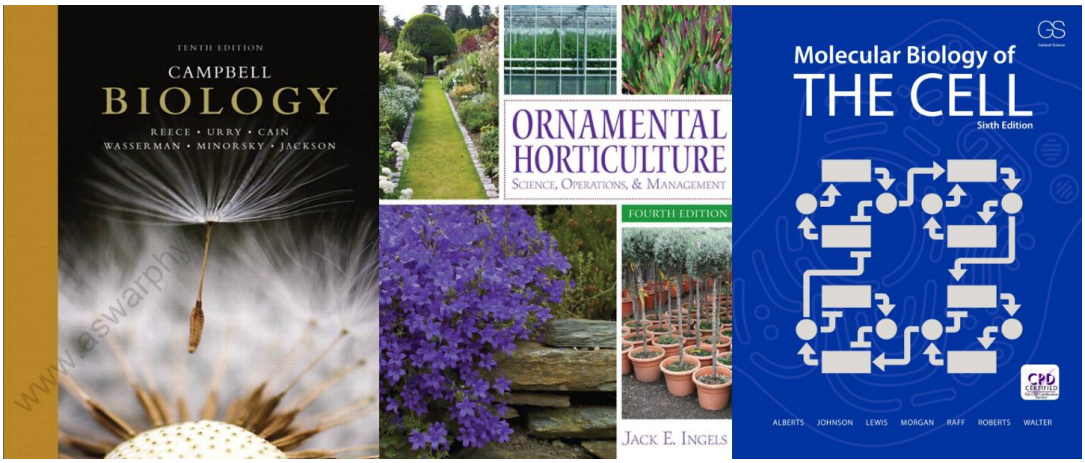


Figure 11: Other Reference Books

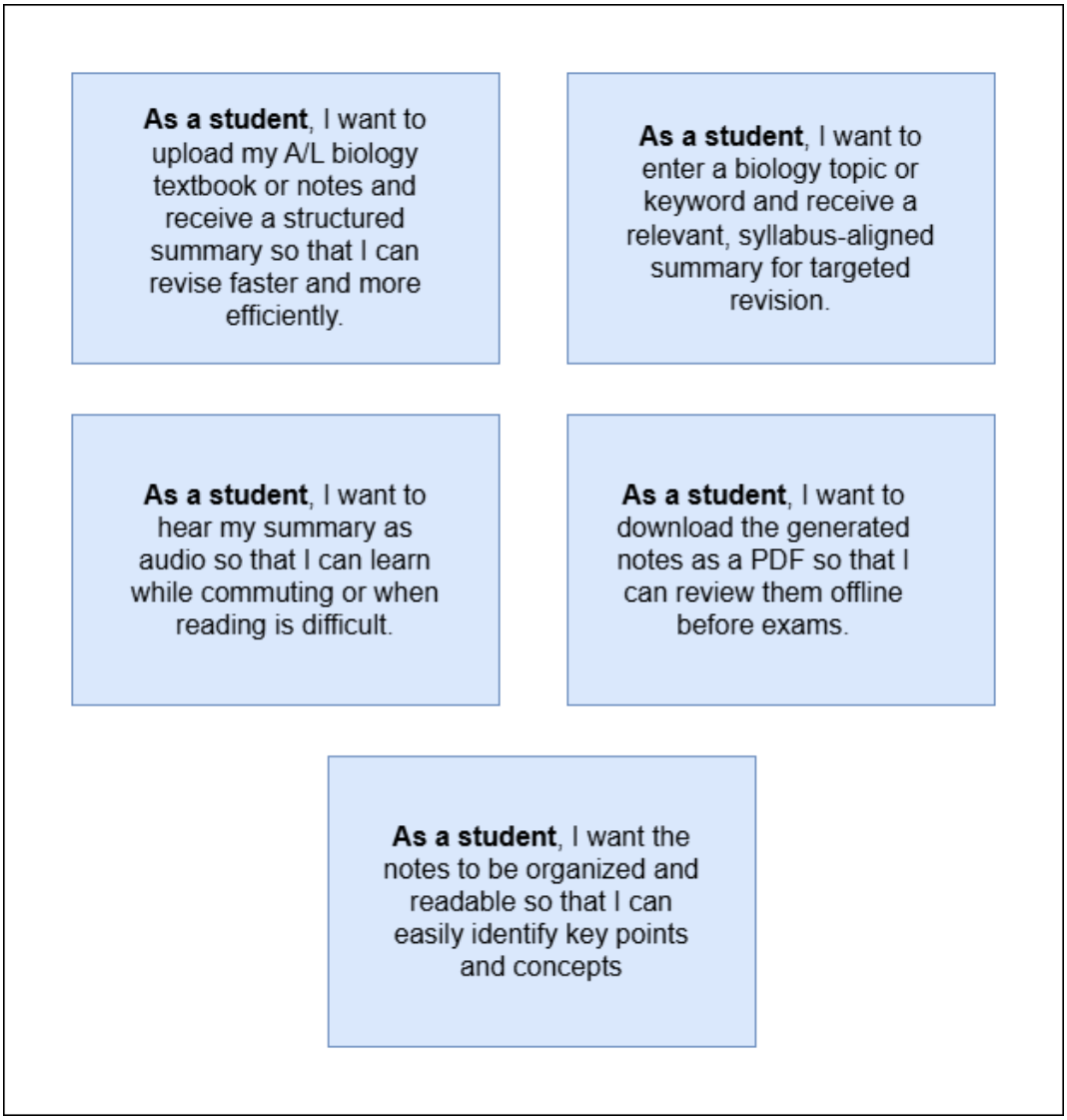


Figure 12: User Stories for Requirements

These user stories were used to shape sprint planning, prioritize development features, and validate that the platform meets the practical expectations of its primary user base, Sri Lankan A/L Biology students.

Accurately mapping user requirements in both the use case diagram and the sequence diagram is crucial for elucidating these requirements and establishing their prioritization. Refer to Figure 13 for an explanation of the use case diagram, while Figure 14 provides an illustration of the sequence diagram.

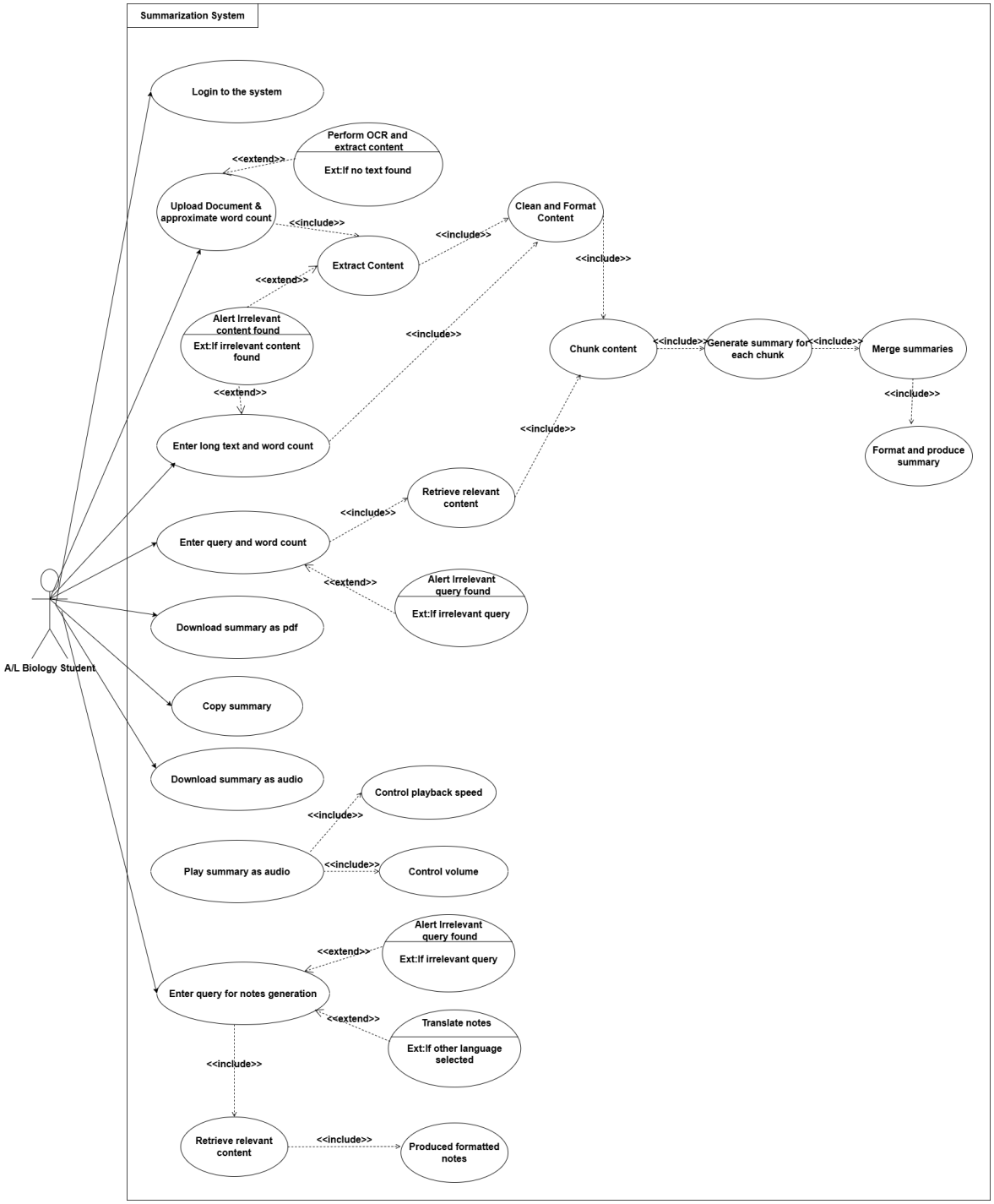


Figure 13: Use case Diagram

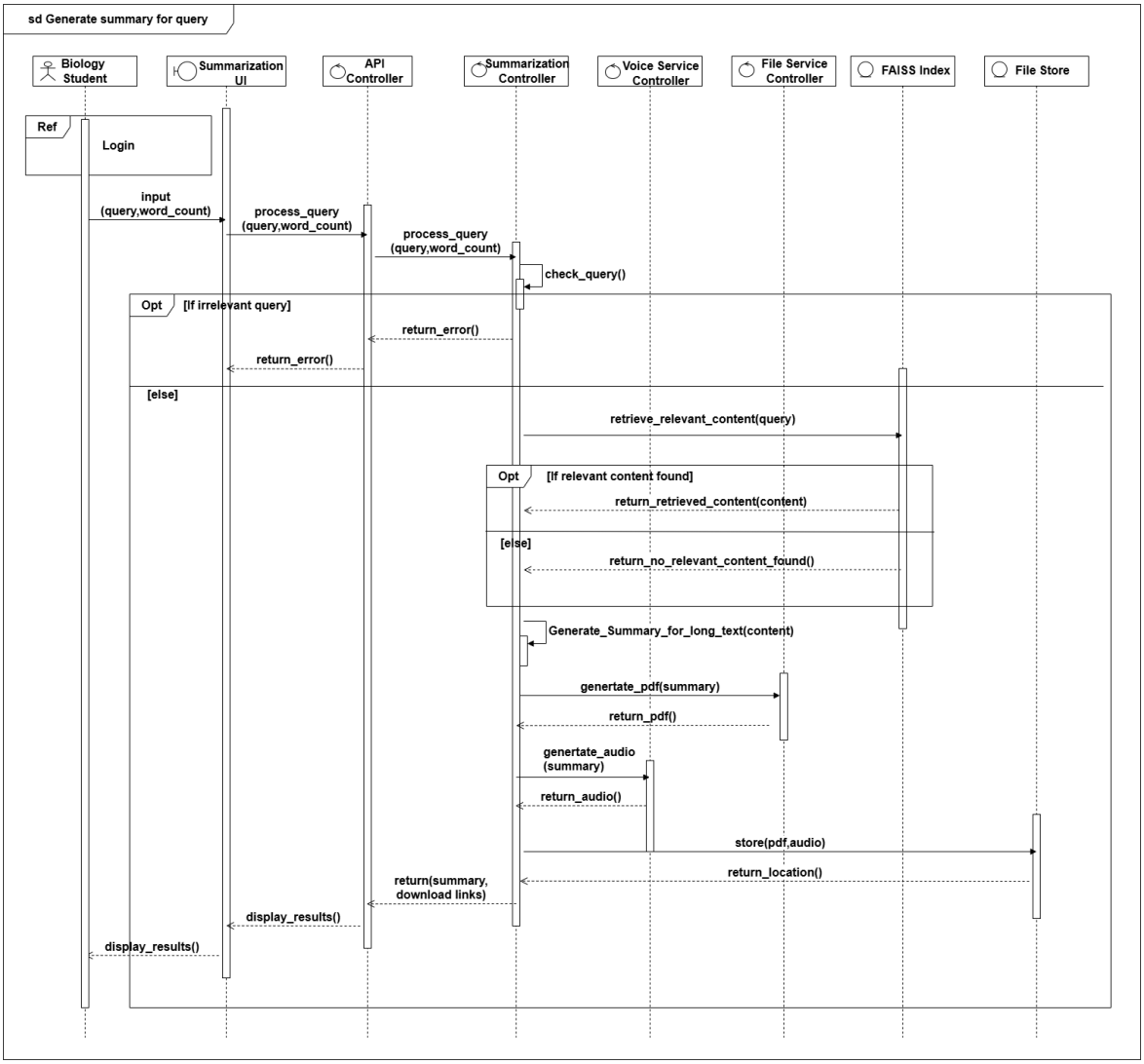


Figure 14: Sequence Diagram

2.1.2.2 Non-Functional Requirements

Non-functional requirements define the quality attributes, performance metrics, and constraints associated with the BioMentor system. These elements are critical in ensuring the platform's reliability, accessibility, and adherence to ethical and educational standards.

- **Feedback & Survey Approach**: Instead of using formal interviews, information was collected from students through an online survey and informal conversations with teachers and former A/L candidates. Special attention was given to immediate feedback, clarity of results, accessibility options, and the reliability of the system.

- **Performance Benchmarks**: The summarization engine needs to efficiently handle lengthy and intricate academic documents, producing coherent summaries that align with the syllabus in near real-time. The benchmarks are established as follows:
  - **Response Time**: Summary generation should be accomplished within 2 minutes for typical inputs.
  - **Document Handling**: The system must accurately support and summarize content from PDF, DOCX, and image-based files without losing formatting.
  - **TTS Support & Translation**: Although these features are secondary, they should not add more than 3 minutes to the output rendering time.

- **Content Quality & Relevance**: In the context of educational deployment, it is imperative that the generated summaries are rigorously aligned with the Sri Lankan A/L Biology syllabus. This alignment is guaranteed through the exclusive utilization of content derived from government-sanctioned teacher guides and reference texts, thereby ensuring adherence to the prescribed curriculum. The quality of the summaries is evaluated using established metrics, notably ROUGE-1. This criterion is critical for confirming that

essential information is effectively retained and articulated within the presentations.

- **Data Privacy & Processing Ethics**: The system is designed to handle all user-uploaded content either locally or within secure infrastructure, ensuring that no user data is retained following the processing phase. This protocol safeguards the confidentiality of student materials and examination-related content. Additionally, the absence of third-party APIs in the summarization process mitigates any potential exposure of sensitive data. The summarizer has been meticulously fine-tuned to prevent the generation of biased or factually incorrect information.

- **Usability and Accessibility:**
  - The interface is designed to be simple, mobile-friendly, and intuitive for students with minimal technical experience.
  - Audio summaries include playback controls like play, pause, speed, and volume adjustment, improving accessibility for auditory learners.
  - Users can customize the summary length and preferred language output for notes (English, Sinhala, or Tamil).

The Requirement Gathering and Analysis phase offered a thorough understanding of the BioMentor system's boundaries, essential features, and anticipated quality attributes. By conducting informal feedback sessions, a structured survey (Figure 12), and collaborating continuously with an external A/L Biology subject expert, a definitive and actionable set of functional and non-functional requirements was developed. These findings will inform the following phases of system design, implementation, testing, and deployment. Attention was given to ensuring that the summarization engine aligns with the Sri Lankan A/L Biology curriculum while also considering critical aspects such as usability, accessibility, real-time performance, and secure document management to fulfill the practical learning needs of students in a virtual environment.

The principal non-functional requirements identified are as follows:

- **Usability** – The design features a user-friendly interface, incorporating intuitive controls for customizing summaries, selecting languages, and managing audio playback.
- **Reliability** – The system is expected to maintain consistent performance, delivering accurate results across various document formats.
- **Availability** – High system uptime is crucial, particularly during peak academic periods, ensuring users can access resources when needed.
- **Accuracy** – The generated summaries must achieve high-quality ratings, reflected in strong ROUGE scores while remaining aligned with the curriculum.
- **Performance** – The solution should efficiently process large documents with minimal latency to accommodate the real-time demands of educational activities.

2.1.3 Designing

The Designing phase of the Software Development Life Cycle (SDLC) represents a crucial shift in the development of the BioMentor' s Summarization system, a platform specifically created for the intelligent summarization of biology material for Sri Lankan G.C.E. Advanced Level (A/L) students. This stage converts the conceptual ideas obtained during the requirements analysis into a detailed implementation plan, establishing system frameworks that are modular, scalable, and aligned with educational goals.

The primary focus of this design is on abstractive summarization, which allows the system to produce coherent, curriculum-compliant content from extensive educational texts. While additional features such as audio output of summaries, structured note creation, and multilingual translation of notes improve the overall learning experience, the summarization engine remains the key component.

System Architecture Diagram:

The architecture of BioMentor has been developed with a focus on modularity and natural language processing, with the goal of effectively converting intricate educational content into accessible summaries that align with curricula. The primary structure focuses on summarization, further supported by features such as document parsing, data retrieval, text-to-speech (TTS) capabilities, note creation and multilingual translation.

In Figure 15, the full architecture of BioMentor is illustrated, highlighting the input process from students to the final summarized results. This system allows for both document-based summarization (supporting uploads in PDF, Word, and PPT formats) and topic-driven summarization through keyword searches. This two-pronged approach addresses various learning styles and educational contexts spanning in-depth textbook summarization to focused, syllabus-related revision material.

Additionally, the system is engineered to automatically produce structured notes, arranging them into well-organized, revision-friendly formats. To promote linguistic

diversity, the notes can also be translated into Sinhala or Tamil, ensuring that students from different language backgrounds can benefit from localized content delivery. These outputs are subsequently offered as downloadable text or PDF files, depending on the selected language, thereby improving accessibility and user-friendliness for Sri Lankan A/L Biology students.

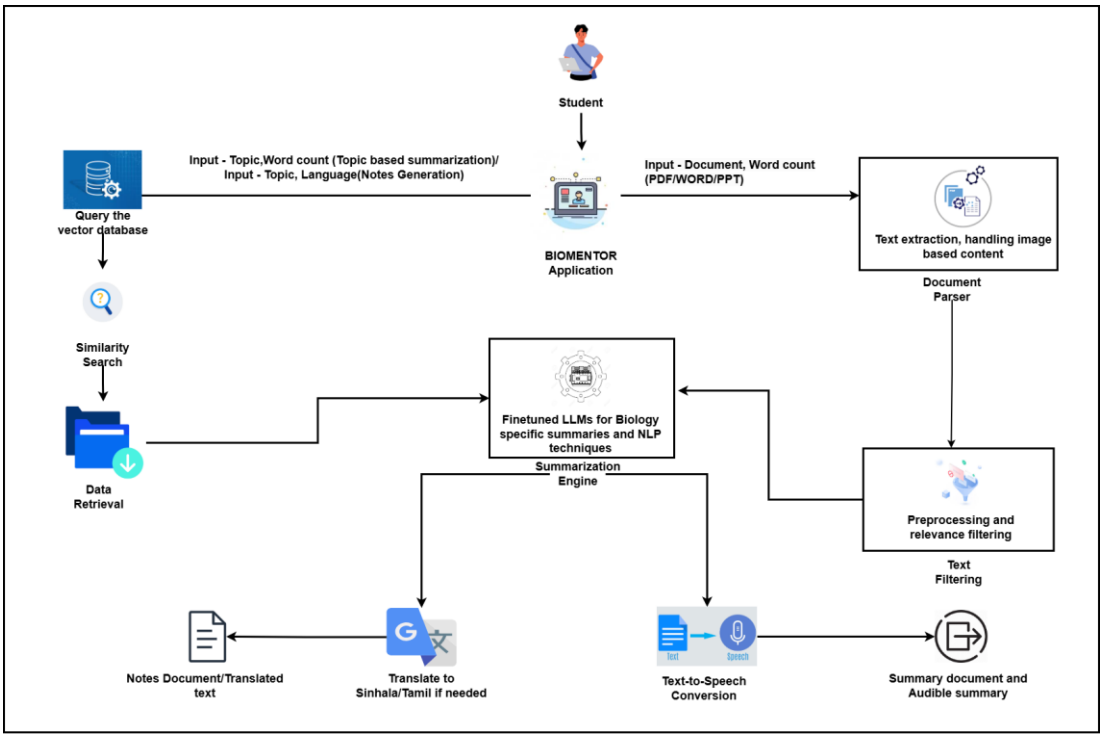


Figure 15: System Diagram

Key Architectural Components

1. User Input Interface:

Students engage with the BioMentor platform through a responsive web interface designed for optimal use on both desktop and mobile devices. This allows students to access educational resources conveniently, whether they are at home using a personal computer or on the move with a smartphone. The design principle emphasizes enhancing accessibility and user-friendliness in various real-world learning situations.

The interface supports two main methods of input:

- Document Upload: Accepts various formats such as PDF, DOCX, PPTX, and scanned image-based PDFs so students can submit learning materials in formats they typically utilize.
- Topic Query Input: Enables students to input biology-related keywords (e.g., “Cell Division”, “Photosynthesis”) to receive concise, syllabus-aligned summaries/notes derived from a curated knowledge base.
- Text Upload: Students can upload long, unstructured biology content

Other customizable features include:

- Preferred Summary Length: Allows customization of the word count for brief or comprehensive outputs.
- Audio Output Option (TTS): Supports auditory learning by providing text-to-speech conversion for English summaries.
- Output Language Selection: Permits the generation of notes in English, Sinhala, or Tamil, promoting linguistic inclusivity.

Student-Centered UI/UX Design

The frontend interface was created using a student-centered design methodology. Initial versions of the platform were showcased as low-fidelity mockups and assessed by current and former A/L Biology students. Feedback sessions concentrated on pinpointing issues related to navigation, layout clarity, and language accessibility. Based on their feedback, multiple revisions were made to enhance:

- Usability (clear workflows, intuitive input methods)
- Navigation flow (reduced clicks to obtain output)
- Responsiveness and layout for mobile devices

This continuous feedback process ensured that the interface stayed in tune with the actual requirements of the target users, making BioMentor a useful and accessible resource for Sri Lankan A/L students.

2. Document Parsing (Text Extraction + OCR):

When a student uploads a document to the system, it gets routed to the Document Parser module. This module is specifically designed to accommodate various types of educational documents typically utilized by A/L Biology students and educators, ensuring a flexible and inclusive approach to content input.

The parser is capable of handling the following input formats:

- Text-based Files, PDFs and Word Documents: These files are processed to retrieve well-organized textual content, maintaining key features like headings, bullet points, and subheadings.
- PowerPoint Presentations: The extraction is done slide by slide, allowing for systematic summarization of lecture or revision slides.
- Image based PDFs: Image-based content is analyzed using Optical Character Recognition (OCR) to transform visuals into functional text.

The aim of this module is to convert various formats into a uniform textual representation, ensuring that all content irrespective of its source is suitable for subsequent summarization and NLP activities. By achieving this, the parser ensures maximum accessibility to both digitized and non-digitized biology learning materials, which is vital for the inclusion of learners in rural or resource-limited settings.

3. Preprocessing & Relevance Filtering:

The extracted text undergoes a Preprocessing Layer that conducts the following:

- Cleaning: Elimination of non-text elements, including headers, footers, and unnecessary characters.

- Segmentation: Lengthy documents are divided into smaller, manageable units suitable for processing by transformer models.

- Relevance Filtering: Eliminates unrelated or unsuitable content by screening for explicit or off-topic terms, guaranteeing that only appropriate material is forwarded for summarization.

This phase enhances the input quality for improved output while reducing token overload during the summarization process.

4. Summarization Engine (Core Component):

the Summarization Engine employs a finetuned Large Language Model (LLM) specifically fine-tuned for A/L biology subject:

a. Document-Based Summarization

It utilizes a finetuned Flan-T5 model that has been trained on summaries corresponding to the Sri Lankan A/L Biology syllabus, enabling it to handle lengthy inputs and produce coherent summaries. This feature allows for customization in the length of the summary, providing either brief or in-depth outputs.

b. Topic-Based Summarization

This method uses a Retrieval-Augmented Generation (RAG) framework where keywords provided by students query a vector database (through FAISS). The relevant information is retrieved using similarity search techniques and then summarized by the Flan-T5 engine.

c. Notes Generation

The Notes Generation module utilizes the RAG (Retrieval-Augmented Generation) framework to acquire pertinent information based on keywords given by students. After pinpointing the most relevant details through similarity search in the FAISS-enabled vector database, the system processes this information to create structured notes.

These approaches ensure both abstraction and contextual relevance, making them particularly effective for summarizing intricate A/L Biology topics.

5.Text-to-Speech (TTS) Module:

To assist auditory learners and students with visual impairments, Summaries in English are sent to the TTS conversion.

Audio files feature functions for:

- Play / Pause
- Adjusting speed
- Controlling volume

The produced audio is integrated into the output interface or made available for download, promoting engagement through multiple learning modalities.

6. Translation Module (Sinhala/Tamil Support):

To enhance inclusivity and assist students who do not speak English, the Translation Module is utilized for the notes. The system translates the notes into either Sinhala or Tamil according to the user's choice.

Considering that this is a scientific platform, translations are performed to a certain degree, ensuring that biological terms are accurately represented. While maintaining the formatting and structure of the sentences, the translations may be simplified to ensure clarity and relevance. This functionality is especially useful for students in rural areas or those who attend Sinhala or Tamil-medium schools, allowing them to access educational information in their first language.

6.Output Delivery : Depending on the chosen features, BioMentor generates one or more of the following outputs:

- Summary Document (available for download in PDF format for English or text display)
- Structured Notes (available for download in PDF format for English or text display for Sinhala/Tamil)
- Audible Summary (audio file generated through TTS)

Outputs are produced in browser using the frontend interface and are available for download for offline access.

Document-Based Summarization Flow:

Figure 16 illustrates the flow of document-based summarization very clearly.

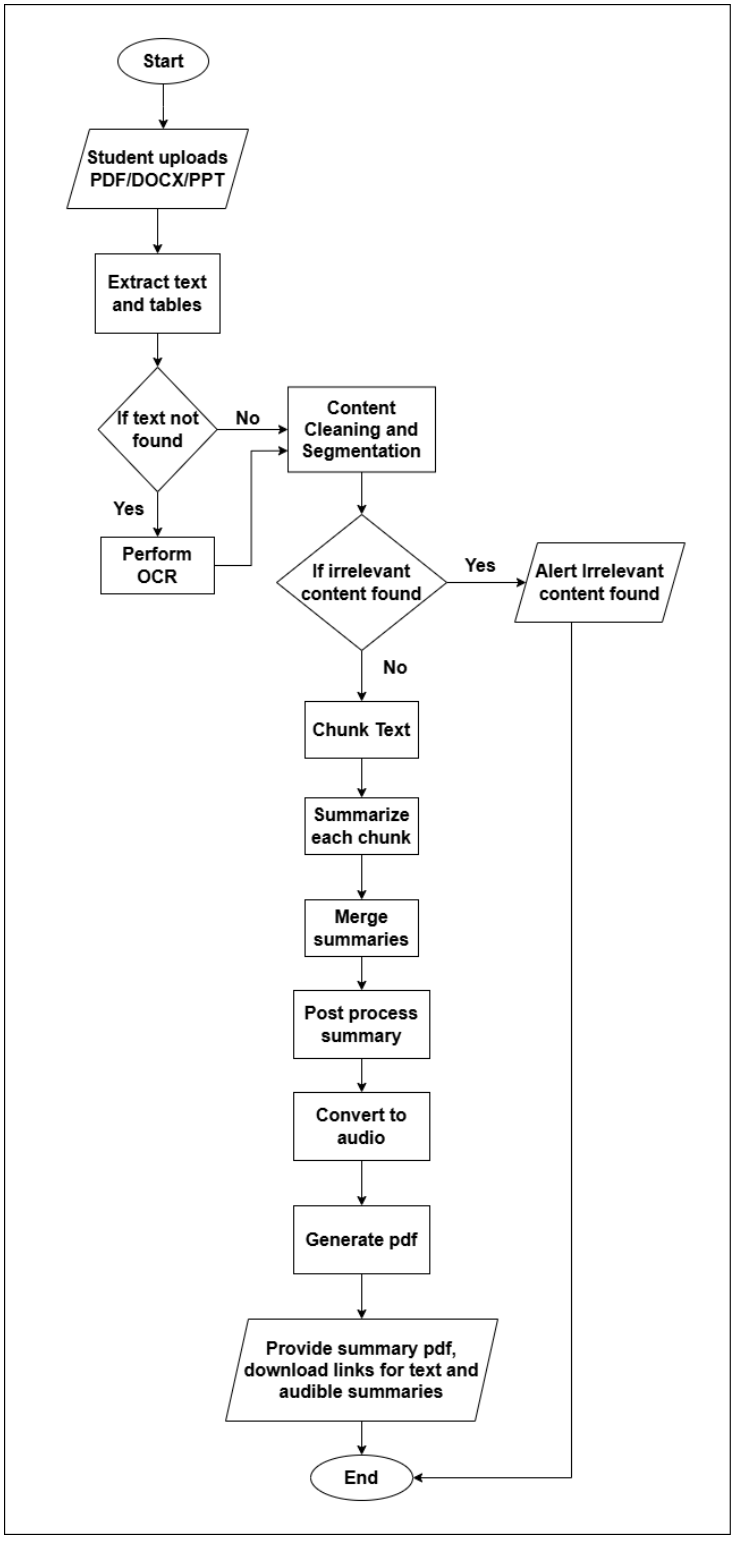


Figure 16: Flow of Document-Based Summarization

Topic-Based Summarization Flow:

Figure 17 illustrates the flow of topic-based summarization very clearly.

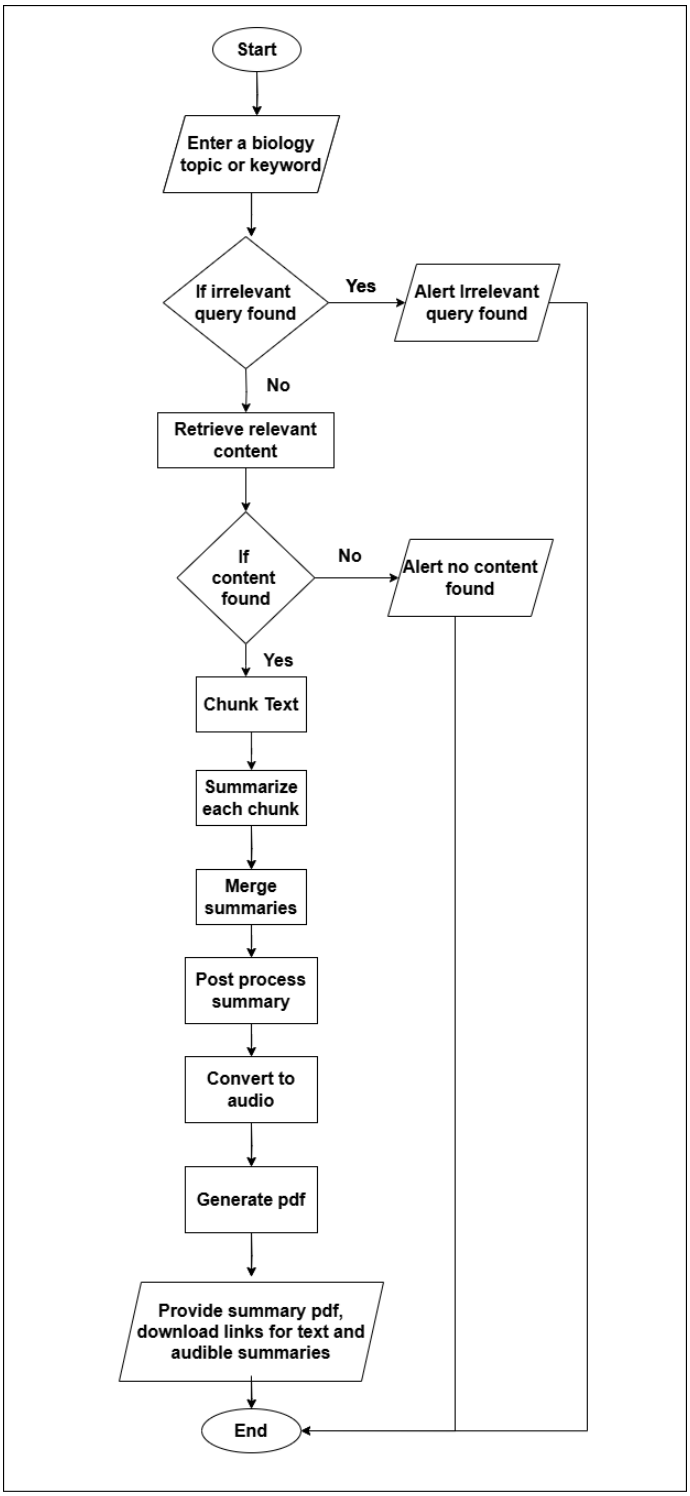


Figure 17: Flow of Topic-Based Summarization

Notes Generation Flow:

Figure 18 illustrates the flow of notes generation very clearly.

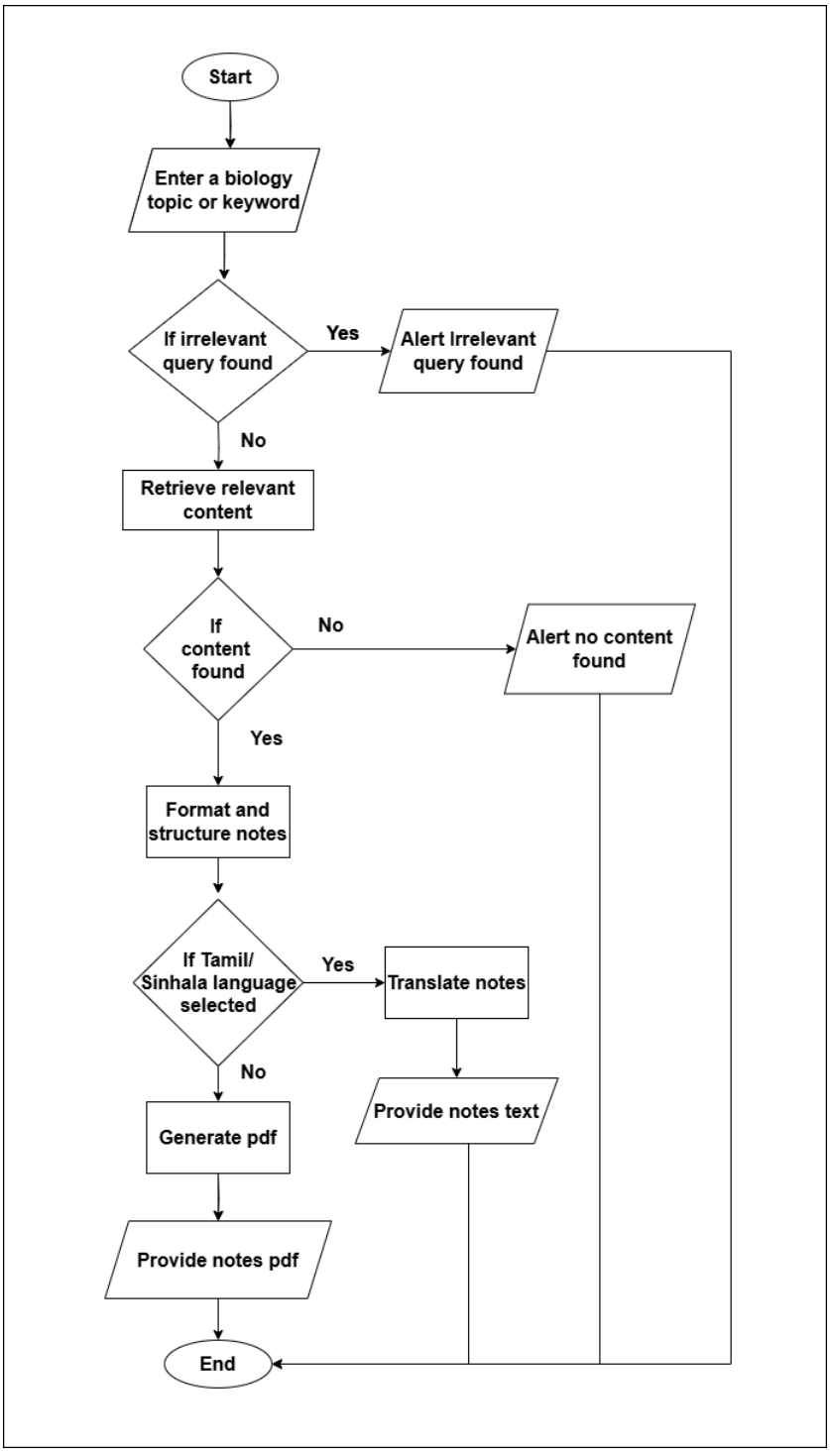


Figure 18: Flow of Notes Generation

The BioMentor system showcases a highly modular and flexible NLP pipeline, capable of evolving with future academic and technological requirements. To support both performance optimization and future scalability, the system was architecturally designed to accommodate two deployment strategies: monolithic architecture and microservices-based architecture.

During the design phase, this dual-architecture model allowed for logical separation of components such as summarization, parsing, translation, and TTS while still maintaining a unified flow. The monolithic design was structured for simplicity, ease of debugging, and tight integration with the front-end. In contrast, microservices were planned for independent deployment of each module (summarizer, notes generator, translator, TTS) via API endpoints, offering better scalability and fault tolerance.

However, the emphasis at this stage remained on defining modular component interactions, data flow, and functional boundaries, without committing to one architecture over the other. The actual performance comparison, deployment method, and integration decision were finalized during the implementation phase based on empirical results.

This forward-compatible design ensures that BioMentor can be scaled or restructured in the future with minimal disruption, supporting cloud deployment, containerization, or third-party integration if required.

2.1.4 Implementation

The Implementation phase of this research project signified a pivotal shift from the conceptual planning to the actual deployment of the BioMentor's Summarization system, a holistic solution for biology students. The platform was created to provide features such as document-based summarization, topic-based summarization using keyword input, structured note generation, text-to-speech (TTS) capabilities and translation into Sinhala and Tamil. This phase concentrated on transforming the clearly defined design architecture into a fully functional system, utilizing a mix of advanced machine learning techniques and contemporary development tools.

Task Breakdown and Project Management:

Before the actual process of implementation could be started, a detailed task breakdown and project management process was carried out. Jira was used to handle and coordinate the tasks in a streamlined way, breaking down the whole project into smaller, manageable subtasks. Each subtask was given a clear objective, timeline, and priority level, thus ensuring the overall process was streamlined and deadlines were always achieved. This ordered method ensured that the development process remained clear and enabled efficient communication and collaboration among team members. Leveraging Jira's agile functionality assisted us in depicting project progress, allocating tasks, and monitoring each phase of the implementation accurately. The project was divided into three primary sprints, which enabled the easy adoption of an iterative development model, where the feedback from a phase was utilized to enhance and modify further tasks. At the conclusion of every sprint, the deliverables underwent an evaluation process to verify that all the modules had been completely developed, tested, and integrated prior to proceeding to the subsequent phase. This methodical way of managing and developing tasks enabled us to preserve focus and flexibility during the implementation process, thereby guaranteeing the successful delivery of the platform's key functionalities, which comprised document parsing, text extraction, summarization, translation, and the user interface. Figure 19 pictures the Jira Board of our project management.

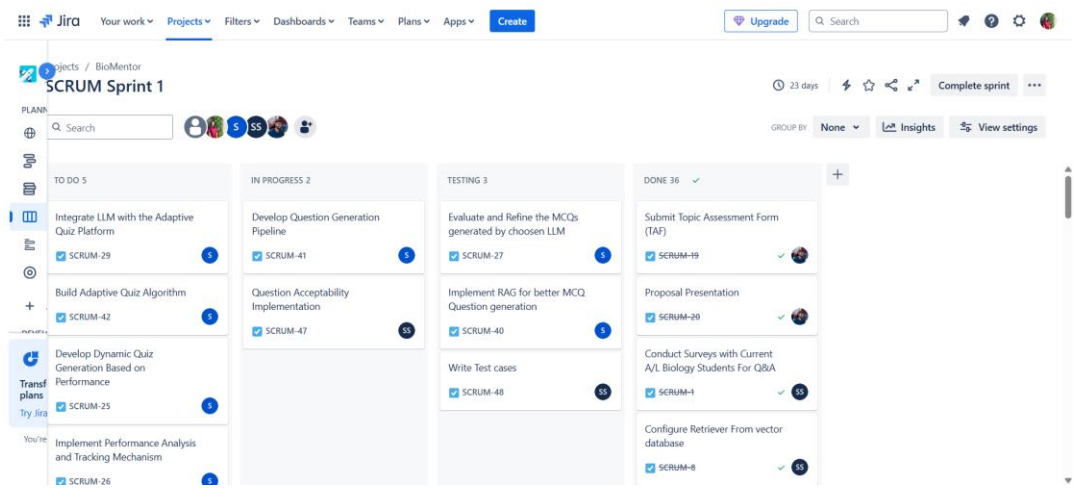


Figure 19: Jira Board

The Work Breakdown Structure (WBS) was essential in organizing and outlining the project tasks, decomposing the primary objectives into smaller, manageable parts. By classifying and ordering tasks, the WBS enabled more effective distribution of responsibilities among team members based on their skills. This organized method enhanced efficiency and simplified the tracking of progress, guaranteeing that every element of the project was systematically addressed. Additionally, WBS helped pinpoint potential delays early on, allowing for prompt solutions and ensuring that the project remained consistently focused on its objectives. Figure 20 demonstrates the WBS for summarization and other related modules.

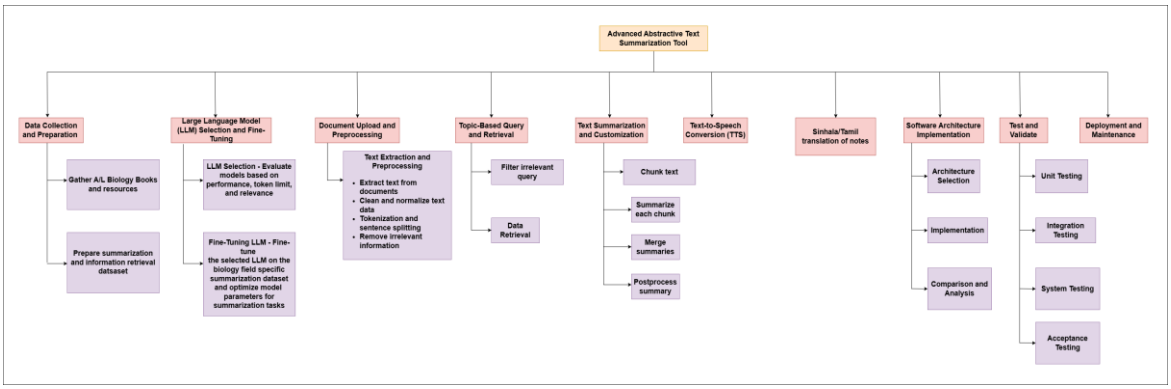


Figure 20: Work Breakdown Structure

Development Environment and Tools:

The implementation stage of this project primarily occurred within Visual Studio Code (VS Code), a flexible and lightweight Integrated Development Environment (IDE) that enabled efficient coding, debugging, and version control. The extensive support for Python in VS Code allowed for straightforward integration of various libraries and smooth execution of scripts. As the principal programming language for this research, Python provided the necessary versatility to realize the different components of the system.

For model development and fine-tuning, Google Colab was employed because of its powerful computational capabilities and compatibility with machine learning frameworks. The Flan-T5 model, which was specifically utilized for summarization tasks, underwent fine-tuning in Colab, taking advantage of GPU support for effective training and evaluation. This environment enabled the team to explore various model parameters and assess the system's performance on the biology-focused dataset.

Alongside these environments, several Python libraries and frameworks played a crucial role in the implementation. Tools such as PyMuPDF and python-docx were used for document parsing and text extraction, while the RAG (Retrieval-Augmented Generation) framework was implemented with FAISS to manage topic-based summarization. The deep_translator library facilitated translation, providing multilingual support for Sinhala and Tamil in notes generation. For text-to-speech capabilities, Google’s TTS API was incorporated to transform English summaries into audible formats. All these elements were effectively integrated into the BioMentor platform, which was created using the previously mentioned tools and libraries.

2.1.4.1 Data Collection Script for Summaries

Here are the main steps involved in this procedure:

1. Loading the Dataset: The initial step involves loading the dataset file titled bio_summary_keywords.csv through the pandas library. The encoding is specified as ISO-8859-1 to manage any special characters or formatting issues.

2. Cleaning Columns: An irrelevant column (Unnamed: 3) is eliminated to focus on important fields. The primary columns utilized for summarization include:
    - Long Text: Comprehensive biology content.
    - Summary: Summaries that are either human-created or generated.
    - Keywords: Terms linked to the content.

3. Text Normalization: A function called standardize_text() is defined in the script to clean and standardize the textual data:
    - Transforms text to lowercase.
    - Eliminates special characters and non-alphanumeric symbols.
    - Reduces excessive whitespace and formatting irregularities. This function is applied to both the Long Text and Summary columns.

4. Removal of Duplicates: Rows that have identical pairs of Long Text and Summary are eliminated using the drop_duplicates() function to ensure uniqueness and mitigate model overfitting.

5. Checking for Missing Values: The script inspects essential columns for null values. It determines if there are any missing entries in the Long Text, Summary, or Keywords columns, ensuring that only complete data is processed.

6. Final Adjustments and Export: Following the cleaning process, only the Long Text and Summary columns are retained. These are renamed to lowercase (long

text, summary) for uniformity. The final cleaned dataset is saved as bio_summary_final.csv.

The processed and refined data serves as a high-quality input for training and evaluating the summarization model. Figure 21 illustrates the implementation of the preprocessing logic in Python using pandas and regex for text cleaning.

```
# %%
import pandas as pd

# %%
# Load the dataset with a specified encoding
data = pd.read_csv('biology_summarization_example.csv', encoding='ISO-8859-1')

# %%
# Display first few rows
print(data.head())


# %%
print(data.info())  # Check data info

# %%
print(data.describe())

# %%
#printData Columns
print(data.columns)

# %%
#Drop the unnamed unnecessary column
data = data.drop(columns=['Unnamed: 3'])

# %%
#print Data Columns after the deletion
print(data.columns)

# %%
#Check for null values
data.isnull().sum()

# %%
#Remove unwanted characters, space, special characters
import re

# Function to standardize text: convert to lowercase, remove irrelevant symbols
def standardize_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove irrelevant symbols (e.g. unusual bullet points)
    text = re.sub(r'[¦]', '', text)
    # Remove any remaining special characters (e.g., non-alphanumeric symbols)
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    return text

# Apply the function to standardize the 'Long Text' and 'Summary' columns
data['Long Text'] = data['Long Text'].apply(standardize_text)
data['Summary'] = data['Summary'].apply(standardize_text)

# Display a sample to verify the standardization process
data[['Long Text', 'Summary']].head()

# %%
# Check and remove duplicates on columns 'Long Text' and 'Summary'
data_cleaned = data.drop_duplicates(subset=['Long Text', 'Summary'])

# Display the number of rows before and after removing duplicates
original_count = data.shape[0]
cleaned_count = data_cleaned.shape[0]

original_count, cleaned_count


# %%
# Check for missing values in the essential columns 'Long Text' and 'Summary'
missing_long_text = data['Long Text'].isnull().sum()
missing_summary = data['Summary'].isnull().sum()

# Display the count of missing values in each essential column
missing_long_text, missing_summary


# %%
# Select only 'Long Text' and 'Summary' columns and rename the headers to lowercase
data_final = data_cleaned[['Long Text', 'Summary']].rename(columns={'Long Text': 'long text',
'Summary': 'summary'})

# Save this final cleaned dataset to a new CSV file
output_path_final = 'bio_summary_final.csv'
data_final.to_csv(output_path_final, index=False)

output_path_final



const pluckDeep = key => obj => key.split('.').reduce((accum, key) => accum[key], obj)

const compose = (...fns) => res => fns.reduce((accum, next) => next(accum), res)

const unfold = (f, seed) => {
  const go = (f, seed, acc) => {
    const res = f(seed)
    return res ? go(f, res[1], acc.concat([res[0]])) : acc
  }
  return go(f, seed, [])
}
```

Figure 21: Data Preprocessing Script for Summarization

Mechanism for Preparing Summarization Dataset:

The process aims to convert unrefined educational text into organized, structured input for summarization. Below is a detailed breakdown of its operation:

1. Importing the Raw Data: The CSV file is accessed using pandas with the correct encoding, ensuring it is compatible across different platforms and avoiding decoding issues due to special characters.
2. Initial Inspection and Filtering:
    - Unrelated columns (such as index or empty headers) are eliminated.
    - Data statistics are generated to gain insight into the distribution of both text and summary lengths.
3. Text Cleaning and Formatting:
    - A cleaning function eliminates unnecessary punctuation, formatting symbols, and excessive whitespace.
    - Text is converted to lowercase to aid the model in recognizing consistent patterns.
    - Special characters like ¦, •, and other list markers are removed.
4. Removing Duplicates: Duplicate entries are removed to guarantee that the model does not face repeated training instances, which could skew or confuse the learning process.
5. Handling Null Values: The script detects any rows lacking Long Text or Summary entries and addresses them appropriately by either deleting them or filling in minimal values (based on the requirements of the task).
6. Exporting for Model Usage: After cleaning, the dataset is saved to a new CSV file that includes only the essential fields (long text, summary). This file is utilized for training, fine-tuning, or evaluating transformer-based summarization models like Flan-T5.

This preprocessing script guarantees that the summarization model receives input that is well-structured, clean, and semantically rich, thereby enhancing the quality and accuracy of the summaries produced. Figure 22 displays an image of the original

dataset (bio_summary_keywords.csv) before preprocessing and Figure 23 presents the processed dataset after cleaning, showcasing the standardized long text and summary columns prepared for model training.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Long Text	Summary	Keywords														
2	issues pertaining to biology. understar	biological diversity refers to the variety of life on earth, er	Biological diversity, Species, Genes, Ecosystems, Evolution, Human anatomy, Physiology, Sustainable resource management, Overexploitation, Pollution, Biodiversity l														
3	in accordance with different criteria w	living organisms show vast diversity in size, shape, form,	Diversity, Size, Shape, Form, Habitat, Metabolism, Growth, Development, Adaptation, Reproduction, Heredity, Evolution														
4	physical and chemical properties of w	water is an essential molecule for life due to its unique pl	Water molecule, Polarity, Hydrogen bonds, Cohesion, Adhesion, High specific heat, Heat of vaporization, Temperature regulation, Expansion upon freezing, Solvent, Pol														
5	carbohydrates most abundant group c	carbohydrates are the most abundant organic compoun	Carbohydrates, Monosaccharides, Disaccharides, Polysaccharides, Glycosidic bond, Reducing sugars, Non-reducing sugars, Starch, Glycogen, Cellulose, Pectin, Glu														
6	lipids diverse group of hydrophobic m	lipids are a diverse group of hydrophobic molecules that	Lipids, Fats, Phospholipids, Saturated fats, Unsaturated fats, Trans fats, Glycerol, Fatty acids, Ester bond, Phosphate group, Hydrophobic, Hydrophilic, Energy storage,														
7	proteins are made up of amino acids.	proteins are composed of amino acids, with 20 different	Proteins, Amino acids, Peptide bond, Polypeptide, Primary structure, Secondary structure, Alpha helix, Beta pleated sheet, Tertiary structure, Quaternary structure, Der														
8	nucleic acids are polymers exist as po	nucleic acids are large biopolymers that exist as polynuc	Nucleic acids, Nucleotides, DNA, RNA, Pentose sugar, Nitrogenous base, Phosphate group, Purines, Pyrimidines, Phospho-di-ester bond, Double helix, Base pair rule,														
9	advancement of cytology is mostly bac	the advancement of cytology has been largely driven by tl	cytology, light microscope, compound microscope, resolution power, magnification, electron microscope, transmission electron microscope, scanning electron micro														
10	cell theory all organisms are compose	cell theory states that all organisms are composed of on	Cell Theory, Schleiden, Schwann, Virchow, Robert Hooke, Anton Van Leeuwenhoek, Prokaryotic cells, Eukaryotic cells, Binary fission, Mitosis, Meiosis, Plasma membra														
11	plasmamembrane is the outer limit of	the plasma membrane, the outer boundary of the cytopl	Plasma membrane, Fluid mosaic model, Singer and Nicolson, Phospholipids, Proteins, Cholesterol, Amphipathic, Hydrophilic, Hydrophobic, Transmembrane proteins,														
12	there are many subcellular componer	cells contain numerous subcellular components, some	Nucleus, Ribosomes, Endoplasmic Reticulum (ER), Golgi Apparatus, Lysosomes, Mitochondria, Chloroplasts, Cytoskeleton, Cilia, Flagella, Vacuoles.														
13	1. cell wall cell wall is an extracellular	1. cell wall the cell wall is a rigid extracellular structure ir	Cell Wall, Rigid Structure, Plant Cells, Cellulose, Pectin, Hemicellulose, Lignin, Suberin, Protection, Support, Turgidity, Cell Growth, Plasmodesmata, Cell Junctions, Pla														
14	the sequence of events that takes plac	the cell cycle is the sequence of events from one cell divi	Cell Cycle, Mitosis, Interphase, Mitotic Phase, G1 Phase, S Phase, G2 Phase, Prophase, Prometaphase, Metaphase, Anaphase, Telophase, Cytokinesis, Chromatin, Ch														
15	sexually reproducing organisms under	meiosis is a type of nuclear division in sexually reproduci	meiosis, haploid, diploid, crossing over, homologous chromosomes, genetic variation, cancer, galls, cytokinesis, prophase, metaphase, anaphase, telophase														
16	sum of all biochemical reactions of liv	metabolism is the sum of all biochemical reactions in livi	Metabolism, catabolism, anabolism, ATP, photophosphorylation, substrate-level phosphorylation, oxidative phosphorylation, enzymes, activation energy, enzyme inhibi														
17	photosynthesis photosynthesis is a m	photosynthesis is a vital metabolic process where light e	Photosynthesis, chloroplasts, light-dependent reactions, Calvin cycle, ATP, NADPH, carbon fixation, Rubisco, photosystems, photophosphorylation.														
18	as its name suggests, rubisco is capal	rubisco, an enzyme in photosynthesis, catalyzes two rea	Rubisco, carboxylation, oxygenation, photorespiration, C4 plants, CO? concentration, mesophyll cells, bundle sheath cells, Kranz anatomy, oxaloacetate (OAA), photos														
19	cellular respiration is the process by v	cellular respiration is the biochemical process by which	Cellular respiration, aerobic respiration, anaerobic respiration, glycolysis, citric acid cycle, electron transport chain, NADH, ATP, fermentation, respiratory quotient, etl														
20	origin of life on earth condition of eartl	the origin of life on earth began about 4.6 billion years ag	Origin of life, Earth, atmosphere, organic molecules, protocells, photosynthesis, eukaryotes, diversification, geological eons, geological eras, evolution														
21	evolution can be defined as a change i	evolution is defined as a change in the genetic compositi	Evolution, genetic composition, LamarckÂ's Theory, use and disuse, inheritance of acquired characteristics, Darwin-Wallace Theory, natural selection, variation, survi														
22	methods of artificial and natural class	classification arranges organisms into groups based on	Classification, taxonomy, artificial classification, natural classification, phylogeny, binomial nomenclature, kingdoms, Protista, molecular biology, three-domain systen														
23	species is a group of organisms who s	the concept of species refers to a group of organisms tha	species, morphological species concept, ecological species concept, phylogenetic species concept, binomial nomenclature, Carolus Linnaeus, Bacteria, Archaea, Eu														
24	key characteristics of kingdom protist	kingdom protista is a diverse and polyphyletic group of m	Kingdom Protista, unicellular, colonial, multicellular, polyphyletic, photoautotrophs, heterotrophs, mixotrophs, Euglena, Paramecium, Amoeba, Ulva, Gelidium, Sargas														
25	kingdom plantae evolutionary relation	the kingdom plantae evolved from chlorophytes green al	Kingdom Plantae, chlorophytes, vascular plants, non-vascular plants, bryophytes, gametophyte, sporophyte, seedless plants, gymnosperms, angiosperms, monocots,														
26	kingdom fungi characteristic features	kingdom fungi comprises eukaryotic organisms with chit	Kingdom Fungi, eukaryotic, chitin, heterotrophs, hyphae, mycelium, reproduction, spores, Chytridiomycota, Zygomycota, Ascomycota, Basidiomycota, zoospores, zygo														
27	kingdom animalia characteristic feat	the kingdom animalia comprises multicellular, heterotrc	Kingdom Animalia, multicellular, heterotrophic, eukaryotes, tissues, radial symmetry, bilateral symmetry, Cnidaria, Platyhelminthes, Nematoda, Annelida, Mollusca, Ar														
28	the main focus of this unit is on struct	the unit focuses on the structure, growth, and developm	vascular plants, root system, shoot system, meristems, apical meristems, lateral meristems, intercalary meristems, primary growth, secondary growth, dermal tissue,														
29	plant growth growth involves irreversit	plant growth refers to the irreversible increase in dry ma	Plant growth, indeterminate growth, primary structure, roots, stems, secondary growth, heartwood, sapwood, growth rings.														
30	anatomy of typical dicot and monocot	leaves in vascular plants are the primary organs for phot	leaves, dicot, monocot, stomata, epidermis, mesophyll, photosynthesis, gas exchange, turgor pressure, potassium ions, abscisic acid, K+ influx, drought, environmenta														
31	need for transport as land plants evolv	as land plants evolved, they developed vascular tissues s	vascular tissues, xylem, phloem, active transport, passive transport, diffusion, osmosis, facilitated diffusion, imbibition, bulk flow, water potential, solute potential, pres														
32	transport of water and minerals enter	transport of water and minerals from the soil into the xyle	radial transport, xylem, apoplastic route, symplastic route, transmembrane route, endodermis, Casparian strip, transpiration, cohesion, adhesion, mineral absorption,														
33	basic characteristics of phloem trans	phloem transport or phloem translocation is the moven	phloem transport, phloem translocation, phloem sap, sucrose, sugar source, sugar sink, sieve-tube elements, pressure flow hypothesis, transpiration, root pressure, g														

Figure 22: Summarization Dataset before processing

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	long text	summary																					
2	issues pert	biological diversity refers to the variety of life on earth encompassing species genes and ecosystems such as deserts rainforests and coral reefs life began around 35 billion years ago with biodiversity growing through evolution currently scientists estimat																					
3	in accorda	living organisms show vast diversity in size shape form and habitat they can be as small as bacteria 15 m or as large as the giant sequoia over 100 meters shapes vary from cylindrical like earthworms to streamlined like birds organisms can be unicellular																					
4	physical ar	water is an essential molecule for life due to its unique physical and chemical properties it is a polar angular molecule with hydrogen bonds forming between water molecules these hydrogen bonds give water several vital properties cohesive behavior col																					
5	carbohydr	carbohydrates are the most abundant organic compounds on earth composed of carbon hydrogen and oxygen c h o in a 21 ratio of h they are classified into three major groups monosaccharides simple sugars with 37 carbon atoms such as glucose fructo																					
6	lipids diver	lipids are a diverse group of hydrophobic molecules that are important for biological functions but are not considered polymers they consist of carbon hydrogen and oxygen with a higher ratio of hydrogen to oxygen compared to carbohydrates lipids are ca																					
7	proteins ar	proteins are composed of amino acids with 20 different types involved in their structure each amino acid contains an amino group carboxyl group hydrogen atom and a variable r group amino acids are amphoteric meaning they can behave as both acids a																					
8	nucleic ac	nucleic acids are large biopolymers that exist as polynucleotides made up of monomers called nucleotides each nucleotide consists of three components a pentose sugar a nitrogenous base and a phosphate group there are two main types of nucleic ac																					
9	advancem	the advancement of cytology has been largely driven by the development of microscopes the light microscope uses visible light to pass through specimens magnified through glass lenses a compound light microscope commonly used in schools and med																					
10	cell theory	cell theory states that all organisms are composed of one or more cells the cell is the basic structural and functional unit of life and all cells arise from preexisting cells through division the theory was developed by schleiden schwann and virchow robert h																					
11	plasmame	the plasma membrane the outer boundary of the cytoplasm follows the fluid mosaic model proposed by singer and nicolson in 1972 it is composed of phospholipids proteins and sometimes cholesterol the membranes phospholipid bilayer is amphipathic																					
12	there are n	cells contain numerous subcellular components some of which are membranebound organelles that perform specialized functions nucleus the control center of the cell enclosed by a nuclear envelope with pores contains chromatin dnaprotein comple																					
13	1 cell wall i	1 cell wall the cell wall is a rigid extracellular structure in plant cells made of cellulose pectin hemicellulose lignin and suberin it provides protection support and maintains cell shape it consists of a primary wall flexible and permeable and a secondary wa																					
14	the sequer	the cell cycle is the sequence of events from one cell division to the next producing two genetically identical daughter cells through mitosis the eukaryotic cell cycle consists of two main phases interphase 90 of the cycle and the mitotic phase m phase int																					
15	sexually re	meiosis is a type of nuclear division in sexually reproducing organisms resulting in four haploid genetically nonidentical daughter nuclei from a diploid mother nucleus it consists of two stages meiosis i reduction division and meiosis ii similar to mitosis ke																					
16	sum of all t	metabolism is the sum of all biochemical reactions in living organisms comprising catabolic and anabolic processes catabolism breaks down complex molecules into simpler ones releasing energy exergonic anabolism synthesizes complex molecules fr																					
17	photosynth	photosynthesis is a vital metabolic process where light energy is converted into chemical energy stored in carbohydrates fats oils and proteins it occurs in chloroplasts of eukaryotic cells and involves two main stages lightdependent reactions and the cal																					
18	as its nam	rubisco an enzyme in photosynthesis catalyzes two reactions carboxylation using co and oxygenation using o the oxygenase reaction produces 3pga and 2phosphoglycolate the latter being toxic and processed via photorespiration leading to a net loss of c																					
19	cellular res	cellular respiration is the biochemical process by which cells convert organic molecules primarily glucose into atp adenosine triphosphate energy it occurs in two main types aerobic respiration occurs in the presence of oxygen involves three main stage																					
20	origin of lif	the origin of life on earth began about 46 billion years ago with the planet forming amidst rock and ice bombardment the first atmosphere was thick with water vapor and volcanic gases leading to a reducing environment with little oxygen key processes su																					
21	evolution c	evolution is defined as a change in the genetic composition of a population over generations often taking millions of years key theories of evolution include lamarcks theory proposed in 1809 it consists of two principles use and disuse body parts used exte																					
22	methods o	classification arranges organisms into groups based on common characteristics and taxonomy is the scientific study of this process there are two main classification methods artificial classification groups organisms based on a few selected traits ignor																					
23	species is	the concept of species refers to a group of organisms that share similar characteristics and can interbreed to produce viable offspring different definitions of species include morphological species concept distinguishes species based on physical traits																					
24	key charac	kingdom protista is a diverse and polyphyletic group of mostly unicellular organisms although some are colonial or multicellular they thrive in various environments including freshwater marine and damp soil and can be symbiotic nutritionally protists can																					
25	kingdom pl	the kingdom plantae evolved from chlorophytes green algae and consists of two major groups vascular and nonvascular plants nonvascular plants or bryophytes eg mosses liverworts hornworts lack vascular tissues and are characterized by a dominant l																					
26	kingdom fu	kingdom fungi comprises eukaryotic organisms with chitinous cell walls heterotrophic lifestyles and diverse forms including unicellular yeasts and multicellular filamentous fungi known as hyphae key characteristics include hyphae and mycelium multic																					
27	kingdom an	the kingdom animalia comprises multicellular heterotrophic eukaryotes that digest food using enzymes and are organized into tissues most animals reproduce sexually displaying either radial or bilateral symmetry key phyla include cnidaria primarily ma																					
28	the main fc	the unit focuses on the structure growth and development of vascular plants which consist of root and shoot systems plant growth occurs at tips called meristems which are undifferentiated tissues that divide and differentiate into various tissues key me																					
29	plant grow	plant growth refers to the irreversible increase in dry mass and cell number occurring throughout their life indeterminate growth key structures in plant growth include																					
30	anatomy o	leaves in vascular plants are the primary organs for photosynthesis and gas exchange through stomata which are pores regulated by guard cells epidermis is a single cell layer covering the leaf while mesophyll made up of parenchyma cells is responsible																					
31	need for tr	as land plants evolved they developed vascular tissues xylem and phloem for efficient longdistance transport of water minerals and nutrients key transport mechanisms include active transport requires atp to move substances against their concentratio																					
32	transport c	transport of water and minerals from the soil into the xylem of roots known as radial transport occurs through three routes apoplastic symplastic and transmembrane apoplastic route involves movement through cell walls and extracellular spaces blocke																					
33	basic char	phloem transport or phloem translocation is the movement of photosynthesis products primarily sucrose through specialized sieve tube elements in angiosperms key points include phloem sap containing about 20 sucrose alongwith amino acids hormone																					

Figure 23: Summarization Dataset after processing

2.1.4.2 Dataset for Information Retrieval (RAG)

Here are the main features of this dataset:

1. Structured Format: The dataset contains 156 entries and includes the following fields:
    - Document ID: A distinct numeric identifier assigned to each entry.
    - Topic: The general subject area (e.g., "Introduction to Biology").
    - Sub-topic: A more detailed category within the primary topic.
    - Text Content: The core educational text.
    - Source: The textbook or resource from which the material was taken (e.g., "Biology, Grade 12, Resource Book").

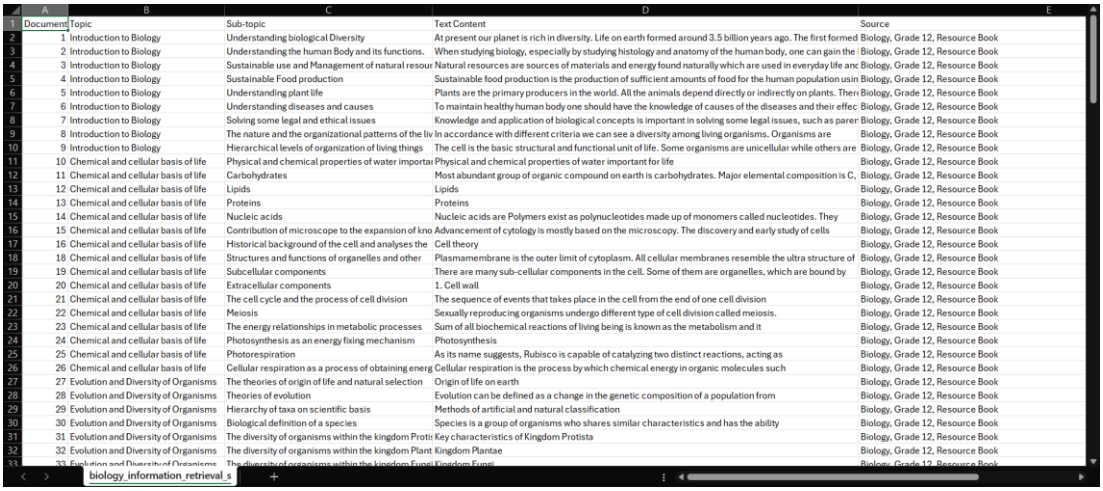
2. Raw but Usable: Although this dataset hasn't been extensively cleaned or transformed, it remains well-organized and easy to read. Each entry offers concise and relevant biology content pertinent for indexing.

3. Tailored for Retrieval Tasks: This dataset is specifically designed for Retrieval-Augmented Generation (RAG) workflows. The Text Content can be embedded and indexed using semantic vector representations (e.g., FAISS), while the Topic and Sub-topic fields serve as metadata for contextual filtering and scoring of relevance.

4. Source-Attributed Data: The presence of the Source column aids in tracking the content's origin, enhancing transparency, enabling filtering by curriculum, or supporting dataset enhancement from similar sources.

5. Compatibility with RAG Model: The data can be directly utilized in a RAG-based framework, where a user's query is semantically matched with the Text Content entries. After retrieving the relevant content, it can then be processed by a transformer model for summarization or question-answer generation.

Figure 24 illustrates a segment of the original dataset, highlighting its structure and effectiveness for information retrieval in educational NLP applications.



Document	Topic	Sub-topic	Text Content	Source
1	Introduction to Biology	Understanding biological Diversity	At present our planet is rich in diversity. Life on earth formed around 3.5 billion years ago. The first formed	Biology, Grade 12, Resource Book
2	Introduction to Biology	Understanding the human Body and its functions.	When studying biology, especially by studying histology and anatomy of the human body, one can gain the	Biology, Grade 12, Resource Book
3	Introduction to Biology	Sustainable use and Management of natural resour	Natural resources are sources of materials and energy found naturally which are used in everyday life anc	Biology, Grade 12, Resource Book
4	Introduction to Biology	Sustainable Food production	Sustainable food production is the production of sufficient amounts of food for the human population usin	Biology, Grade 12, Resource Book
5	Introduction to Biology	Understanding plant life	Plants are the primary producers in the world. All the animals depend directly or indirectly on plants. Ther	Biology, Grade 12, Resource Book
6	Introduction to Biology	Understanding diseases and causes	To maintain healthy human body one should have the knowledge of causes of the diseases and their effec	Biology, Grade 12, Resource Book
7	Introduction to Biology	Solving some legal and ethical issues	Knowledge and application of biological concepts is important in solving some legal issues, such as parer	Biology, Grade 12, Resource Book
8	Introduction to Biology	The nature and the organizational patterns of the liv	In accordance with different criteria we can see a diversity among living organisms. Organisms are	Biology, Grade 12, Resource Book
9	Introduction to Biology	Hierarchical levels of organization of living things	The cell is the basic structural and functional unit of life. Some organisms are unicellular while others are	Biology, Grade 12, Resource Book
10	Chemical and cellular basis of life	Physical and chemical properties of water importar	Physical and chemical properties of water important for life	Biology, Grade 12, Resource Book
11	Chemical and cellular basis of life	Carbohydrates	Most abundant group of organic compound on earth is carbohydrates. Major elemental composition is C,	Biology, Grade 12, Resource Book
12	Chemical and cellular basis of life	Lipids	Lipids	Biology, Grade 12, Resource Book
13	Chemical and cellular basis of life	Proteins	Proteins	Biology, Grade 12, Resource Book
14	Chemical and cellular basis of life	Nucleic acids	Nucleic acids are Polymers exist as polynucleotides made up of monomers called nucleotides. They	Biology, Grade 12, Resource Book
15	Chemical and cellular basis of life	Contribution of microscope to the expansion of kno	Advancement of cytology is mostly based on the microscopy. The discovery and early study of cells	Biology, Grade 12, Resource Book
16	Chemical and cellular basis of life	Historical background of the cell and analyses the	Cell theory	Biology, Grade 12, Resource Book
17	Chemical and cellular basis of life	Structures and functions of organelles and other	Plasmamembrane is the outer limit of cytoplasm. All cellular membranes resemble the ultra structure of	Biology, Grade 12, Resource Book
18	Chemical and cellular basis of life	Subcellular components	There are many sub-cellular components in the cell. Some of them are organelles, which are bound by	Biology, Grade 12, Resource Book
19	Chemical and cellular basis of life	Extracellular components	1. Cell wall	Biology, Grade 12, Resource Book
20	Chemical and cellular basis of life	The cell cycle and the process of cell division	The sequence of events that takes place in the cell from the end of one cell division	Biology, Grade 12, Resource Book
21	Chemical and cellular basis of life	Meiosis	Sexually reproducing organisms undergo different type of cell division called meiosis.	Biology, Grade 12, Resource Book
22	Chemical and cellular basis of life	The energy relationships in metabolic processes	Sum of all biochemical reactions of living being is known as the metabolism and it	Biology, Grade 12, Resource Book
23	Chemical and cellular basis of life	Photosynthesis as an energy fixing mechanism	Photosynthesis	Biology, Grade 12, Resource Book
24	Chemical and cellular basis of life	Photorespiration	As its name suggests, Rubisco is capable of catalyzing two distinct reactions, acting as	Biology, Grade 12, Resource Book
25	Chemical and cellular basis of life	Cellular respiration as a process of obtaining energ	Cellular respiration is the process by which chemical energy in organic molecules such	Biology, Grade 12, Resource Book
26	Evolution and Diversity of Organisms	The theories of origin of life and natural selection	Origin of life on earth	Biology, Grade 12, Resource Book
27	Evolution and Diversity of Organisms	Theories of evolution	Evolution can be defined as a change in the genetic composition of a population from	Biology, Grade 12, Resource Book
28	Evolution and Diversity of Organisms	Hierarchy of taxa on scientific basis	Methods of artificial and natural classification	Biology, Grade 12, Resource Book
29	Evolution and Diversity of Organisms	Biological definition of a species	Species is a group of organisms who shares similar characteristics and has the ability	Biology, Grade 12, Resource Book
30	Evolution and Diversity of Organisms	The diversity of organisms within the kingdom Protis	Key characteristics of Kingdom Protista	Biology, Grade 12, Resource Book
31	Evolution and Diversity of Organisms	The diversity of organisms within the kingdom Plant	Kingdom Plantae	Biology, Grade 12, Resource Book
32	Evolution and Diversity of Organisms	The diversity of organisms within the kingdom Fungi	Kingdom Fungi	Biology, Grade 12, Resource Book

biology_information_retrieval_s

Figure 24: Information Retrieval Dataset

2.1.4.3 Model Fine-Tuning for Summarization Using Flan-T5-Base

The objective of this stage is to customize the pre-trained Flan-T5-Base model for producing effective summaries from educational material related to biology. Fine-tuning of the model was conducted using a carefully crafted dataset (bio_summary_final.csv) that features extensive academic texts along with their brief summaries.

This procedure was carried out exclusively in Google Colab, utilizing its GPU acceleration, engaging coding interface, and seamless access to files via Google Drive.

Step 1: Configuring the Environment in Google Colab

The fine-tuning process starts within Google Colab, a cloud-based Jupyter notebook platform. It offers:

- Complimentary access to GPUs (such as Tesla T4, P100).
- Seamless integration with Google Drive for storing and retrieving data.
- Built-in support for Python, making it perfect for rapid model training.

To set up the environment:

- The transformers and datasets libraries were installed to leverage Hugging Face's robust model and data functionalities.
- Google Drive was connected to access the training dataset and to save the outputs of the trained model.

Step 2: Loading and Structuring the Dataset

The dataset utilized for fine-tuning is bio_summary_final.csv, a refined and standardized CSV file generated from the preprocessing stage. It consists of:

- longtext: A column that holds in-depth biological information.
- summary: A column that features the associated human-generated summaries.

The CSV file was imported into a pandas dataframe and subsequently transformed into a Hugging Face Dataset object. This transformation is essential for ensuring compatibility with the Trainer API employed during the training process.

Step 3: Data Preparation and Tokenization

Prior to inputting the data into the model:

- Each entry in the longtext column was prefixed with the phrase "summarize: " to direct the model towards the summarization task. This aligns with the instruction-based learning approach utilized by T5 models.
- Tokenization was performed on both the input and output texts:
  - Input sequences were restricted to a maximum length of 512 tokens.
  - Summaries were either truncated or padded to a maximum of 150 tokens.
- Both input and output tokens were transformed into numerical formats using the Flan-T5 tokenizer.

This phase ensured consistency in input dimensions and helped prevent runtime errors during the training process.

Step 4: Loading the Pre-trained Model

The foundational model utilized was google/flan-t5-base, a compact yet effective variant of Flan-T5 designed for sequence-to-sequence tasks such as summarization.

Flan-T5 is a version of the T5 (Text-To-Text Transfer Transformer) model that has been fine-tuned for instruction-following, making it well-suited for this task because of:

- Its capability to produce natural, coherent summaries.
- Pretraining on a diverse array of language-related tasks.

Step 5: Establishing Training Parameters

A collection of training settings was established to dictate the manner in which the model would be trained:

- Output Directory: The destination for storing model checkpoints and the final version.
- Batch Size: The quantity of samples processed simultaneously (configured to 4 per device).
- Number of Epochs: The overall number of training cycles (3 epochs were implemented).
- Logging Steps: The interval for recording training progress.
- Checkpointing: A strategy for saving to prevent data loss during extended training periods.

These settings were provided to Hugging Face's TrainingArguments class.

Step 6: Training the Model with Hugging Face Trainer API

Having prepared the model, tokenized dataset, and training parameters, the Trainer API from Hugging Face was set up.

The .train() function was executed to start the fine-tuning procedure. During the training phase:

- The model starts to generate target summaries corresponding to the input text.
- Loss is calculated by comparing the predicted outputs with the actual summaries.
- Incremental weight adjustments enable the model to focus on the structure and meaning of language specific to biology.

The training status was tracked in real-time on Colab, with loss figures displayed at each logging interval.

Step 7: Preserving the Fine-Tuned Model

Once the training is finished:

- The model and tokenizer were stored in a Google Drive folder for long-term access.
- This preserved version can be retrieved later for inference, additional fine-tuning, or integration into an educational application.

The fine-tuning procedure effectively adjusted the Flan-T5 model for the field of educational summarization. The resulting model can take comprehensive biology texts and produce clear, pertinent, and succinct summaries designed for an academic setting.

Figure 26 illustrates the configuration of the Google Colab notebook utilized for this process, featuring the training pipeline and tokenization approach.

```
# %%
!pip install transformers datasets

# %%
from google.colab import drive
drive.mount('/content/drive')

# %%
import pandas as pd
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, Trainer, TrainingArguments

# %%
# Load the dataset
data_path = '/content/drive/MyDrive/bio_summary_final.csv'
df = pd.read_csv(data_path)

# %%
# Convert the DataFrame to a Hugging Face Dataset format
dataset = Dataset.from_pandas(df)

# %%
# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-base")

# %%
# Tokenization function with padding
def preprocess_data(examples):
    inputs = ["summarize: " + doc for doc in examples["longtext"]]
    model_inputs = tokenizer(inputs, max_length=512, padding="max_length", truncation=True)  # Adding
padding

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(examples["summary"], max_length=150, padding="max_length", truncation=True)
 # Adding padding

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

# %%
print(df.columns)

# %%
# Apply preprocessing to the dataset
tokenized_dataset = dataset.map(preprocess_data, batched=True)

# %%
# Load the model
model = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-base")

# %%
# Set up training arguments
training_args = TrainingArguments(
    output_dir="./flan_t5_finetuned",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    weight_decay=0.01,
    save_total_limit=2,
    push_to_hub=False
)

# %%
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    eval_dataset=tokenized_dataset,
)

# %%
trainer.train()

# %%
model.save_pretrained('/content/drive/MyDrive/flan_t5_finetuned_model')
tokenizer.save_pretrained('/content/drive/MyDrive/flan_t5_finetuned_model')

# %%
# Define the path
model_save_path = '/content/drive/MyDrive/flan_t5_finetuned_model_2nd'

# %%
# Save the model
trainer.model.save_pretrained(model_save_path)

# %%
# Save the tokenizer
tokenizer.save_pretrained(model_save_path)
```

Figure 26: Finetuning the LLM for summarization

2.1.4.4 Summarization System (summarization.py)

This integrated application provides intelligent summarization, structured note generation, audio support, and multilingual translation for educational content. It leverages pre-trained language models and a Retrieval-Augmented Generation (RAG) pipeline to enhance learning and engagement.

1. Core Components Initialization

The summarizzation.py file begins by setting up the essential components:

- Flan-T5 Fine-Tuned Model designed for generating notes and performing abstractive summarization.
- FAISS Index utilized semantic search and quick retrieval of pertinent biology information.
- Sentence Transformer employed for embedding both queries and content.
- These components are encapsulated within a class named RAGModel, ensuring their reusability across various endpoints.

2. API Endpoints and Functionality

The FastAPI application provides a variety of POST and GET endpoints:

a) /process-document/

- Manages document uploads (PDF, DOCX, PPTX, TXT).
- Extracts text using specialized handlers and formats and cleans it.
- Creates a summary with a specified word count and returns both:
  - A PDF summary file.
  - An MP3 audio version generated with gTTS.

b) /process-query/

- Accepts a user query (e.g., “Explain photosynthesis”).

- Conducts semantic searches across pre-loaded educational content utilizing FAISS.
- Obtains the most relevant paragraphs, summarizes them, and returns:
  - A PDF file.
  - An audio file for accessibility purposes.

c) /summarize-text/

- Summarizes raw text input directly into a brief explanation.
- Converts results into a downloadable summary (PDF) and speech (MP3).

d) /generate-notes/

- Produces organized notes based on a given topic, formatting and chunking the output with the Flan-T5 model.
- Allows multilingual output with translation into Tamil (ta) or Sinhala (si) through deep_translator.
- Generates a PDF for English notes, or provides plain text output for translated versions.

3. Text Extraction and Preprocessing

- The system enables the extraction of rich content from:
- PDFs using fitz or pytesseract (for Optical Character Recognition).
- Word and PowerPoint documents (utilizing python-docx and python-pptx).
- Tables are interpreted and transformed into coherent paragraphs.
- A multi-step cleaning process:
  - Eliminates headings, bullet points, and unnecessary noise.
  - Enhances spelling and grammar.
  - Rectifies word splits and removes repetitive phrases.

4. Retrieval-Augmented Generation (RAG)

- FAISS indexing allows for effective semantic searching.

- The retrieval base is constructed using predefined datasets (bio_summary_keywords.csv, biology_information_retrieval_sample.csv).
- When a query is made:
  - Embeddings are created.
  - Relevant information is retrieved.
  - The Flan-T5 model condenses the content.

5. Detection of Inappropriate Content

Every question or submitted file undergoes a review for:

- Offensive language or unsuitable terms.
- Hostile sentiments.
- Non-English or nonsensical expressions.
- If any are found, a friendly message is displayed, encouraging rewording.

6. Voice Generation

- Summarized content is transformed into speech with the help of gTTS.
- The audio is kept in memory and can be streamed as MP3 for downloading or listening.

7.Translation Support

The system supports multilingual translation for structured notes via the /generate-notes/ endpoint.

- Using the deep_translator package:
  - Generated notes can be translated from English into Tamil (ta) and Sinhala (si).
  - The translation happens automatically after summarization.
- For translated outputs:

- Notes are returned as plain text in the requested language.
- Ideal for regional learners and inclusive education platforms.

8. Output and File Handling

All results are stored in memory for session-based access:

- file_handler.py manages uploads and generates PDFs.
- summarization_functions.py tracks tasks, handles cancellations, and manages temporary storage.
- Downloadable files:
  - /download-summary-text/{task_id}
  - /download-summary-audio/{task_id}
  - /download-notes/{file_name}

All the functionalities described above summarization of documents, processing of topic queries, audio synthesis, note creation, and multilingual translation were realized using two contrasting architectural paradigms for the purpose of measuring performance, modularity, and deployability. The first realization was based on monolithic architecture, in which all the components and processing were part of one application. The second realization used microservices architecture, dividing the system into separate deployable services for summarization, translation, file handling, and voice synthesis.

The two-architecture strategy enabled the examination of the simplicity-versus-scalability trade-off in its entirety. Figures 27 and 28 present the organization of folder structure and modular partitioning of the implementation, depicting the placement of components in both versions of the two architectures.

Following the comparison, monolithic deployment was adopted for integration with the frontend system as it offered faster response times, easier debugging, and easier deployment especially well-suited for a student-facing learning platform like BioMentor. The detailed comparative study of CPU/memory utilization, deployment

time, and fault tolerance, is reported in the Results & Discussion section, providing a comprehensive view of the trade-offs and design decisions taken.

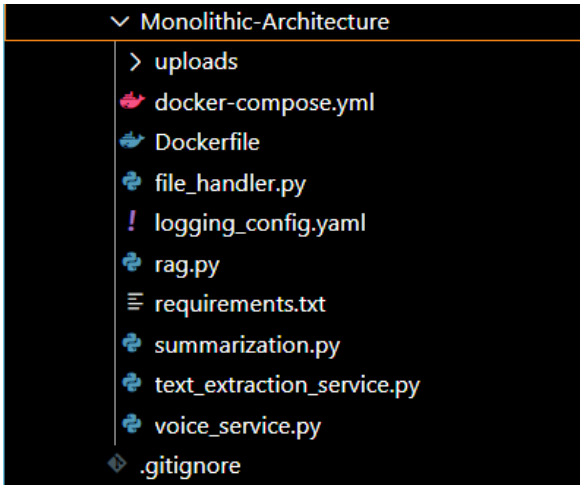


Figure 27: Folder Structure of Monolithic Architecture

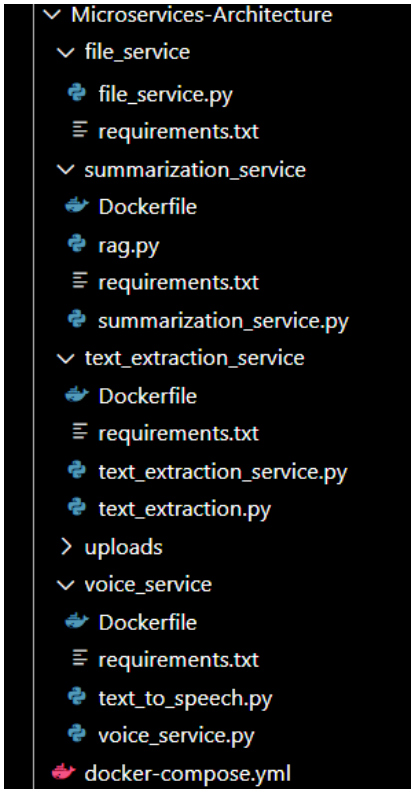


Figure 28: Folder Structure of Microservices Architecture

Overall Mechanism

The system aims to enhance the quality of digital education by:

- Automatically transforming intricate biology material into straightforward summaries.
- Fetching pertinent content in response to student inquiries.
- Delivering organized, exam-prepared notes.
- Facilitating the use of regional languages to improve accessibility.
- Providing audio output for auditory learners and users with visual impairments.
- The complete process is smooth and exceptionally optimized for quick, real-time engagement in educational settings.

Figures 29–34 illustrate various parts of the application code from API, text extraction and semantic retrieval to summarization logic, voice output, and structured note generation.

```python
from fastapi import FastAPI, UploadFile, Form, Request, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
from summarization_functions import process_document_function
from summarization_functions import process_query_function
from summarization_functions import summarize_text_function
from summarization_functions import generate_notes_function
from summarization_functions import get_summary_file
from summarization_functions import get_audio_file
from summarization_functions import get_pdf_file
from rag import RAGModel
import yaml
import logging


# Load logging configuration
with open("logging_config.yaml", "r") as file:
    config = yaml.safe_load(file)
    logging.config.dictConfig(config)

logger = logging.getLogger("myapp")


app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Initialize RAG Model
try:
    logger.info("Initializing RAG Model...")
    rag_model = RAGModel(
        model_path='D:/Downloads/RP/Summarization/flan_t5_finetuned_model',
        embedding_model_name='all-MiniLM-L6-v2',
        dataset_paths=[
            '../../../Model-Training/Summarization/bio_summary_keywords.csv',
            '../../../Model-Training/Summarization/biology_information_retrieval_sample.csv'
        ]
    )
    logger.info("RAG Model initialized successfully.")
except Exception as e:
    logger.error(f"Failed to initialize RAG Model: {e}", exc_info=True)
    raise


@app.post("/process-document/")
async def process_document(request: Request, file: UploadFile, word_count: int = Form(...)):
    """
    Endpoint to process a document: extract, summarize, and convert to speech.
    """
    summary, task_id = await process_document_function(request, file, word_count, rag_model)

    if summary is None:
        raise HTTPException(status_code=499, detail="Request was canceled.")

    return {
        "status": "success",
        "message": "Document processed successfully.",
        "summary": summary,
        "summary_file": f"/download-summary-text/{task_id}",
        "voice_file": f"/download-summary-audio/{task_id}"
    }


@app.post("/process-query/")
async def process_query(request: Request, query: str = Form(...), word_count: int =
Form(...)):
    Endpoint to retrieve and summarize content based on a query.
    """
    summary, task_id = await process_query_function(request, query, word_count, rag_model)

    if summary is None:
        raise HTTPException(status_code=499, detail="Request was canceled.")

    return {
        "status": "success",
        "message": "Query processed successfully.",
        "summary": summary,
        "summary_file": f"/download-summary-text/{task_id}",
        "voice_file": f"/download-summary-audio/{task_id}"
    }


@app.post("/summarize-text/")
async def summarize_text(request: Request, text: str = Form(...), word_count: int =
Form(...)):
    Endpoint to summarize a given text.
    """
    summary, task_id = await summarize_text_function(request, text, word_count, rag_model)

    if summary is None:
        raise HTTPException(status_code=499, detail="Request was canceled.")

    return {
        "status": "success",
        "message": "Summary generated successfully.",
        "summary": summary,
        "summary_file": f"/download-summary-text/{task_id}",
        "voice_file": f"/download-summary-audio/{task_id}"
    }


@app.post("/generate-notes/")
async def generate_notes(
    request: Request,
    topic: str = Form(...),
    lang: str = Form(None)  # Optional: "ta" (Tamil) or "si" (Sinhala)
):
    """
    Endpoint to generate structured notes for a given topic.
    """
    response = await generate_notes_function(request, topic, lang, rag_model)

    if response is None:
        raise HTTPException(status_code=499, detail="Request was canceled.")

    return JSONResponse(content=response)


@app.get("/download-summary-text/{task_id}")
async def download_summary_text(task_id: str):
    """
    Endpoint to download a summary text file based on task_id.
    Works for all three APIs: process-document, process-query, and summarize-text.
    """
    return await get_summary_file(task_id)


@app.get("/download-summary-audio/{task_id}")
async def download_summary_audio(task_id: str):
    """
    Endpoint to download a summary audio file based on task_id.
    Works for all three APIs: process-document, process-query, and summarize-text.
    """
    return await get_audio_file(task_id)


@app.get("/download-notes/{file_name}")
async def download_notes(file_name: str):
    """
    Endpoint to download the generated notes PDF file.
    """
    return await get_pdf_file(file_name)

if __name__ == "__main__":
    import uvicorn
    logger.info("Starting FastAPI server...")
    uvicorn.run(app, host="0.0.0.0", port=8082)
```

Figure 29: summarization.py

```python
import io
import asyncio
import hashlib
import logging
from fastapi import Request, HTTPException, BackgroundTasks
from text_extraction_service import extract_content, clean_text, format_as_paragraph
from voice_service import text_to_speech
from file_handler import save_uploaded_file, generate_pdf
import io
import hashlib
import asyncio
import yaml
from fastapi.responses import StreamingResponse


with open("logging_config.yaml", "r") as file:
    config = yaml.safe_load(file)
    logging.config.dictConfig(config)

logger = logging.getLogger("myapp")

file_store = {}  # Temporary storage for files
ongoing_tasks = {}  # Track running tasks


async def process_document_function(request: Request, file, word_count, rag_model):
    """
    Handles document processing: extraction, summarization, and text-to-speech conversion.
    Ensures request cancellations do not keep processing.
    """
    task_id = hashlib.md5(file.filename.encode()).hexdigest()

    # Cancel existing task if already running
    if task_id in ongoing_tasks:
        logger.warning(f"Cancelling previous request for {file.filename}")
        ongoing_tasks[task_id].cancel()
        del ongoing_tasks[task_id]

    async def process():
        try:
            logger.info(f"Processing document: {file.filename}")

            # Step 1: Save uploaded file
            file_path = save_uploaded_file(file)
            logger.info(f"File saved to {file_path}. Extracting content...")

            raw_text = extract_content(file_path)

            if not raw_text.strip():
                logger.warning(f"No content extracted from {file.filename}.")
                raise HTTPException(
                    status_code=400, detail="No content extracted.")

            # Check if the client disconnected before proceeding
            if await request.is_disconnected():
                logger.warning(
                    f"Request was canceled. Stopping processing for {file.filename}.")
                return None

            # Step 2: Clean and format text
            cleaned_text = clean_text(raw_text)
            formatted_text = format_as_paragraph(cleaned_text)

            # Check again before summarization
            if await request.is_disconnected():
                logger.warning(
                    f"Request was canceled. Stopping processing for {file.filename}.")
                return None

            # Generate summary
            logger.info(f"Generating summary for {file.filename}...")

            # Step 3: Generate summary
            summary = rag_model.generate_summary_for_long_text(
                formatted_text, max_words=word_count)

            if not summary:
                logger.warning(
                    f"Failed to generate summary for {file.filename}.")
                raise HTTPException(
                    status_code=500, detail="Summary generation failed.")

            summary_file_path = f"summary_{task_id}.pdf"
            audio_file_path = f"summary_{task_id}.mp3"

            # Generate PDF
            pdf_data = generate_pdf(summary, topic="Summary")
            file_store[summary_file_path] = pdf_data

            logger.info(
                f"PDF summary saved as {summary_file_path}. Generating audio...")

            # Check before generating audio
            if await request.is_disconnected():
                logger.warning(
                    f"Request was canceled. Stopping processing for {file.filename}.")
                return None

            # Step 4: Convert summary to speech
            audio_file = io.BytesIO()
            text_to_speech(summary, audio_file)
            audio_file.seek(0)
            file_store[audio_file_path] = audio_file.read()

            logger.info(
                f"Audio file saved as {audio_file_path}. Processing complete.")

            # Cleanup task tracking
            del ongoing_tasks[task_id]

            return summary, task_id

        except asyncio.CancelledError:
            logger.warning(f"Processing for {file.filename} was canceled.")
            del ongoing_tasks[task_id]
            return None, None

        except HTTPException as http_err:
            del ongoing_tasks[task_id]
            raise http_err  # Return FastAPI HTTPException directly

        except Exception as e:
            logger.error(
                f"Error processing document {file.filename}: {e}", exc_info=True)
            del ongoing_tasks[task_id]
            raise HTTPException(
                status_code=500, detail="Failed to process document.")

    # Store and run the task in the background
    task = asyncio.create_task(process())
    ongoing_tasks[task_id] = task
    return await task
```

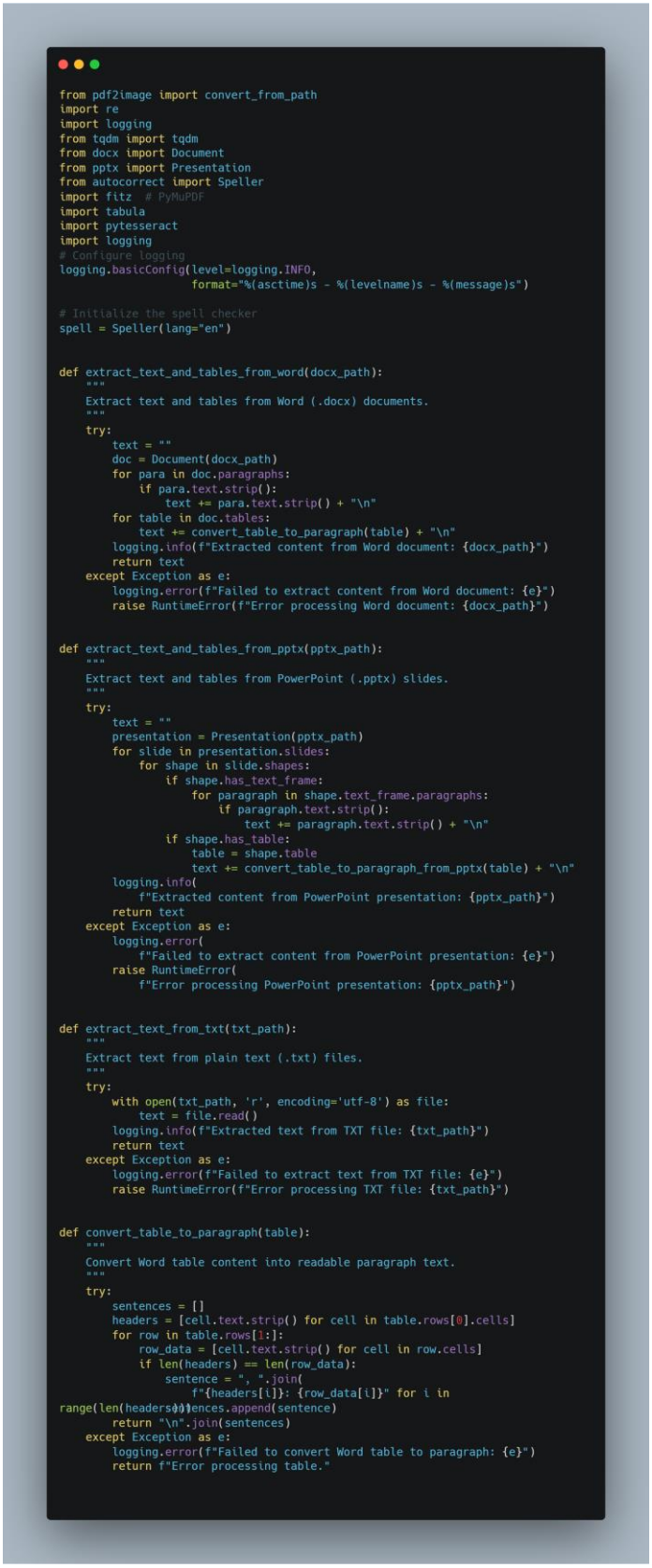
Figure 30: summarization_function.py

```python
from utils.dangerous_keywords import DANGEROUS_KEYWORDS
import logging
import pandas as pd
from sentence_transformers import SentenceTransformer
import faiss
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from autocorrect import Speller
import language_tool_python
import re
from better_profanity import profanity
import joblib
from textblob import TextBlob
from nltk.corpus import words, wordnet
import nltk
import asyncio
from deep_translator import GoogleTranslator
import textwrap


nltk.download("words")
nltk.download("wordnet")
ENGLISH_WORDS = set(words.words())


# Configure logging
logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s"
)


class RAGModel:
    def __init__(self, model_path, embedding_model_name, dataset_paths,
max_tokens=512):
        Initialize the Retrieval-Augmented Generation (RAG) Model.
        """
        try:
            # Load model and tokenizer
            self.tokenizer = AutoTokenizer.from_pretrained(model_path)
            self.model = AutoModelForSeq2SeqLM.from_pretrained(model_path)
            self.embedder = SentenceTransformer(embedding_model_name)
            self.spell_checker = Speller(lang="en")
            self.grammar_tool = language_tool_python.LanguageTool("en-US")
            self.max_tokens = max_tokens
            logging.info("RAG Model components loaded successfully.")

            # Load datasets and initialize FAISS index
            self.long_texts, self.notes_content = self._load_data(
                dataset_paths)
            self.faiss_index = self._initialize_faiss(
                self.long_texts, self.notes_content
            )
            logging.info("FAISS index initialized successfully.")

        except Exception as e:
            logging.error(f"Error initializing RAG Model: {e}")
            raise

    def _load_data(self, dataset_paths):
        """
        Load and preprocess datasets from given file paths.
        """
        long_texts = []
        notes_content = []
        try:
            for dataset_path in dataset_paths:
                try:
                    df = pd.read_csv(dataset_path, encoding="ISO-8859-1")
                    if "Long Text" in df.columns:
                        long_texts.extend(df["Long Text"].tolist())
                    if "Text Content" in df.columns:
                        notes_content.extend(df["Text Content"].tolist())
                except FileNotFoundError:
                    logging.error(f"Dataset not found: {dataset_path}")
                    continue
                except pd.errors.EmptyDataError:
                    logging.error(f"Empty dataset at path: {dataset_path}")
                    continue

            if not long_texts and not notes_content:
                raise ValueError("No valid data found in datasets.")

            logging.info("Datasets loaded successfully.")
            return long_texts, notes_content

        except Exception as e:
            logging.error(f"Error loading datasets: {e}")
            raise
```

Figure 31: rag.py

```python
import logging
import os
from fastapi import UploadFile
import os
from io import BytesIO
from fpdf import FPDF


# Configure logging
logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s"
)

import os

# Change this to a writable location
UPLOAD_DIR = "/tmp/uploads"


def save_uploaded_file(file: UploadFile):
    """
    Save an uploaded file to the uploads directory.
    """
    try:
        if not os.path.exists(UPLOAD_DIR):
            os.makedirs(UPLOAD_DIR)
            logging.info(f"Created upload directory: {UPLOAD_DIR}")

        file_path = os.path.join(UPLOAD_DIR, file.filename)
        with open(file_path, "wb") as f:
            f.write(file.file.read())
        logging.info(f"File saved successfully: {file_path}")
        return file_path
    except Exception as e:
        logging.error(f"Failed to save uploaded file: {e}")
        raise RuntimeError("Error in saving uploaded file.")


def generate_pdf(structured_notes: str, topic: str) -> bytes:
    """
    Generate a PDF in English only using FPDF, returning the PDF data as bytes.

    :param structured_notes: The textual content for the PDF.
    :param topic: Title/topic for the PDF.
    :return: The PDF file data in bytes.
    """
    pdf = FPDF()
    pdf.add_page()

    # Set a title font (bold, size 16)
    pdf.set_font("Arial", "B", 16)
    pdf.cell(0, 10, txt=f"{topic}", ln=True, align="C")


    # Change to a normal font for content
    pdf.set_font("Arial", size=12)

    # Ensure proper UTF-8 encoding for special characters
    structured_notes = structured_notes.encode("utf-8", "ignore").decode("utf-8")

    # Write multi-line text; FPDF's multi_cell wraps text automatically
    for line in structured_notes.split("\n"):
        pdf.multi_cell(0, 10, txt=line, align="L")

    # Generate the PDF in-memory properly
    pdf_bytes = pdf.output(dest='S').encode('latin1')  # Correct encoding

    return pdf_bytes
```

Figure 32: file_handler.py



```python
from pdf2image import convert_from_path
import re
import logging
from tqdm import tqdm
from docx import Document
from pptx import Presentation
from autocorrect import Speller
import fitz  # PyMuPDF
import tabula
import pytesseract
import logging
# Configure logging
logging.basicConfig(level=logging.INFO,
                    format="%(asctime)s - %(levelname)s - %(message)s")

# Initialize the spell checker
spell = Speller(lang="en")


def extract_text_and_tables_from_word(docx_path):
    """
    Extract text and tables from Word (.docx) documents.
    """
    try:
        text = ""
        doc = Document(docx_path)
        for para in doc.paragraphs:
            if para.text.strip():
                text += para.text.strip() + "\n"
        for table in doc.tables:
            text += convert_table_to_paragraph(table) + "\n"
        logging.info(f"Extracted content from Word document: {docx_path}")
        return text
    except Exception as e:
        logging.error(f"Failed to extract content from Word document: {e}")
        raise RuntimeError(f"Error processing Word document: {docx_path}")


def extract_text_and_tables_from_pptx(pptx_path):
    """
    Extract text and tables from PowerPoint (.pptx) slides.
    """
    try:
        text = ""
        presentation = Presentation(pptx_path)
        for slide in presentation.slides:
            for shape in slide.shapes:
                if shape.has_text_frame:
                    for paragraph in shape.text_frame.paragraphs:
                        if paragraph.text.strip():
                            text += paragraph.text.strip() + "\n"
                if shape.has_table:
                    table = shape.table
                    text += convert_table_to_paragraph_from_pptx(table) + "\n"
        logging.info(
            f"Extracted content from PowerPoint presentation: {pptx_path}")
        return text
    except Exception as e:
        logging.error(
            f"Failed to extract content from PowerPoint presentation: {e}")
        raise RuntimeError(
            f"Error processing PowerPoint presentation: {pptx_path}")


def extract_text_from_txt(txt_path):
    """
    Extract text from plain text (.txt) files.
    """
    try:
        with open(txt_path, 'r', encoding='utf-8') as file:
            text = file.read()
        logging.info(f"Extracted text from TXT file: {txt_path}")
        return text
    except Exception as e:
        logging.error(f"Failed to extract text from TXT file: {e}")
        raise RuntimeError(f"Error processing TXT file: {txt_path}")


def convert_table_to_paragraph(table):
    """
    Convert Word table content into readable paragraph text.
    """
    try:
        sentences = []
        headers = [cell.text.strip() for cell in table.rows[0].cells]
        for row in table.rows[1:]:
            row_data = [cell.text.strip() for cell in row.cells]
            if len(headers) == len(row_data):
                sentence = ", ".join(
                    f"{headers[i]}: {row_data[i]}" for i in
range(len(headers)))tences.append(sentence)
        return "\n".join(sentences)
    except Exception as e:
        logging.error(f"Failed to convert Word table to paragraph: {e}")
        return f"Error processing table."
```

Figure 33: text_extraction_service.py

```python
import logging
import io
from gtts import gTTS

# Configure logging
logging.basicConfig(level=logging.INFO,
                    format="%(asctime)s - %(levelname)s - %(message)s")


def text_to_speech(text, output_buffer):
    """
    Convert text to speech using gTTS and save it to an in-memory buffer.

    Args:
        text (str): Text to convert into speech.
        output_buffer (io.BytesIO): The buffer to store the MP3 file.
    """
    try:
        # Generate speech from text
        tts = gTTS(text, lang="en")

        # Save to buffer instead of file
        tts.write_to_fp(output_buffer)

        # Move buffer cursor to the beginning
        output_buffer.seek(0)

        logging.info("Voice file generated successfully in memory.")

    except Exception as e:
        logging.error(f"Failed to convert text to speech: {e}", exc_info=True)
        raise RuntimeError("Error in text-to-speech conversion.")
```

Figure 34: voice_service.py

2.1.4.5 Frontend Development with HTML, CSS, and JavaScript (React.js)

The user interface for the Summarization System is developed with React.js, complemented by Tailwind CSS for utility-focused styling, Framer Motion for animations, and Heroicons & React Icons for improved visual functionality. It connects with the backend API, enabling users to upload documents, create summaries, translate text, and utilize voice output all within an engaging and intuitive interface.

1. Core Layout and Structure (Summarization.jsx)

- Acts as the main page combining all features: Hero Banner, Document Summarization, Topic-based Summary, and Note Generation.
- Each section is modularized into its own React component:
  - Hero.jsx
  - SummarizeDocument.jsx
  - TopicSummary.jsx
  - GenerateNotes.jsx
- Scroll references allow smooth in-page navigation when clicking "Start Exploring."

Figure 35 shows the Summarization.jsx code structure, while Figure 36 shows the overall UI layout rendered on the homepage.

```jsx
import React, { useRef } from "react";
import Hero from "./Hero";
import SummarizeDocument from "./SummarizeDocument";
import TopicSummary from "./TopicSummary";
import GenerateNotes from "./GenerateNotes";

const Summarization = () => {
  const summarizeRef = useRef(null);
  const topicSummaryRef = useRef(null); // Ref for
TopicSummary
  const scrollToSummarize = () => {
    if (summarizeRef.current) {
      const yOffset = -80;
      const y =
        summarizeRef.current.getBoundingClientRect().top +
        window.scrollY +
        yOffset;
      window.scrollTo({ top: y, behavior: "smooth" });
    }
  };

  return (
    <div className="bg-gray-100">
      {/* Pass scroll function to Hero */}
      <Hero scrollToSummarize={scrollToSummarize} />

      {/* Sections */}
      <div ref={summarizeRef} className="mt-10">
        <SummarizeDocument />
      </div>

      {/* Add ref to Topic Summary */}
      <div ref={topicSummaryRef} className="mt-10">
        <TopicSummary />
      </div>

      <div ref={topicSummaryRef} className="mt-10">
        <GenerateNotes />
      </div>
    </div>
  );
};

export default Summarization;
```

Figure 35: Summarization.jsx

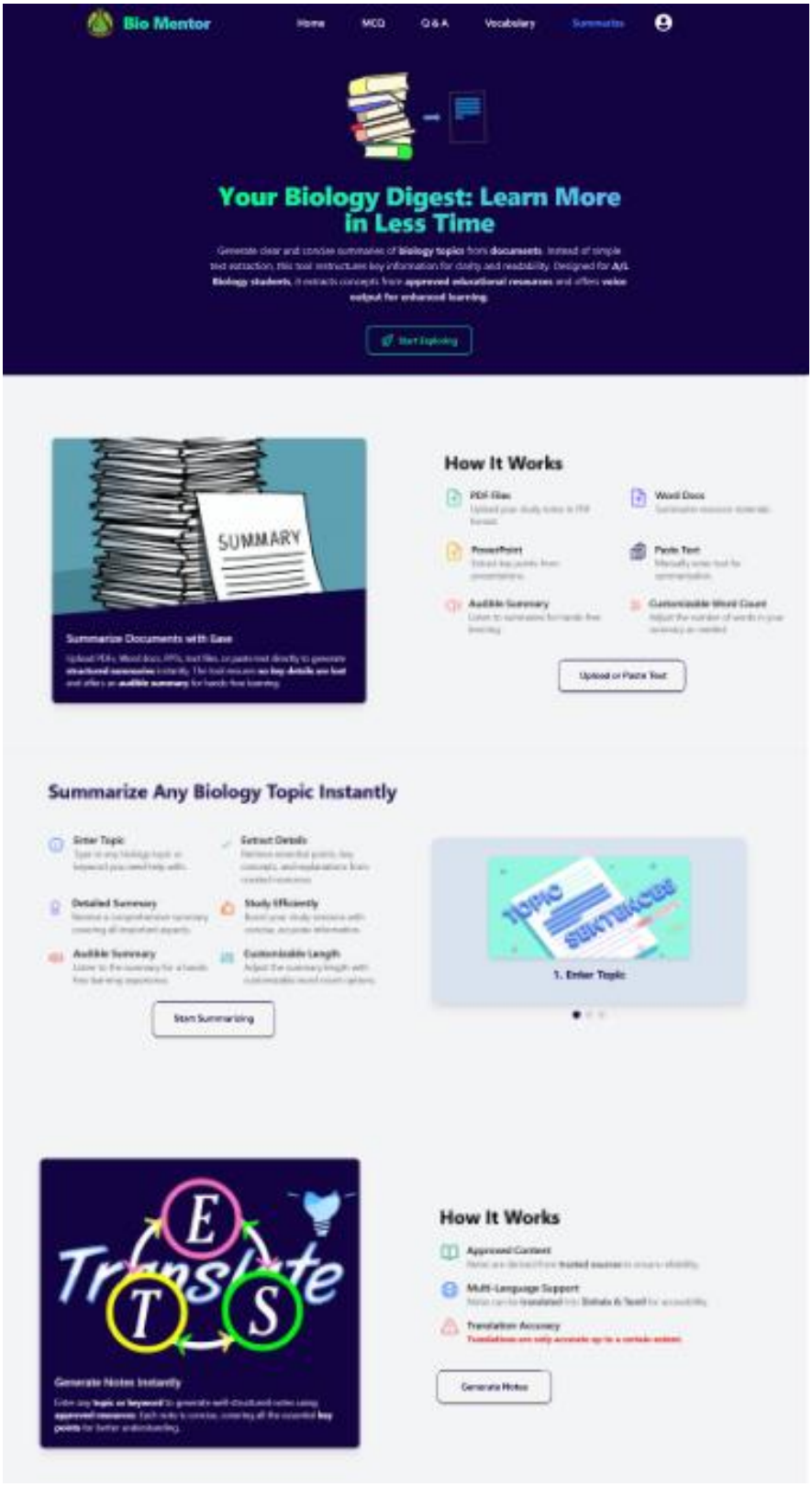
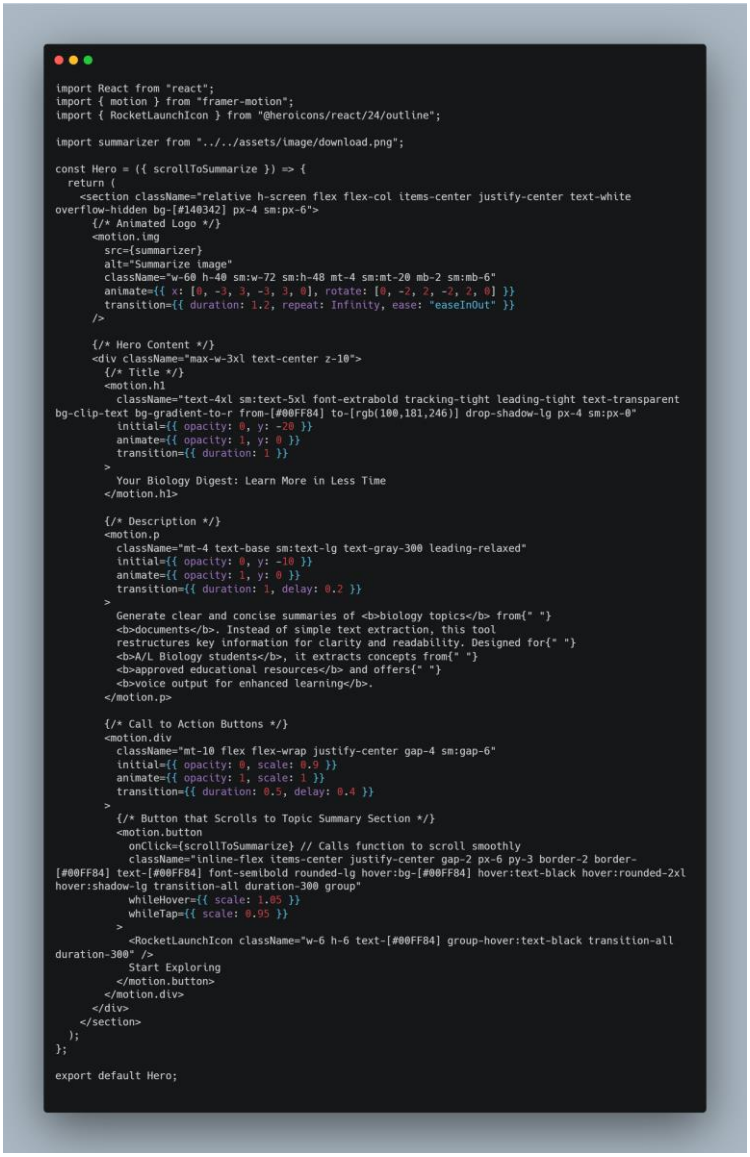


Figure 36: Summarization Home Page UI

2. Hero Section (Hero.jsx)

- Purpose: Introduces the tool with animation, a floating biology image, and bold typography.
- Animations: Image wiggles using Framer Motion to create engagement.
- CTA Button: “Start Exploring” scrolls to the summarization section.

Figure 37 shows the Hero.jsx code structure, while Figure 38 shows the UI layout rendered on the hero section



```jsx
import React from "react";
import { motion } from "framer-motion";
import { RocketLaunchIcon } from "@heroicons/react/24/outline";

import summarizer from "../../assets/image/download.png";

const Hero = ({ scrollToSummarize }) => {
  return (
    <section className="relative h-screen flex flex-col items-center justify-center text-white
overflow-hidden bg-[#140342] px-4 sm:px-6">
      {/* Animated Logo */}
      <motion.img
        src={summarizer}
        alt="Summarize image"
        className="w-60 h-40 sm:w-72 sm:h-48 mt-4 sm:mt-20 mb-2 sm:mb-6"
        animate={{ x: [0, -3, 3, -3, 3, 0], rotate: [0, -2, 2, -2, 2, 0] }}
        transition={{ duration: 1.2, repeat: Infinity, ease: "easeInOut" }}
      />

      {/* Hero Content */}
      <div className="max-w-3xl text-center z-10">
        {/* Title */}
        <motion.h1
          className="text-4xl sm:text-5xl font-extrabold tracking-tight leading-tight text-transparent
bg-clip-text bg-gradient-to-r from-[#00FF84] to-[rgb(100,181,246)] drop-shadow-lg px-4 sm:px-0"
          initial={{ opacity: 0, y: -20 }}
          animate={{ opacity: 1, y: 0 }}
          transition={{ duration: 1 }}
        >
          Your Biology Digest: Learn More in Less Time
        </motion.h1>

        {/* Description */}
        <motion.p
          className="mt-4 text-base sm:text-lg text-gray-300 leading-relaxed"
          initial={{ opacity: 0, y: -10 }}
          animate={{ opacity: 1, y: 0 }}
          transition={{ duration: 1, delay: 0.2 }}
        >
          Generate clear and concise summaries of <b>biology topics</b> from{" "}
          <b>documents</b>. Instead of simple text extraction, this tool
          restructures key information for clarity and readability. Designed for{" "}
          <b>A/L Biology students</b>, it extracts concepts from{" "}
          <b>approved educational resources</b> and offers{" "}
          <b>voice output for enhanced learning</b>.
        </motion.p>

        {/* Call to Action Buttons */}
        <motion.div
          className="mt-10 flex flex-wrap justify-center gap-4 sm:gap-6"
          initial={{ opacity: 0, scale: 0.9 }}
          animate={{ opacity: 1, scale: 1 }}
          transition={{ duration: 0.5, delay: 0.4 }}
        >
          {/* Button that Scrolls to Topic Summary Section */}
          <motion.button
            onClick={scrollToSummarize} // Calls function to scroll smoothly
            className="inline-flex items-center justify-center gap-2 px-6 py-3 border-2 border-
[#00FF84] text-[#00FF84] font-semibold rounded-lg hover:bg-[#00FF84] hover:text-black hover:rounded-2xl
hover:shadow-lg transition-all duration-300 group"
            whileHover={{ scale: 1.05 }}
            whileTap={{ scale: 0.95 }}
          >
            <RocketLaunchIcon className="w-6 h-6 text-[#00FF84] group-hover:text-black transition-all
duration-300" />
            Start Exploring
          </motion.button>
        </motion.div>
      </div>
    </section>
  );
};

export default Hero;
```

Figure 37: Hero.jsx

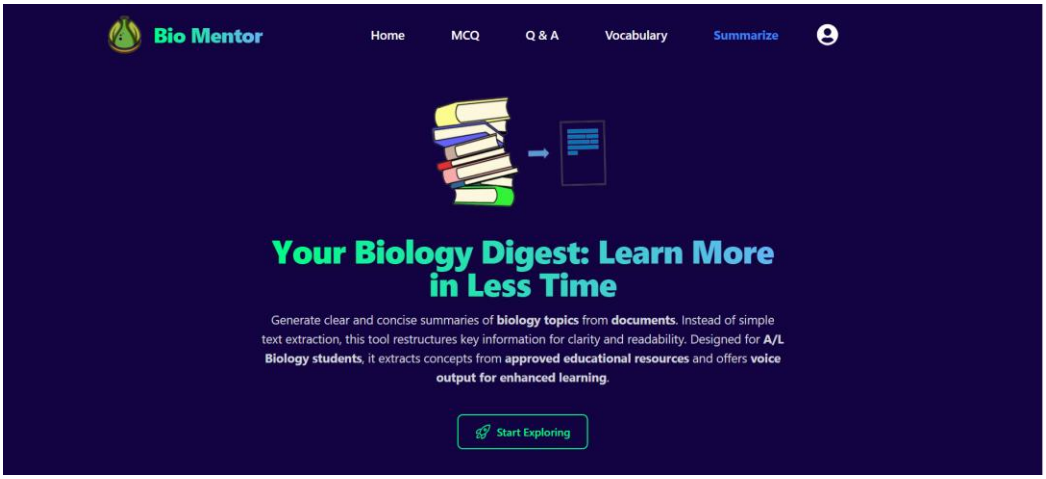


Figure 38: Hero Section UI

3. Document Summarization Section (SummarizeDocument.jsx)

- Function: Allows users to upload files (PDF, DOCX, PPTX, TXT) or paste custom text for summarization.
- Interface:
    - Rich feature cards (icons with descriptions).
    - Animated buttons that open UploadModal.jsx.

Modal (UploadModal.jsx)

- Two Tabs:
    - Document Upload for file summarization.
    - Text Input for direct content summarization.
- Custom Word Count input for controlling summary length.
- MP3 Audio Playback & Download using an in-browser audio player.
- PDF Summary Download.
- Copy-to-Clipboard functionality.
- Playback Speed Control (0.25x to 2x).
- Real-time Alerts for feedback and errors.

Figures 39-42 shows the code structure and the UI layout of the above section.

```jsx
import React, { useState } from "react";
import {
  DocumentArrowUpIcon,
  SpeakerWaveIcon,
  ClipboardDocumentListIcon,
  AdjustmentsHorizontalIcon,
} from "@heroicons/react/24/outline";
import uploadImg from "../../assets/image/doc-summarize.jpeg";
import { motion } from "framer-motion";
import UploadModal from "./UploadModal"; // Import the modal component

const SummarizeDocument = () => {
  const [isModalOpen, setIsModalOpen] = useState(false); // State for modal

  return (
    <section className="relative bg-gray-100 px-6 sm:px-12 py-16">
      {/* Content Wrapper */}
      <div className="relative flex flex-col sm:flex-row items-center justify-between max-w-7xl mx-
auto">
        {/* Left Side - Image & Description with Gradient Background */}
        <div className="w-full sm:w-1/2 flex justify-center relative">
          <div className="relative bg-[#140342] shadow-lg rounded-lg overflow-hidden w-full max-w-lg">
            {/* Image with diagonal cut effect */}
            <div className="relative">
              <img
                src={uploadImg}
                alt="Upload Document"
                className="w-full h-72 object-cover"
                style={{
                  clipPath: "polygon(0% 0%, 100% 0%, 100% 85%, 0% 100%)",
                }}
              />
            </div>

            {/* Text Section */}
            <div className="p-6 text-black p-6 bg-[#140342] text-white">
              <h3 className="text-lg font-semibold">
                Summarize Documents with Ease
              </h3>
              <p className="mt-2 text-sm">
                Upload PDFs, Word docs, PPTs, text files, or paste text directly
                to generate <b>structured summaries</b> instantly. The tool
                ensures <b>no key details are lost</b> and offers an{" "}
                <b>audible summary</b> for hands-free learning.
              </p>
            </div>
          </div>
        </div>

        {/* Right Side - Upload Info & Features */}
        <div className="w-full sm:w-1/2 flex flex-col justify-center sm:pl-16 mt-10 sm:mt-0">
          <h2 className="text-3xl font-bold">How It Works</h2>

          <div className="mt-6 grid grid-cols-2 gap-6">
            {[
              {
                icon: (
                  <DocumentArrowUpIcon className="w-8 h-8 text-green-400" />
                ),
                label: "PDF Files",
                desc: "Upload your study notes in PDF format.",
              },
              {
                icon: (
                  <DocumentArrowUpIcon className="w-8 h-8 text-[#6440FB]" />
                ),
                label: "Word Docs",
                desc: "Summarize resource materials.",
              },
              {
                icon: (
                  <DocumentArrowUpIcon className="w-8 h-8 text-[#FFAA00]" />
                ),
                label: "PowerPoint",
                desc: "Extract key points from presentations.",
              },
              {
                icon: (
                  <ClipboardDocumentListIcon className="w-8 h-8 text-[#140342]" />
                ),
                label: "Paste Text",
                desc: "Manually enter text for summarization.",
              },
              {
                icon: <SpeakerWaveIcon className="w-8 h-8 text-rose-400" />,
                label: "Audible Summary",
                desc: "Listen to summaries for hands-free learning.",
              },
              {
                icon: (
                  <AdjustmentsHorizontalIcon className="w-8 h-8 text-[#FF6347]" />
                ),
                label: "Customizable Word Count",
                desc: "Adjust the number of words in your summary as needed.",
              },
            ].map(({ icon, label, desc }) => (
              <div key={label} className="flex items-start space-x-3">
                {icon}
                <div>
                  <p className="font-semibold">{label}</p>
                  <p className="text-gray-600 text-sm">{desc}</p>
                </div>
              </div>
            ))}
          </div>

          {/* Upload Button */}
          <motion.div
            className="mt-10 flex flex-wrap justify-center gap-4 sm:gap-6"
            initial={{ opacity: 0, scale: 0.8 }}
            animate={{ opacity: 1, scale: 1 }}
            transition={{ duration: 0.5, delay: 0.4 }}
          >
            <motion.button
              onClick={() => setIsModalOpen(true)} // Opens Modal on Click
              className="inline-flex items-center justify-center gap-2 px-6 py-3 border-2 border-
[#140342] text-[#140342]
                font-semibold rounded-lg
                hover:bg-[#140342] hover:text-white hover:rounded-2xl hover:shadow-lg transition-all
duration-300 group"
              whileHover={{ scale: 1.05 }}
              whileTap={{ scale: 0.95 }}
            >
              Upload or Paste Text
            </motion.button>
          </motion.div>
        </div>
      </div>

      {/* Upload Modal */}
      <UploadModal isOpen={isModalOpen} onClose={() => setIsModalOpen(false)} />
    </section>
  );
};

export default SummarizeDocument;
```

Figure 39: SummarizeDocument.jsx

```jsx
import React, { useState, useEffect, useRef } from "react";
import { motion } from "framer-motion";
import {
  FaFileUpload,
  FaRegCopy,
  FaDownload,
  FaPlay,
  FaPause,
  FaVolumeUp,
  FaTimes,
  FaCheck,
} from "react-icons/fa";
import { MdOutlineClose } from "react-icons/md";
import axios from "axios";
import AlertMessage from "../Alert/Alert";
import ModalLoadingScreen from "../LoadingScreen/ModalLoadingScreen";
import { SUMMARIZE_URL } from "../util/config";

const UploadModal = ({ isOpen, onClose }) => {
  const [activeTab, setActiveTab] = useState("document");
  const [selectedFile, setSelectedFile] = useState(null);
  const [wordCount, setWordCount] = useState("");
  const [inputText, setInputText] = useState("");
  const [summary, setSummary] = useState(
    "Your summarized text will appear here..."
  );
  const [isLoading, setIsLoading] = useState(false);
  const [isSummaryGenerated, setIsSummaryGenerated] = useState(false);
  const [taskId, setTaskId] = useState(null);
  const [audioUrl, setAudioUrl] = useState(null);
  const [isPlaying, setIsPlaying] = useState(false);
  const [currentTime, setCurrentTime] = useState(0);
  const [duration, setDuration] = useState(0);
  const [volume, setVolume] = useState(1);
  const [isMediaPlayerOpen, setIsMediaPlayerOpen] = useState(false);
  const [alert, setAlert] = useState({ message: "", type: "" });
  const [copied, setCopied] = useState(false);
  const [playbackSpeed, setPlaybackSpeed] = useState(1);

  const audioRef = useRef(null);

  useEffect(() => {
    if (isOpen) {
      resetModal();
      document.body.style.overflow = "hidden";
    } else {
      document.body.style.overflow = "auto";
    }
  }, [isOpen]);

  const resetModal = () => {
    setSelectedFile(null);
    setWordCount("");
    setInputText("");
    setSummary("Your summarized text will appear here...");
    setIsSummaryGenerated(false);
    setIsLoading(false);
    setTaskId(null);
    setAudioUrl(null);
  };

  const handleFileChange = (event) => {
    setSelectedFile(event.target.files[0]);
  };

  // Safely close modal
  const handleClose = () => {
    if (typeof onClose === "function") {
      console.log("Closing modal...");
      onClose();
    } else {
      console.error("onClose is not a function!");
    }
  };

  const handleCopy = () => {
    if (summary) {
      navigator.clipboard.writeText(summary);
      setCopied(true);
      setTimeout(() => setCopied(false), 2000);
    }
  };

  const handleProcessSummary = async () => {
    setIsLoading(true);
    setIsSummaryGenerated(false);
    setSummary("Processing...");

    if (!wordCount.trim() || parseInt(wordCount) < 100) {
      setAlert({
        message: "Please enter a valid word count (min: 100).",
        type: "error",
      });
      setIsLoading(false);
      return;
    }

    const formData = new FormData();
    formData.append("word_count", wordCount);

    if (activeTab === "document") {
      if (!selectedFile) {
        setAlert({
          message: "Please upload a file to summarize.",
          type: "error",
        });
        setIsLoading(false);
        return;
      }
      formData.append("file", selectedFile);
    } else {
      if (!inputText.trim()) {
        setAlert({ message: "Please enter text to summarize.", type: "error" });
        setIsLoading(false);
        return;
      }
      formData.append("text", inputText);
    }

    try {
      const endpoint =
        activeTab === "document"
          ? `${SUMMARIZE_URL}/process-document/`
          : `${SUMMARIZE_URL}/summarize-text/`;

      const response = await axios.post(endpoint, formData, {
        headers: { "Content-Type": "multipart/form-data" },
      });
```

Figure 40: UploadModal.jsx

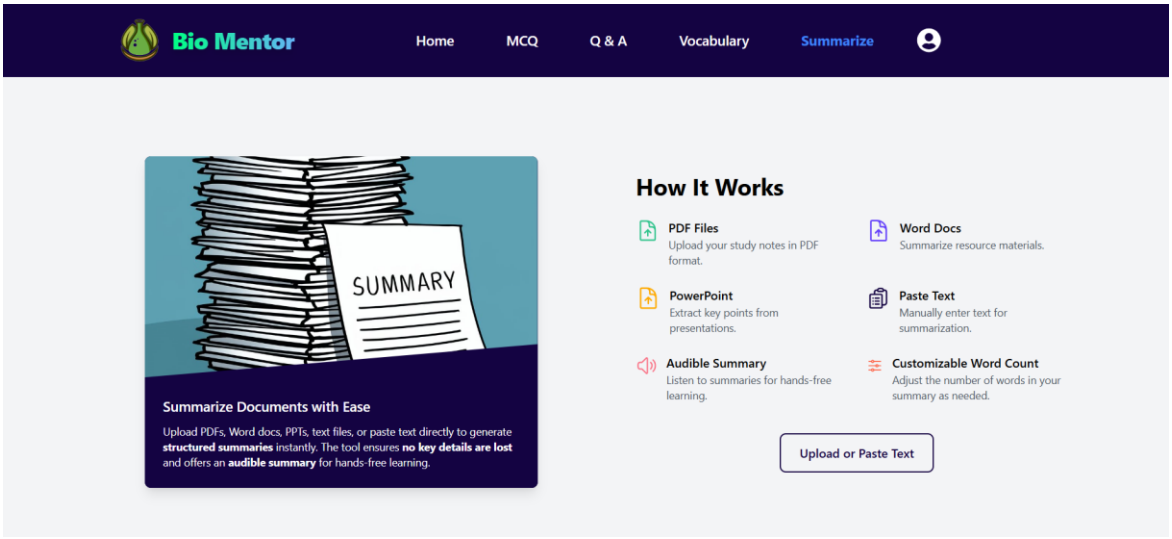


Figure 41: Summarize Document Section UI

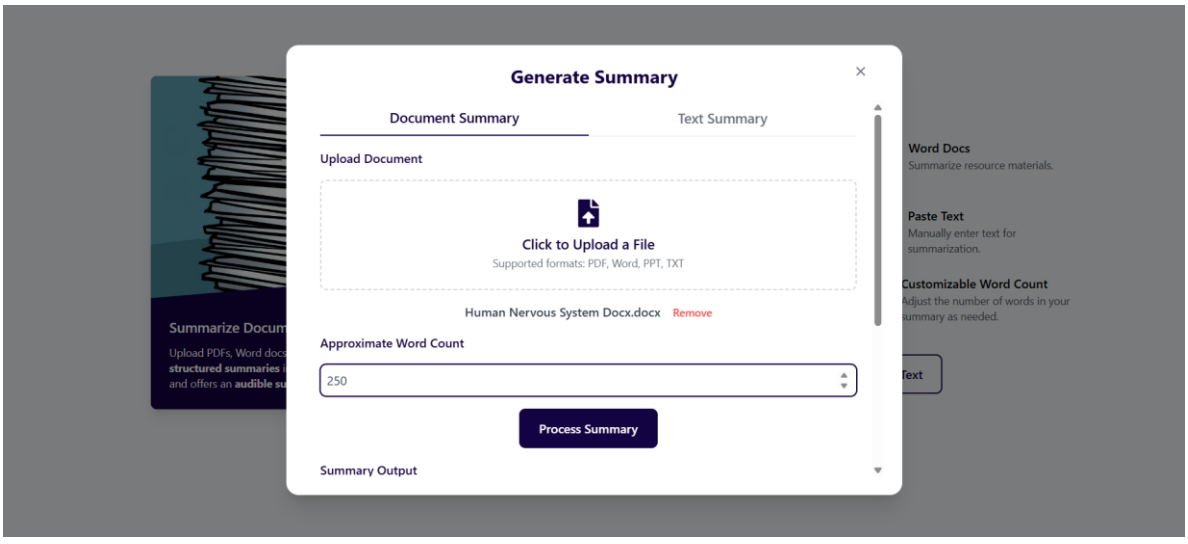


Figure 42: Upload Modal UI

4. Topic-Based Summarization (TopicSummary.jsx)

- Slideshow Guide: Uses a rotating slide to show the steps visually:
    1. Enter topic
    2. Extract relevant data
    3. Generate summary
- Feature Grid: Highlights major capabilities like:
    - Custom word count
    - Voice output
    - Concept-based retrieval
    - Accurate summaries
- Launches TopicSummaryModal.jsx to process topic-based summaries.

Modal (TopicSummaryModal.jsx)

- Accepts:
    - Topic name
    - Approximate word count
- Fetches summary using /process-query/ API endpoint.
- Allows:
    - Summary download as PDF
    - Audio playback & MP3 download
    - Playback speed and volume control
    - Media player pop-up for immersive experience

Figures 43-46 shows the code structure and the UI layout of the above section.

```jsx
import React, { useState, useEffect } from "react";
import { motion, AnimatePresence } from "framer-motion";
import {
  FiInfo,
  FiCheck,
  FiAward,
  FiThumbsUp,
  FiVolume2,
  FiSliders,
} from "react-icons/fi";
import extractImg from "../../assets/image/extract.jpg";
import summarizing from "../../assets/image/summary.jpg";
import topicImg from "../../assets/image/topic.jpeg";
import TopicSummaryModal from "./TopicSummaryModal";

// Slide data for the slideshow
const slides = [
  {
    id: 1,
    title: "1. Enter Topic",
    image: topicImg,
    bgColor: "#D9E3F0",
    textColor: "#1A0342",
  },
  {
    id: 2,
    title: "2. Extract Relevant Content",
    image: extractImg,
    bgColor: "#C1D3FE",
    textColor: "#0057D9",
  },
  {
    id: 3,
    title: "3. Generate Summary",
    image: summarizing,
    bgColor: "#90E0C6",
    textColor: "#009668",
  },
];

// Feature items data with larger icons
const features = [
  {
    icon: <FiInfo className="text-blue-500 mr-2 w-10 h-10" />,
    label: "Enter Topic",
    desc: "Type in any biology topic or keyword you need help with.",
  },
  {
    icon: <FiCheck className="text-green-500 mr-2 w-10 h-10" />,
    label: "Extract Details",
    desc: "Retrieve essential points, key concepts, and explanations from curated resources.",
  },
  {
    icon: <FiAward className="text-purple-500 mr-2 w-10 h-10" />,
    label: "Detailed Summary",
    desc: "Receive a comprehensive summary covering all important aspects.",
  },
  {
    icon: <FiThumbsUp className="text-orange-500 mr-2 w-10 h-10" />,
    label: "Study Efficiently",
    desc: "Boost your study sessions with concise, accurate information.",
  },
  {
    icon: <FiVolume2 className="text-red-500 mr-2 w-10 h-10" />,
    label: "Audible Summary",
    desc: "Listen to the summary for a hands-free learning experience.",
  },
  {
    icon: <FiSliders className="text-teal-500 mr-2 w-10 h-10" />,
    label: "Customizable Length",
    desc: "Adjust the summary length with customizable word count options.",
  },
];

// Component for individual feature item
const FeatureItem = ({ icon, label, desc }) => (
  <div classNames="flex items-start space-x-2">
    {icon}
    <div>
      <p classNames="font-semibold">{label}</p>
      <p classNames="text-gray-600 text-sm">{desc}</p>
    </div>
  </div>
);

const TopicSummary = () => {
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [currentSlide, setCurrentSlide] = useState(0);

  // Auto-slide every 5 seconds
  useEffect(() => {
    const interval = setInterval(() => {
      setCurrentSlide((prev) => (prev + 1) % slides.length);
    }, 5000);
    return () => clearInterval(interval);
  }, []);

  return (
    <section classNames="relative bg-gray-100 px-6 sm:px-20 py-16 max-w-7xl mx-auto">
      {/* Topic Heading Always on Top */}
      <h2 classNames="text-2xl font-bold text-[#1A0342] text-center sm:text-left">
        Summarize Any Biology Topic Instantly
      </h2>

      {/* Two-Column Container */}
      <div classNames="mt-8 flex flex-col sm:flex-row sm:gap-x-16 items-center justify-between">
        {/* Slider: On mobile, this comes immediately under the heading with extra bottom margin */}
        <div classNames="order-1 sm:order-2 w-full sm:w-1/2 flex justify-center items-center mt-6 mb-6
sm:mt-0 sm:mb-0">
          <div classNames="relative w-[400px] sm:w-[500px] h-auto">
            <AnimatePresence mode="wait">
              <motion.div
                key={slides[currentSlide].id}
                classNames="p-8 rounded-xl shadow-lg flex flex-col items-center text-center w-full"
                style={{
                  backgroundColor: slides[currentSlide].bgColor,
                  color: slides[currentSlide].textColor,
                }}
                initial={{ opacity: 0, x: 50 }}
                animate={{ opacity: 1, x: 0 }}
                exit={{ opacity: 0, x: -50 }}
                transition={{ duration: 0.5 }}
              >
                <img
                  src={slides[currentSlide].image}
                  alt={slides[currentSlide].title}
                  classNames="w-72 sm:w-80 h-auto rounded-lg shadow-md"
                />
                <p classNames="text-lg font-bold mt-3">
                  {slides[currentSlide].title}
                </p>
              </motion.div>
            </AnimatePresence>

            {/* Dots Indicator */}
            <div classNames="flex justify-center mt-3 space-x-2">
              {slides.map((_, index) => (
                <div
                  key={index}
                  classNames={`h-3 w-3 rounded-full ${
                    currentSlide === index ? "bg-[#1A0342]" : "bg-gray-300"
                  }`}
                />
              ))}
            </div>
          </div>
        </div>

        {/* Text Section: On mobile, this comes below the slider */}
        <div classNames="order-2 sm:order-1 w-full sm:w-1/2 flex flex-col items-center sm:items-start
text-center sm:text-left">
          {/* Two-column grid for features */}
          <div classNames="mt-6 grid grid-cols-2 gap-4">
            {features.map((feature, index) => (
              <FeatureItem
                key={index}
                icon={feature.icon}
                label={feature.label}
                desc={feature.desc}
              />
            ))}
          </div>
          {/* Centered Start Summarizing Button */}
          <motion.div
            classNames="mt-6 flex justify-center w-full"
            initial={{ opacity: 0, scale: 0.5 }}
            animate={{ opacity: 1, scale: 1 }}
            transition={{ duration: 0.5, delay: 0.2 }}
          >
            <motion.button
              onClick={() => setIsModalOpen(true)}
              classNames="px-6 py-3 border-2 border-[#1A0342] text-[#1A0342] font-semibold rounded-lg
hover:bg-[#1A0342] hover:text-white hover:rounded-2xl hover:shadow-lg transition-all duration-300
group"
              whileHover={{ scale: 1.05 }}
              whileTap={{ scale: 0.95 }}
            >
              Start Summarizing
            </motion.button>
          </motion.div>
        </div>
      </div>

      {/* Topic Summary Modal */}
      <TopicSummaryModal
        isOpen={isModalOpen}
        onClose={() => setIsModalOpen(false)}
      />
    </section>
  );
};

export default TopicSummary;
```

Figure 43: TopicSummary.jsx

```jsx
import React, { useState, useEffect, useRef } from "react";
import { motion } from "framer-motion";
import {
  FaRegCopy,
  FaDownload,
  FaPlay,
  FaPause,
  FaVolumeUp,
  FaTimes,
  FaCheck,
} from "react-icons/fa";
import { MdOutlineClose } from "react-icons/md";
import axios from "axios";
import AlertMessage from "../Alert/Alert";
import ModalLoadingScreen from "../LoadingScreen/ModalLoadingScreen";
import { SUMMARIZE_URL } from "../util/config";

const TopicSummaryModal = ({ isOpen, onClose }) => {
  const [topic, setTopic] = useState("");
  const [wordCount, setWordCount] = useState("");
  const [summary, setSummary] = useState(
    "Your summarized topic text will appear here..."
  );
  const [isLoading, setIsLoading] = useState(false);
  const [isSummaryGenerated, setIsSummaryGenerated] = useState(false);
  const [taskId, setTaskId] = useState(null);
  const [audioUrl, setAudioUrl] = useState(null);
  const [isPlaying, setIsPlaying] = useState(false);
  const [currentTime, setCurrentTime] = useState(0);
  const [duration, setDuration] = useState(0);
  const [volume, setVolume] = useState(1);
  const [isMediaPlayerOpen, setIsMediaPlayerOpen] = useState(false);
  const [alert, setAlert] = useState({ message: "", type: "" });
  const [copied, setCopied] = useState(false);
  const [playbackSpeed, setPlaybackSpeed] = useState(1);

  const audioRef = useRef(null);

  useEffect(() => {
    if (isOpen) {
      resetModal();
      document.body.style.overflow = "hidden";
    } else {
      document.body.style.overflow = "auto";
    }
  }, [isOpen]);

  const resetModal = () => {
    setTopic("");
    setWordCount("");
    setSummary("Your summarized text will appear here...");
    setIsSummaryGenerated(false);
    setIsLoading(false);
    setTaskId(null);
    setAudioUrl(null);
  };

  const handleCopy = () => {
    if (summary) {
      navigator.clipboard.writeText(summary);
      setCopied(true);
      setTimeout(() => setCopied(false), 2000);
    }
  };

  const handleProcessSummary = async () => {
    setIsLoading(true);
    setIsSummaryGenerated(false);
    setSummary("Processing...");

    if (!topic.trim() || !wordCount.trim()) {
      setAlert({
        message: "Please enter a topic and word count.",
        type: "warning",
      });
      setIsLoading(false);
      return;
    }

    try {
      const formData = new FormData();
      formData.append("query", topic);
      formData.append("word_count", wordCount);

      const response = await axios.post(
        `${SUMMARIZE_URL}/process-query/`,
        formData,
        { headers: { "Content-Type": "multipart/form-data" } }
      );

      if (response.data && response.data.summary) {
        setSummary(response.data.summary);
        setIsSummaryGenerated(true);
        setTaskId(response.data.summary_file.split("/").pop());
        // setAlert({
        //   message: "Summary generated successfully!",
        //   type: "success",
        // });
      } else {
        throw new Error("Invalid response from the server.");
      }
    } catch (error) {
      console.error("Error fetching summary:", error);

      if (error.response && error.response.data && error.response.data.detail) {
        setAlert({ message: error.response.data.detail, type: "error" });
      } else {
        setAlert({
          message: "Failed to generate summary. Try again.",
          type: "error",
        });
        resetModal();
      }
```

Figure 44: TopicSummaryModal.jsx

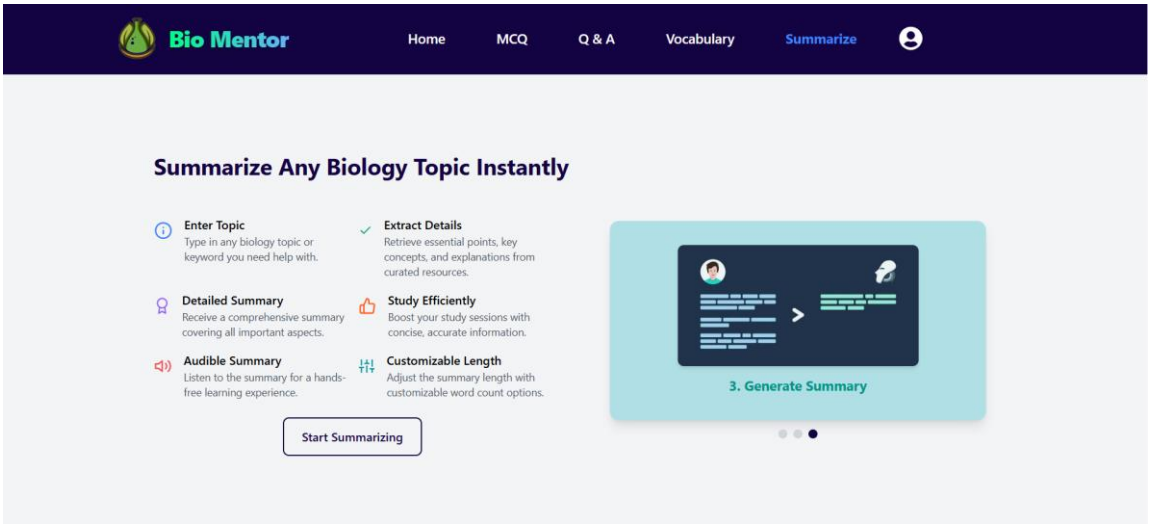


Figure 45: Topic Summary Section UI

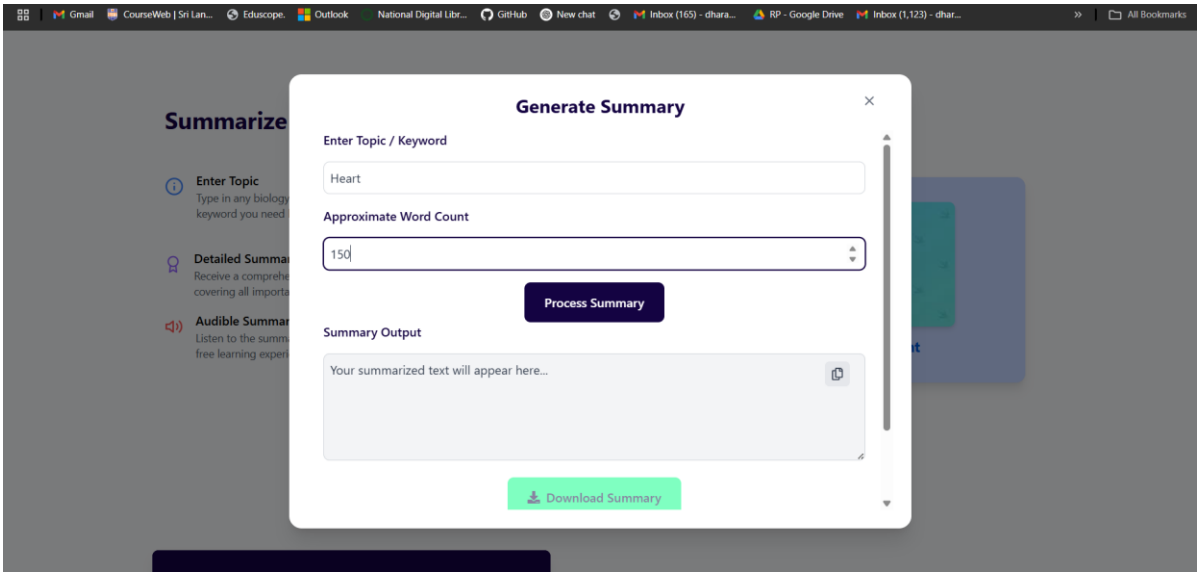


Figure 46: Topic Summary Modal UI

5. Structured Notes Generation (GenerateNotes.jsx)

- Visually Distinct Section with an image and description card (notes.png).
- Features:
  - Trusted content sources
  - Multilingual support (English, Tamil, Sinhala)
  - Translation accuracy disclaimers
- Button triggers GenerateNotesModal.jsx for topic-based note creation.

Modal (GenerateNotesModal.jsx)

- Accepts:
  - Topic/keyword
  - Language selection: English, Tamil, Sinhala
- Calls /generate-notes/ API.
- On Success:
  - Shows translated or English notes.
  - Provides PDF download for English.
- Copy-to-Clipboard button included.
- Language-sensitive output:
  - PDF download only for English
  - Sinhala and Tamil notes are shown in text only (for translation clarity)

Figures 47-50 shows the code structure and the UI layout of the above section.

```jsx
import React, { useState } from "react";
import {
  GlobeAltIcon,
  BookOpenIcon,
  ExclamationTriangleIcon,
} from "@heroicons/react/24/outline";
import notesImage from "../../assets/image/notes.png"; // Replace with actual image
import { motion } from "framer-motion";
import GenerateNotesModal from "./GenerateNotesModal"; // Import the modal

const GenerateNotes = () => {
  const [isModalOpen, setIsModalOpen] = useState(false); // Modal state

  return (
    <section className="relative bg-gray-100 px-6 sm:px-12 py-16">
      {/* Content Wrapper */}
      <div className="relative flex flex-col sm:flex-row items-center justify-between max-w-7xl mx-
auto">
        {/* Left Side - Image & Description with Gradient Background */}
        <div className="w-full sm:w-1/2 flex justify-center relative">
          <div className="relative bg-[#140342] shadow-lg rounded-lg overflow-hidden w-full max-w-lg">
            {/* Image Section with diagonal cut */}
            <div className="relative">
              <img
                src={notesImage}
                alt="Generate Notes"
                className="w-full h-80 object-cover"
              />
            </div>

            {/* Text Section */}
            <div className="p-6 bg-[#140342] text-white">
              <h3 className="text-lg font-semibold">
                Generate Notes Instantly
              </h3>
              <p className="mt-2 text-sm">
                Enter any <b>topic or keyword</b> to generate well-structured notes using{" "}
                <b>approved resources</b>. Each note is concise, covering all
                the essential <b>key points</b> for better understanding.
              </p>
            </div>
          </div>
        </div>

        {/* Right Side - Features & Generate Button */}
        <div className="w-full sm:w-1/2 flex flex-col justify-center sm:pl-16 mt-10 sm:mt-0">
          <h2 className="text-3xl font-bold">How It Works</h2>

          <div className="mt-6 space-y-4">
            {/* Feature 1 */}
            <div className="flex items-start space-x-3">
              <BookOpenIcon className="w-8 h-8 text-green-500" />
              <div>
                <p className="font-semibold">Approved Content</p>
                <p className="text-gray-600 text-sm">
                  Notes are derived from <b>trusted sources</b> to ensure
                  reliability.
                </p>
              </div>
            </div>

            {/* Feature 2 */}
            <div className="flex items-start space-x-3">
              <GlobeAltIcon className="w-8 h-8 text-blue-500" />
              <div>
                <p className="font-semibold">Multi-Language Support</p>
                <p className="text-gray-600 text-sm">
                  Notes can be <b>translated</b> into <b>Sinhala & Tamil</b> for
                  accessibility.
                </p>
              </div>
            </div>

            {/* Feature 3 */}
            <div className="flex items-start space-x-3">
              <ExclamationTriangleIcon className="w-8 h-8 text-red-500" />
              <div>
                <p className="font-semibold">Translation Accuracy</p>
                <p className="text-gray-600 text-sm">
                  <strong className="text-red-600">
                    Translations are only accurate up to a certain extent.
                  </strong>
                </p>
              </div>
            </div>
          </div>

          {/* Generate Notes Button */}
          <motion.div
            className="mt-10 flex justify-center sm:justify-start" // Center on mobile, left-align on
larger screens
            initial={{ opacity: 0, scale: 0.9 }}
            animate={{ opacity: 1, scale: 1 }}
            transition={{ duration: 0.5, delay: 0.4 }}
          >
            <motion.button
              onClick={() => setIsModalOpen(true)} // Opens Modal
              className="inline-flex items-center justify-center gap-2 px-6 py-3 border-2 border-
[#140342] text-[#140342]
    font-semibold rounded-lg
    hover:bg-[#140342] hover:text-white hover:rounded-2xl hover:shadow-lg transition-all duration-300
group"
              whileHover={{ scale: 1.05 }}
              whileTap={{ scale: 0.95 }}
            >
              Generate Notes
            </motion.button>
          </motion.div>
        </div>
      </div>

      {/* Generate Notes Modal */}
      {isModalOpen && (
        <GenerateNotesModal
          isOpen={isModalOpen}
          onClose={() => setIsModalOpen(false)}
        />
      )}
    </section>
  );
};

export default GenerateNotes;
```

Figure 47: GenerateNotes.jsx

```jsx
import React, { useState, useEffect } from "react";
import { motion } from "framer-motion";
import { FaRegCopy, FaDownload, FaCheck } from "react-icons/fa";
import { MdOutlineClose } from "react-icons/md";
import axios from "axios";
import ModalLoadingScreen from "../LoadingScreen/ModalLoadingScreen";
import AlertMessage from "../Alert/Alert"; // Import Alert Component
import { SUMMARIZE_URL } from "../util/config";

const GenerateNotesModal = ({ isOpen, onClose }) => {
  const [topic, setTopic] = useState("");
  const [language, setLanguage] = useState("english"); // Default language
  const [notes, setNotes] = useState(
    "Your generated notes will appear here..."
  );
  const [isGenerating, setIsGenerating] = useState(false);
  const [isNotesGenerated, setIsNotesGenerated] = useState(false);
  const [isLoading, setIsLoading] = useState(false);
  const [downloadLink, setDownloadLink] = useState(null);
  const [alert, setAlert] = useState({ message: "", type: "" }); // Alert state
  const [copied, setCopied] = useState(false);

  // Prevent background scrolling when modal is open
  useEffect(() => {
    if (isOpen) {
      resetModal();
      document.body.style.overflow = "hidden";
    } else {
      document.body.style.overflow = "auto";
    }

    return () => {
      document.body.style.overflow = "auto";
    };
  }, [isOpen]);

  const resetModal = () => {
    setTopic("");
    setLanguage("english");
    setNotes("Your generated notes will appear here...");
    setIsLoading(false);
  };

  const handleGenerateNotes = async () => {
    if (!topic.trim()) {
      setAlert({ message: "Please enter a topic.", type: "warning" });
      return;
    }

    setIsGenerating(true);
    setIsLoading(true);
    setNotes("Generating notes... Please wait.");
    setDownloadLink(null); // Reset download link before request

    try {
      // Prepare FormData
      const formData = new FormData();
      formData.append("topic", topic);

      // Convert language to API expected format
      let selectedLang = "";
      if (language === "tamil") {
        selectedLang = "ta";
      } else if (language === "sinhala") {
        selectedLang = "si";
      }

      // Append language if not English
      if (selectedLang) {
        formData.append("lang", selectedLang);
      }

      const response = await axios.post(
        `${SUMMARIZE_URL}/generate-notes/`,
        formData,
        {
          headers: { "Content-Type": "multipart/form-data" },
        }
      );

      if (response.data) {
        setNotes(response.data.structured_notes);
        setIsNotesGenerated(true);
        console.log(response.data)

        // Set download link ONLY if API provides it
        if (response.data.download_link) {
          setDownloadLink(response.data.download_link);
        } else {
          setDownloadLink(null);
        }
```

Figure 48: GenerateNotesModal.jsx

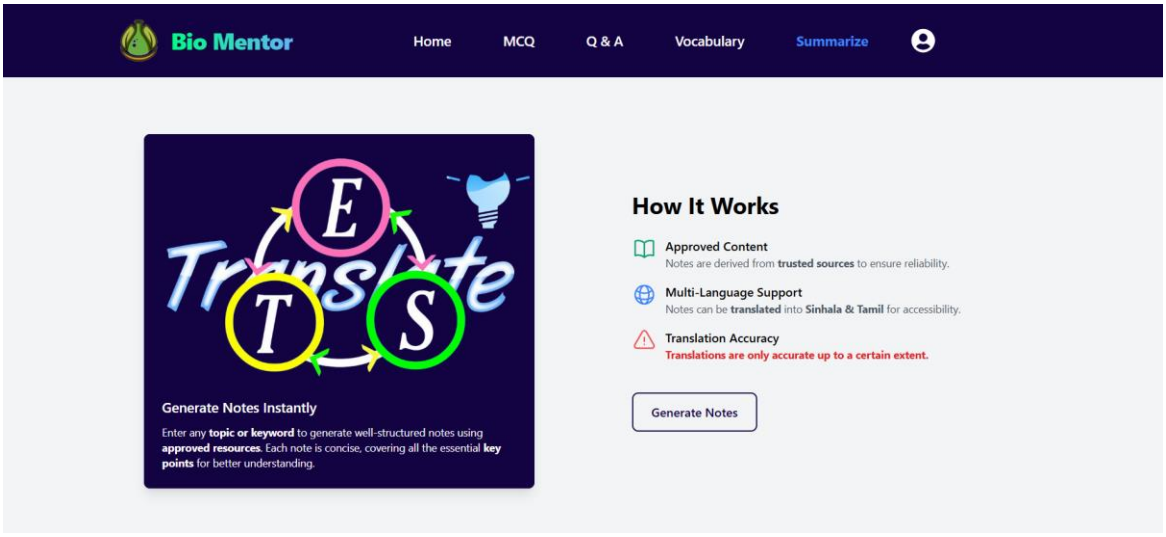


Figure 49: Generate Notes Section UI

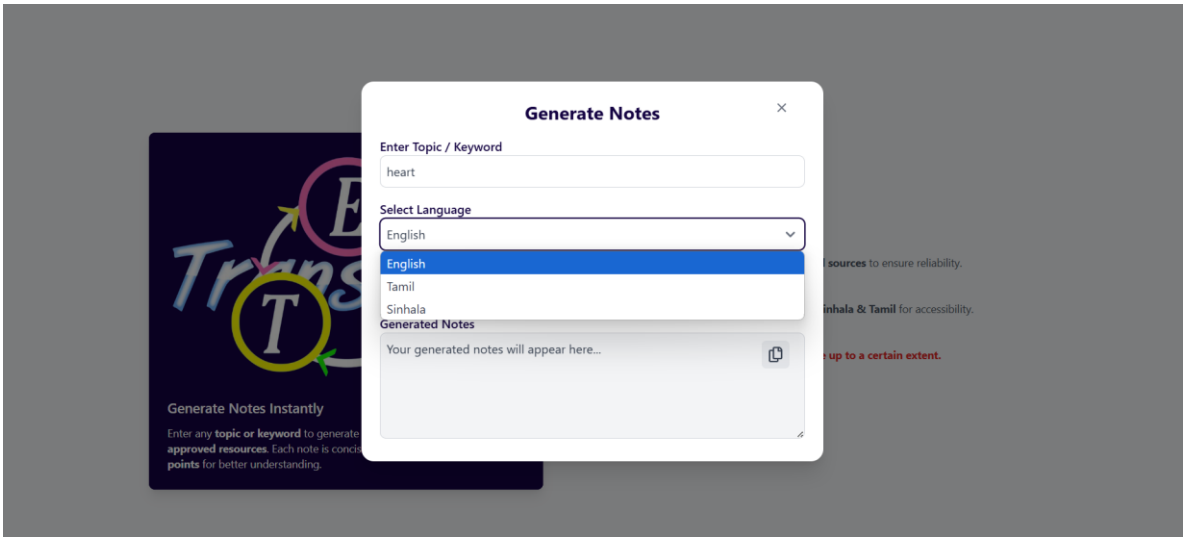


Figure 50: Generate Notes Modal UI

6. Component Utilities and UX Features

- AlertMessage: Custom component to show real-time success, error, and warning messages.
- ModalLoadingScreen: Full-screen spinner for loading feedback.
- Smooth UX:

  - All inputs and buttons have animation transitions.
  - Disabled states provide clarity on required fields.
  - Responsive layout supports all screen sizes.

The frontend plays a critical role in making the summarization system interactive, educational, and accessible. It effectively connects students to backend services via clean APIs and transforms academic tasks into a dynamic experience.

2.1.5 Testing

To guarantee the stability, reliability, and efficiency of the educational summarization system on the BioMentor platform, an extensive testing strategy was employed. This encompassed various testing levels, such as unit testing, integration testing, system testing, and acceptance testing. Every phase aims to verify particular elements of the platform, both at the individual component level and in relation to the entire application.

Unit Testing:

Unit testing serves as the essential base for the testing approach utilized in the BioMentor platform. Its goal is to validate the functionality of individual components or functions independently, without depending on external systems or integrations. This guarantees that the fundamental elements of the application are dependable, easy to maintain, and free from errors.

In the backend of BioMentor, a collection of unit tests was established using Pytest for modules like file_handler.py, text_extraction_service.py, rag.py, and voice_service.py. Each test file corresponds to distinct functionalities. For example, test_file_handler.py checks file saving and PDF creation, while test_text_extraction.py examines the cleaning, formatting, and grammar correction of unrefined text data. The test_rag.py file verifies that the summarization trimming, formatting, and post-processing functions operate correctly, and test_voice_service.py assesses the audio generation process by mimicking text-to-speech conversion using in-memory buffers.

Each test case was crafted to operate with simulated inputs and controlled outputs, providing a consistent and isolated testing environment. These tests are run frequently during development to identify regressions early and confirm the stability of the code.

The following details are the unit testing scope and accompanying visual references:

Test Framework Employed: Pytest (Python)

Modules Assessed:

- test_file_handler.py: Verifies file upload processing and PDF creation
- test_text_extraction.py: Evaluates text cleaning, formatting, and error correction
- test_rag.py: Confirms summarization word count trimming and formatting logic
- test_voice_service.py: Validates text-to-speech conversion and streaming consistency

Execution Strategy:

- Mock data utilized for input simulation
- No external services or dependencies involved
- Rapid execution, repeatable with every code modification

Outcomes:

- Guarantees the underlying reliability of backend logic
- Facilitates safe enhancement and alteration of features
- Increases developer confidence through consistent testing

Figures 51-54 show the code implementation for the above unit testing.

Back-End > Summarization > Monolithic-Architecture > tests > test_rag.py > test_truncate_to_word_count

```python
import pytest
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from rag import RAGModel

class DummyModel:
    def generate_summary_for_long_text(self, text, max_words=100):
        return "summary"

def test_truncate_to_word_count():
    text = "This is a test sentence. " * 50
    model = DummyModel()
    result = RAGModel.truncate_to_word_count(text, max_words=20)
    assert len(result.split()) <= 20

def test_postprocess_summary():
    summary = "this is a test. this is another sentence."
    model = DummyModel()
    result = RAGModel.postprocess_summary(model, summary)
    assert result[0].isupper()
    assert result.endswith(".")

```

Figure 51: test_rag.py

test_file_handler.py 2, U

Back-End > Summarization > Monolithic-Architecture > tests > test_file_handler.py > ...

```python
import pytest
from fastapi import UploadFile
from io import BytesIO
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from file_handler import save_uploaded_file, generate_pdf

def test_save_uploaded_file(tmp_path):
    dummy_content = b"Hello, this is test content."
    file = UploadFile(filename="test.txt", file=BytesIO(dummy_content))
    path = save_uploaded_file(file)
    assert os.path.exists(path)
    os.remove(path)  # Clean up

def test_generate_pdf():
    content = "Line 1\nLine 2"
    title = "Test PDF"
    result = generate_pdf(content, title)
    assert isinstance(result, bytes)
    assert result.startswith(b"%PDF")
```

Figure 52: test_file_handler.py

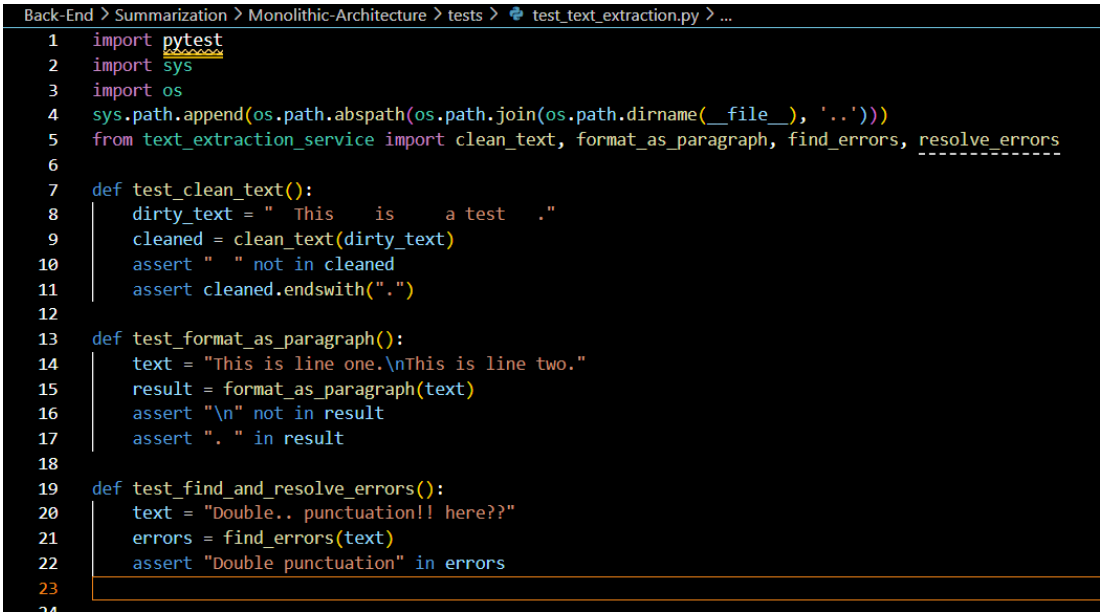


Figure 53: test_text_extraction.py

Figure 54: test_voice_service.py

Integration Testing:

Integration testing is essential for verifying the interactions among various components within the BioMentor platform. While unit testing guarantees the accuracy of single functions, integration testing ensures that these functions operate

effectively when combined, particularly across different modules and API endpoints. It confirms that backend services communicate properly and that data flows seamlessly through the entire system workflow, from handling inputs to generating files and audio outputs.

In the BioMentor platform, integration testing was executed using Pytest in conjunction with FastAPI's TestClient. The testing focused on the critical user flows of the system by emulating actual API requests to endpoints such as /summarize-text/, /process-query/, /generate-notes/, and /process-document/. These tests targeted the integrity of the combined operations between modules like summarization_functions.py, rag.py, file_handler.py, and voice_service.py.

Each integration test was meticulously crafted to replicate user behavior and assess system performance under realistic scenarios. For example,test_process_query verifies that questions based on topics activate the retrieval and summarization pipeline correctly. test_generate_notes checks the note creation and language selection processes, confirming that PDF generation and internal file tracking function as intended. In a more thorough test, test_process_document_and_download evaluates the entire document summarization process, from uploading the PDF to producing file and audio responses, including the download feature via endpoints like /download-summary-text/{task_id} and /download-summary-audio/{task_id}.

The following provides an overview of the integration testing scope and visual references:

Test Framework Employed : Pytest with FastAPI TestClient

Workflows Evaluated:

- test_summarize_text: Validates the summarization of raw text through /summarize-text/
- test_process_query: Confirms topic-based summarization and response formatting through /process-query/

- test_generate_notes: Verifies note creation, PDF generation, and file retention through /generate-notes/
- test_process_document_and_download: Replicates document uploads, response generation, and downloading functionalities through /process-document/, /download-summary-text/, and /download-summary-audio/

Execution Approach:

- Replicated real-world API interactions
- Mock file uploads (PDF) using in-memory buffer generation
- Comprehensive request-response cycle testing for PDF and audio outputs

Results:

- Validates that essential application features function cohesively
- Ensures effective communication across backend modules
- Confirms the reliability of the entire summarization, note generation, and audio output workflows
- Establishes comprehensive trust in the BioMentor system's delivery of educational content

Figure 55 shows the code implementation for integration testing. Figure 56 presents the output after successfully executing the entire test suite including the unit test cases.

```python
import os
import io
import pytest
from fastapi.testclient import TestClient
from summarization import app
from summarization_functions import file_store
from reportlab.pdfgen import canvas

client = TestClient(app)

@pytest.fixture
def dummy_pdf():
    buffer = io.BytesIO()
    c = canvas.Canvas(buffer)
    c.drawString(100, 750, (
        "Photosynthesis is a process used by plants and other organisms to convert light energy into
chemical energy. "
        "This energy is stored in carbohydrate molecules, such as sugars, which are synthesized from
carbon dioxide and water."
    ))
    c.save()
    buffer.seek(0)
    return buffer

def test_summarize_text():
    response = client.post("/summarize-text/", data={
        "text": "Photosynthesis is the process by which plants convert sunlight into chemical energy.",
        "word_count": 30
    })
    assert response.status_code == 200
    data = response.json()
    assert "summary" in data
    assert isinstance(data["summary"], str)

def test_process_query():
    response = client.post("/process-query/", data={
        "query": "Heart",
        "word_count": 50
    })
    assert response.status_code == 200
    data = response.json()
    assert "summary" in data
    assert isinstance(data["summary"], str)

def test_generate_notes():
    response = client.post("/generate-notes/", data={
        "topic": "Heart",
        "lang": "en"
    })
    assert response.status_code == 200
    data = response.json()
    assert "structured_notes" in data
    # Ensure PDF file was registered
    if "download_link" in data:
        filename = data["download_link"].split("/")[-1]
        assert filename in file_store
        assert file_store[filename].startswith(b"%PDF")

def test_process_document_and_download(dummy_pdf):
    response = client.post(
        "/process-document/",
        files={"file": ("test.pdf", dummy_pdf, "application/pdf")},
        data={"word_count": 50}
    )
    assert response.status_code == 200
    data = response.json()
    assert "summary_file" in data
    assert "voice_file" in data

    task_id = data["summary_file"].split("/")[-1]

    # Fake file_store entries if backend failed to write files
    file_store.setdefault(f"summary_{task_id}.pdf", b"PDF test data")
    file_store.setdefault(f"summary_{task_id}.mp3", b"MP3 test data")

    summary_response = client.get(f"/download-summary-text/{task_id}")
    assert summary_response.status_code == 200
    assert summary_response.headers["content-type"] == "application/pdf"

    audio_response = client.get(f"/download-summary-audio/{task_id}")
    assert audio_response.status_code == 200
    assert audio_response.headers["content-type"] == "audio/mpeg"

def test_download_notes_direct():
    filename = "notes_Cell_Division_English.pdf"
    file_store[filename] = b"PDF notes content"
    response = client.get(f"/download-notes/{filename}")
    assert response.status_code == 200
    assert response.headers["content-type"] == "application/pdf"
```

Figure 55: test_integration.py

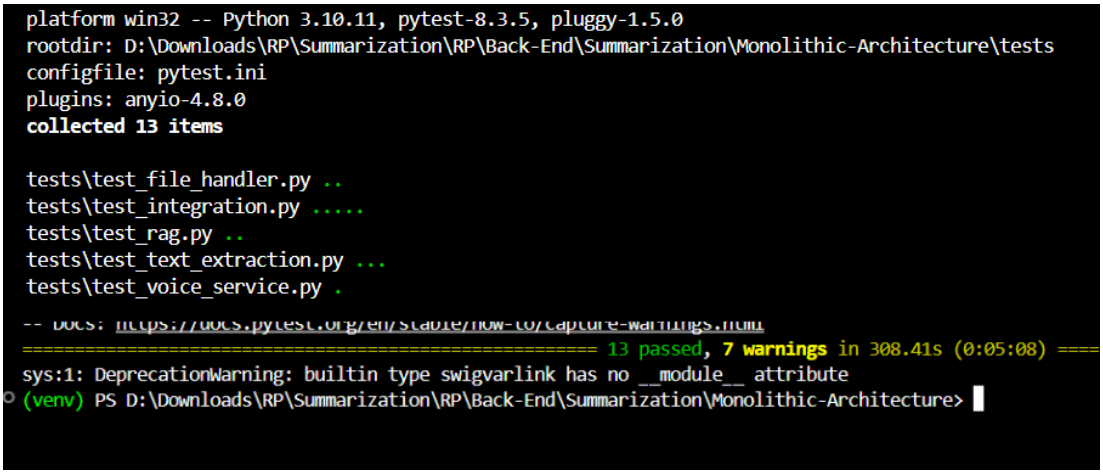


Figure 56: Test Results

System Testing:

System testing on the BioMentor platform was performed to confirm that the application behaves as a unified and integrated system. This phase of testing extends beyond the assessment of standalone modules or paired integrations it emulates actual user scenarios to ensure that all frontend and backend components collaborate effectively under realistic conditions.

For BioMentor, Postman a popular tool for API testing and simulation was utilized to conduct system testing. Through Postman, complete workflows were examined end-to-end, incorporating document uploads, topic-based summarization, multilingual note generation, and the retrieval of downloadable PDF and MP3 files. These tests are intended to replicate genuine user interactions from the frontend, validating backend responses, error management, and content correctness.

Each test case performed in Postman focused on different endpoints and scenarios. For instance, when testing the /process-document/ endpoint, sample biology documents were uploaded, and the resulting response was checked to confirm that it contained valid links to summary PDFs and audio files. Additionally, tests for /process-query/ concentrated on topic-based input, ensuring that the system conducted semantic

retrieval accurately and produced valid summaries for academic inquiries. Download endpoints were also assessed to confirm that files were served correctly with the appropriate content types and response headers.

The system's capacity to process invalid or incorrectly formatted inputs was also tested, such as submitting empty topics, uploading unsupported file formats, or inputting non-academic queries. These negative test scenarios were employed to validate the system's robustness and user safety features.

The following summarizes the system testing scope and findings:

Tool Used : Postman

Workflows Tested:

- /process-document/: Comprehensive document summarization and response verification
- /process-query/: Semantic search and summarization based on topics
- /summarize-text/: Summarization flow for raw input text
- /process-query/: Semantic search and summarization based on topics
- /generate-notes/: Generation of structured notes with translation capabilities
- /download-summary-text/, /download-summary-audio/, /download-notes/: Verification of file download.

Execution Strategy:

- Emulated complete user journeys using Postman requests
- Validated file handling, PDF/audio outputs, and support for multiple languages
- Assessed system performance using both valid and invalid inputs
- Confirmed the accuracy of response structures, status codes, and headers

Outcomes:

- Ensured seamless operation among all system components

- Verified the backend’ s readiness for genuine user interactions
- Identified and addressed edge cases and error handling challenges

Figure 57 and 58 show Postman requests made to the endpoints.

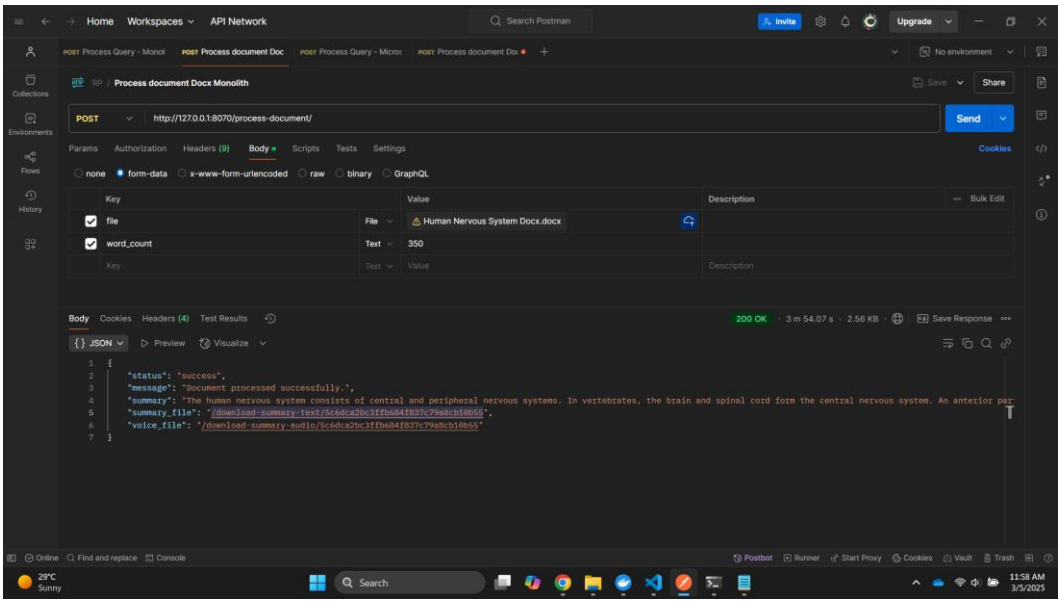


Figure 57: /process-document Postman request

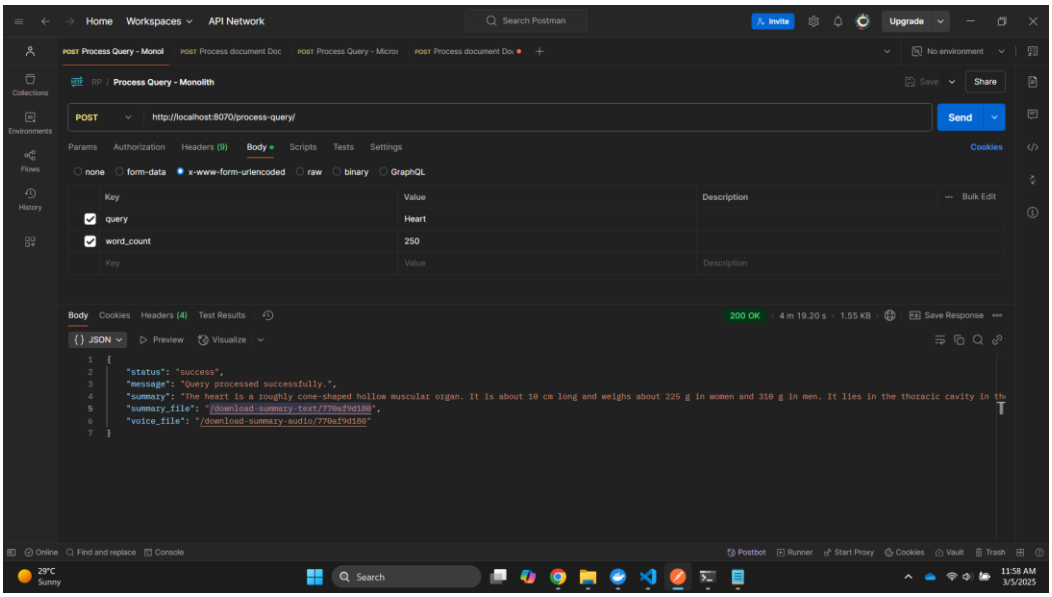


Figure 58: /process-query Postman request

Acceptance Testing:

The acceptance testing for the summarization feature of the BioMentor platform was performed with Advanced Level (A/L) Biology students to evaluate its effectiveness, precision, and usability in genuine academic environments. The objective was to ensure that the summarization, topic-based query, note generation, and translation features met the needs of students and facilitated their learning and revision processes.

The testing was conducted in two stages. In the Alpha Testing phase, a select group of A/L Biology students utilized the platform in a controlled setting. Their feedback contributed to enhancing the summarization output, clarifying user interactions, and improving the overall flow of the component.

During the Beta Testing phase, students accessed BioMentor on their personal devices within real school environments. They evaluated key features such as uploading documents for summarization, submitting topic-based queries, generating notes, and employing the audio and translation functions. These sessions were essential in confirming the platform's performance in actual usage conditions.

Feedback from students indicated that the summarization component was accurate, user-friendly, and beneficial for rapid learning and exam preparation, especially for distilling complex biology topics into easily understandable summaries.

Manual Testing:

Manual testing plays a fundamental role in the testing process for the BioMentor platform, as testers assessed the summarization system by engaging with each feature manually, without relying on automation tools. This approach aided in confirming both the accuracy of the outputs and the general usability and flow of the application from the viewpoint of an actual student.

Purpose of Manual Testing:

- To validate the functionality, performance, and usability of the summarization, query handling, and note generation components.
- To simulate real student interactions, ensuring features behave as intended.
- To detect interface-level issues or inconsistencies not captured during automated tests.

Test Case Development:

- Test cases were designed based on typical A/L Biology student workflows.
- Each test scenario included objective, step-by-step instructions, expected outcomes, and actual results.

Execution:

- Testers interacted with the frontend (React) interface and backend API via forms and download links.
- Outputs such as PDF summaries, MP3 audio files, and translated notes were manually reviewed for accuracy, formatting, and accessibility.

Types of Manual Testing:

- **Functional Testing**:
  Verified whether the summarization, topic query, note generation, translation, and download features functioned correctly according to system requirements.
- **Non-Functional Testing**:
  Assessed system responsiveness, audio playback quality, and file generation performance under normal usage conditions.
- **User Acceptance Testing (UAT)**:
  Involved actual A/L Biology students using the platform to confirm that features met their study and revision needs effectively.

- **Exploratory Testing**:
  Testers navigated the system freely without predefined scripts to uncover unexpected behavior or usability issues.
- **Compatibility Testing**:
  Checked the platform's performance across multiple browsers (e.g., Chrome, Firefox) and devices (desktops, laptops, and mobile phones) to ensure consistent functionality.

Below are the test cases which were done for the manual testing. Tables 4-9 display the manual test cases.

Table 4: Test case of the file upload and summarization

Test Case ID	TC_01
Test Case Objective	Test file upload and summarization
Pre-Requirements	Valid user, system online
Test Steps	1. Login to the platform 2. Open the "Summarize Document" section 3. Upload a biology file and set word count 4. Click "Summarize"
Test Data	A/L Biology content file (Human Nervous System Docx) and wordcount-250
Expected Output	Generates summary and provides PDF and MP3 download links
Actual Output	Summary generated with downloadable files
Status	Pass

Table 5: Test case of the query-based summarization

Test Case ID	TC_02
Test Case Objective	Test topic-based query summarization
Pre-Requirements	Valid user, system online
Test Steps	1. Login to the platform 2. Go to the "Topic Summary" section 3. Enter a topic and set word count 4. Submit the request
Test Data	Topic input – "Heart" and wordcount-250
Expected Output	Generates summary and provides download options
Actual Output	Accurate topic summary with files
Status	Pass

Table 6: Test case of the audio generation

Test Case ID	TC_03
Test Case Objective	Test audio generation
Pre-Requirements	Valid user, system online
Test Steps	1. Use any summarization feature 2. Click the MP3 player or download audio
Test Data	Play button of audio summary
Expected Output	Audio plays the summarized content
Actual Output	Audio successfully played
Status	Pass

Table 7: Test Case of Download summary

Test Case ID	TC_04
Test Case Objective	Test download feature for PDF/MP3
Pre-Requirements	Valid user, system online
Test Steps	1. Complete a summarization or note generation task 2. Click the "Download Summary" or "Download Audio" buttons
Test Data	Download buttons
Expected Output	Files are downloaded in correct formats
Actual Output	PDF and MP3 downloaded successfully
Status	Pass

Table 8: Test case for the notes generation

Test Case ID	TC_05
Test Case Objective	Test structured note generation
Pre-Requirements	Valid user, system online
Test Steps	1. Login to the system. 2. Open the Stress Detection Tab 3. Click the End Call button
Test Data	Topic input – "Heart" and Language-English
Expected Output	Returns notes and file download (PDF for English)
Actual Output	Notes generated with appropriate language output
Status	Pass

Table 9: Test case for the Tamil notes generation

Test Case ID	TC_06
Test Case Objective	Test multilingual translation (Tamil)
Pre-Requirements	Valid user, system online
Test Steps	1. Go to Generate Notes 2. Select topic and choose Tamil 3. Submit
Test Data	Topic input – “Heart” and Language-English
Expected Output	Notes returned in Tamil
Actual Output	Tamil notes generated
Status	Pass

2.1.6 Deployment & Maintenance

Deployment:

The deployment process of the BioMentor summarization module was carried out in a carefully structured manner to ensure reliability, scalability, and consistent performance. The goal was to make the summarization service accessible to users in real-time, particularly for A/L Biology students using the BioMentor platform.

1. Model Hosting on Hugging Face Hub

   The first step in the deployment pipeline was hosting the fine-tuned Flan-T5 model on Hugging Face Hub enabling seamless loading and inference in the backend. This external hosting simplifies model access and version control while offloading model storage from the server itself.

2. Provisioning the Azure Virtual Machine

   A Linux-based Virtual Machine (VM) was established on Microsoft Azure within the dedicated resource group titled BioMentor-Summarization-Resource-Group. This virtual machine was set up to support the backend of the BioMentor summarization component and to handle inference requests from the frontend. The chosen deployment area was Central India (Zone 1) to enhance latency for local users while utilizing Microsoft's cloud infrastructure for dependable availability and uptime.

   The VM was configured with the following specifications to support resource-intensive tasks like document parsing, query handling, and model inference:

   - Operating System: Ubuntu Linux
   - VM Size: *Standard D4s v3*
     - This configuration includes 4 virtual CPUs (vCPUs) and 16 GB of RAM, which provides a solid foundation for hosting a FastAPI application and supporting concurrent user requests without performance degradation.

- VM Generation: *Generation V2*
    - Offers improved security features, boot performance, and compatibility with newer virtual machine images.
- Architecture: *x64*
    - Ensures compatibility with the backend stack and Python runtime environments used in the summarization service.
- Public IP Address: *20.193.248.25*
    - This static IP address allows external clients (including frontend components and Postman testers) to reach the summarization API hosted on the VM.

These specifications were carefully selected to maintain a balance between cost efficiency, compute performance, and reliability, particularly for a monolithic backend handling real-time text summarization, structured note generation, and audio synthesis.

As shown in Figure 60, Azure Portal offers a detailed summary of the VM's setup.

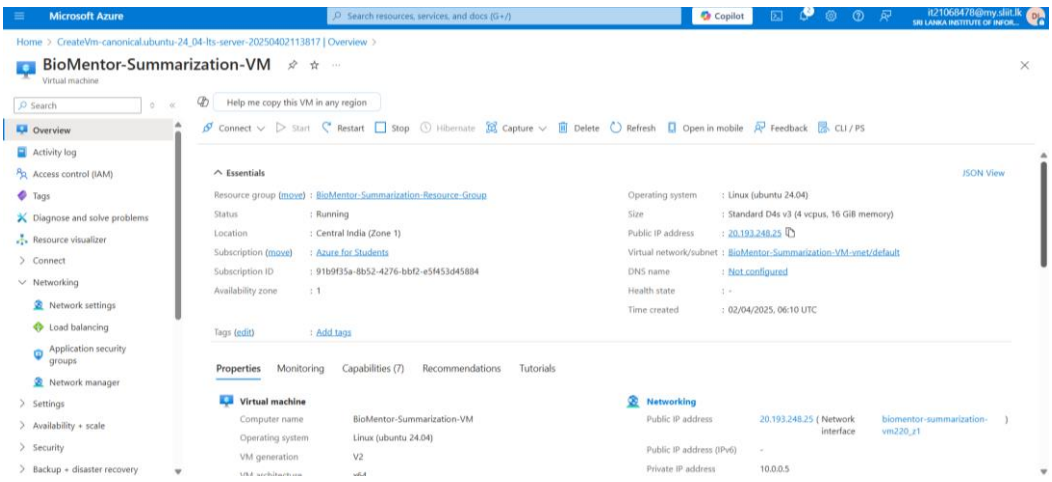


Figure 60: Microsoft Azure VM Setup

3. Virtual Machine Configuration and Runtime Setup

After provisioning the Azure Virtual Machine, the environment was set up to effectively run the BioMentor summarization backend. Secure authentication via SSH keys was established to facilitate remote access, while firewall configurations were managed using Azure's Network Security Group (NSG) to ensure that only authorized users could connect.

Upon accessing the VM, system packages received updates, and critical runtime components such as Python 3.10 and Java 11 were installed to support the backend along with its dependencies. A virtual environment was established to ensure an isolated and conflict-free workspace for the application.

The entire backend project was cloned from the Git repository onto the VM. This included all necessary code, configuration files, and dependency requirements needed to operate the summarization service. Figure 61 shows the cloning of the GitHub repository and the setup of the Python virtual environment. This setup is a foundational step before initiating the installation of required Python libraries listed in the project's requirements file.

Following the environment setup, the system was fully equipped to provide the summarization component. The setup guaranteed reliability, performance, and consistency with the development structure.

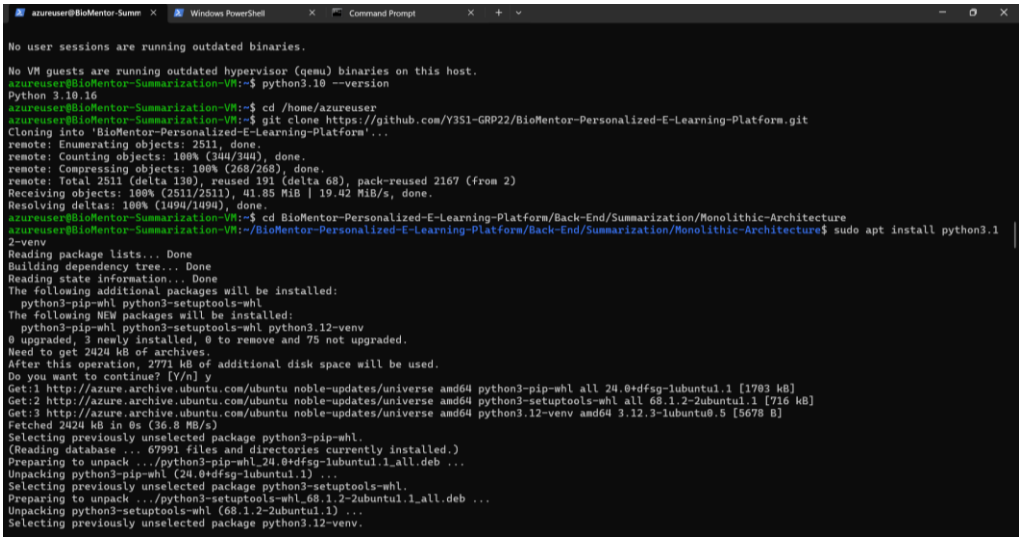


Figure 61: Repository Cloning and Virtual Environment Setup on Azure VM

4. Service Hosting and Port Configuration

   After configuring the runtime environment and cloning the backend code onto the Azure VM, the BioMentor summarization service was launched using a FastAPI application. This backend was intended to manage tasks like processing uploaded documents, generating summaries based on topics, creating structured notes, and generating audio outputs.

   To ensure the service runs persistently and reliably, it was set up as a background system service through systemd. This setup guaranteed that the summarization API would automatically start on boot, restart if it encountered a crash, and operate continuously in the background without requiring manual oversight. The service was directly linked to the virtual environment and the directory housing the backend codebase, ensuring that all dependencies were properly loaded during execution.

   The FastAPI application was configured to operate on port 8002, a specific choice made to segregate backend API traffic from other system services. In order to enable public access to the summarization service, a custom inbound rule was established within Azure's Network Security Group (NSG). This rule, named AllowAnyCustom8002Inbound, permitted external HTTP traffic to access the backend via port 8002 while keeping all other ports securely restricted.

   This setup is clearly illustrated in Figure 61, which displays the NSG rules. The rule provides unrestricted access to port 8002, allowing external systems such as the frontend interface or testing tools like Postman to engage with the summarization API.

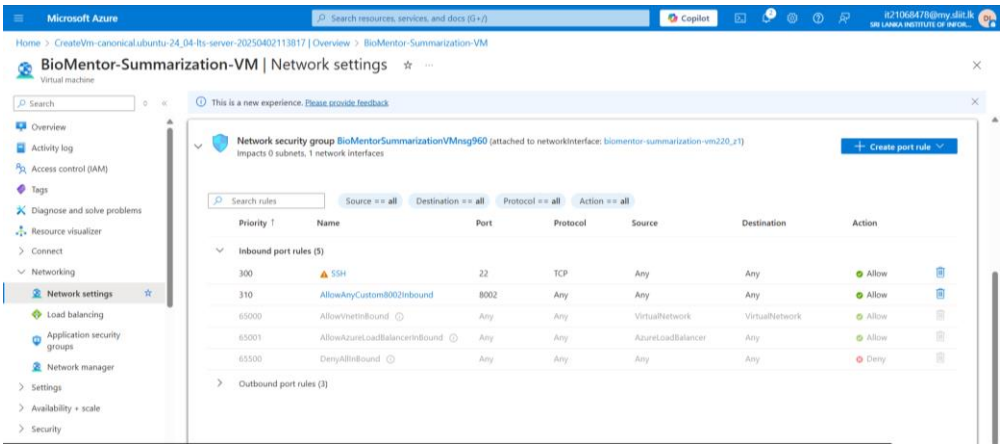


Figure 62: Microsoft Azure Port Rule Setup

With this hosting approach, the BioMentor backend is not only responsive and accessible over the internet but also strong and secure, capable of delivering consistent performance under regular usage by A/L Biology students utilizing the summarization and note-generation features in real time.

5. Service Management and Auto-Restart

   In order to maintain high availability and dependable functioning of the BioMentor summarization backend, the application was set up to run as a service managed by systemd on the Azure Virtual Machine. This configuration offered a solid and production-ready hosting solution by combining the service with the native Linux process manager.

   By registering the summarization API as a persistent background service, it was enabled to:

   - Automatically start when the VM reboots, eliminating the need for manual intervention after updates or power outages.
   - Automatically recover from failures should the service crash or terminate unexpectedly, systemd restarts it according to established restart policies.

- Function independently of terminal sessions, allowing it to continue running even after the SSH session has been closed.

The service was established with its own configuration, encompassing user permissions, environment variables (such as Java and Python paths), the working directory, and the command to launch the FastAPI application. This centralized management permitted the system to oversee the backend reliably and consistently with minimal overhead.

Beyond process management, real-time monitoring and diagnostics were activated using the built-in logging system of Linux. The journalctl utility was utilized to view and follow live logs from the summarization service. This facilitated easy monitoring of request activity, error checking, and validation of the service's correct functioning at all times.

The configuration of the summarization backend as a persistent systemd service is illustrated in Figure 63, which shows the summarize.service file containing environment setup, working directory, and execution parameters required to run the FastAPI application seamlessly on reboot or failure.

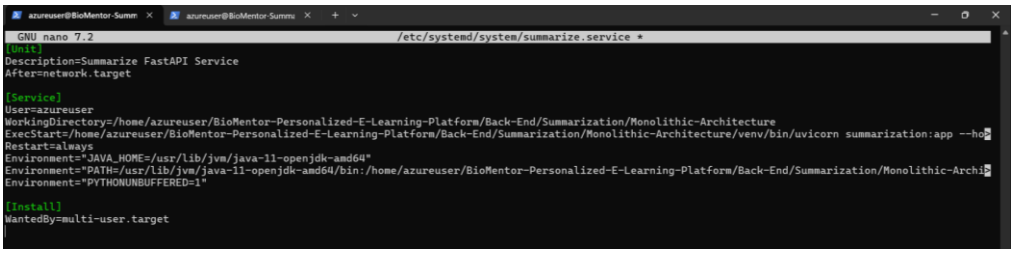


Figure 63: Systemd configuration for running the BioMentor summarization service on Azure VM

This integration at the service level guarantees that the BioMentor summarization API remains active, responsive, and self-healing essential traits for an educational platform that students depend on for studying and revising A/L Biology subjects.

6. To improve the accessibility and production readiness of the deployed summarization service, Nginx was set up as a reverse proxy on the Azure VM.

This setup allows for the API to be reached through the server's public IP address (http://20.193.248.25) without needing to specify the internal port 8002, which simplifies client requests and frontend integration.

The reverse proxy operates by directing all incoming HTTP requests on port 80 to the backend FastAPI service that is hosted on localhost:8002. To support this, an inbound port rule for port 80 was also added in the Azure Network Security Group (NSG), enabling external HTTP access via Nginx This abstraction enhances the API endpoint for users by simplifying it while also improving timeout management, request forwarding, and the ability to scale for future upgrades.

Moreover, Nginx contributes to:

- Enhancing security by concealing backend port visibility.
- Consolidating traffic management for various APIs or services within a single domain/IP.
- Managing upload restrictions and connection timeouts, particularly for substantial document uploads

A personalized configuration was developed at /etc/nginx/sites-available/summarization-api-gateway and connected to the sites-enabled directory. After verifying and rebooting Nginx, the summarization API was made accessible to the public through standard HTTP.

Figure 64 depicts the Nginx setup employed to reverse proxy requests to the backend application, guaranteeing user ease and adherence to deployment best practices.

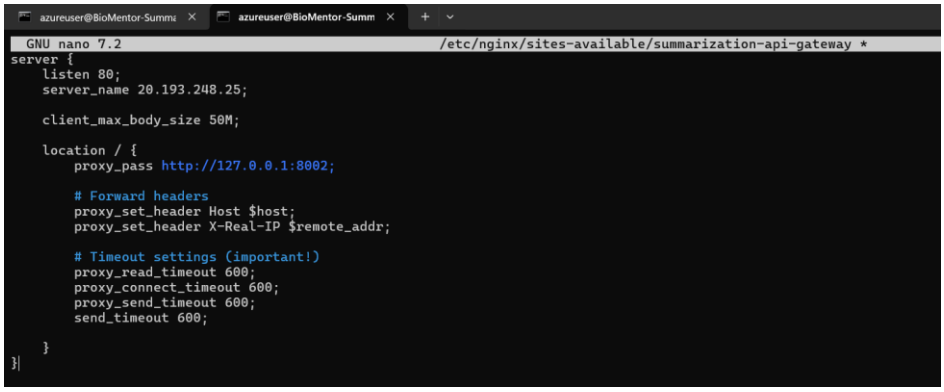


Figure 64: Nginx Reverse Proxy Configuration

The BioMentor summarization feature is currently deployed securely and is operational on a Microsoft Azure Virtual Machine, utilizing a fine-tuned Flan-T5 model that is hosted on Hugging Face for smooth integration. This deployment structure provides a modular design by distinguishing the model hosting from the backend runtime, which ensures both ease of maintenance and scalability. The system achieves high availability and resilience through ongoing service management, while secure public access is upheld by Azure Network Security Group (NSG) regulations. With dedicated computing resources designed for performance, the backend can manage real-time document summarization, topic-focused queries, and structured note creation. This infrastructure allows A/L Biology students to engage with a dependable, accessible, and curriculum-aligned educational resource aimed at improving their learning experience.

Maintenance:

The maintenance phase of the BioMentor summarization component is a vital continual process that commences following successful deployment. It guarantees that the system stays stable, secure, and responsive to student requirements, especially in facilitating real-time summarization, note creation, and topic-based query management for A/L Biology material.

1. Bug Fixes and Issue Monitoring

A key aspect of maintenance involves actively overseeing the backend service to identify bugs or runtime problems. The Azure VM utilizes logging tools that monitor service activity through system-level logs, enabling real-time detection of failures or unusual behavior. These logs are consistently examined using Linux system utilities to pinpoint and resolve issues swiftly. Any problems that occur such as failed summarization attempts, timeouts, or formatting mistakes are scrutinized, debugged, and fixed through controlled

patch updates. This approach ensures minimal disruption to the user experience.

2. Performance and Uptime Monitoring

The summarization service functions as a persistent system service, which allows for automatic restarts and resilience against unexpected shutdowns. The system's health is regularly assessed, focusing on memory usage, CPU load, and response times, to guarantee peak performance. This is especially crucial during peak student usage times, such as during exam study periods, when multiple requests are handled.

3. Feature Improvements Based on Feedback

Maintenance also encompasses the collection of feedback from students utilizing the platform. Input concerning the quality of summaries, improvements in file download processes, and language formatting is routinely reviewed and transformed into feature upgrades. For instance, previous feedback about enhancing topic keyword identification led to improvements in the query parsing logic. Planned enhancements are tested internally before being deployed to the live system, ensuring that changes improve the experience without causing regressions.

4. Model and Dependency Updates

Since the foundational Flan-T5 model is hosted on Hugging Face, model version management and updates can be handled independently of the backend deployment. The backend undergoes regular assessments to identify outdated Python libraries or deprecated APIs, with scheduling for dependency updates to sustain compatibility and security. This also involves verifying updated versions of any third-party tools utilized for PDF creation or voice synthesis.

5. Security and Backup Checks

To maintain data integrity and security of access, routine examinations are conducted on the virtual machine's firewall configurations, SSH access policies, and resource utilization. Configuration snapshots and backups are preserved to facilitate rapid recovery in case of system failure or loss of configuration.

6. CI/CD Pipeline for Automated Deployment

To streamline the deployment process and ensure the system is always up to date with the latest improvements, a Continuous Integration and Continuous Deployment (CI/CD) pipeline was implemented using GitHub Actions. This pipeline is triggered whenever changes are pushed to the main branch within the summarization backend directory. Upon execution, it securely connects to the Azure VM, pulls the latest changes, reinstalls dependencies, and restarts the systemd service. This automated deployment ensures that new updates are applied efficiently without manual intervention, reducing downtime and simplifying version control. It also supports faster development cycles, enabling smoother feature rollouts and bug fixes in future iterations.

2.2 Commercialization

The BioMentor platform, designed as a curriculum-compliant educational tool for A/L Biology, is ready for commercial implementation. By incorporating a well-tuned large language model, organized note creation, topic-specific summarization, support for multiple languages, and on-demand accessibility, BioMentor is set to contribute to the digital advancement of education in Sri Lanka and potentially in other regions. Below is a commercialization plan tailored to its existing functions and target audience.

1. **User Segmentation**
    - Students (A/L Biology stream): Learners looking for immediate summaries, notes, and conceptual understanding through intelligent query responses.
    - Teachers and Tutors: Users who utilize the platform to enhance content delivery, provide revision support, and create organized academic resources.
    - Educational Institutions: Schools and tutoring centers that intend to incorporate AI-powered study aids into their biology programs.

2. **Subscription-Based Pricing Model**

    BioMentor can implement a Software-as-a-Service (SaaS) approach with role-specific access:
    - Free Tier: Limited daily access to topic, document summaries and note creation.
    - Student Premium Plan: Unlimited access to summarization, multilingual translation, structured notes, and downloadable audio/PDF formats.
    - Teacher Plan: Tools for bulk document summarization, custom topic development, and monitoring performance.
    - Institutional Plan: Comprehensive platform access for student groups, centralized administrative controls, and integration with LMS.

3. **Authentication & Access Management**
    - Secure access through email/password or Google OAuth login.
    - Role-based access control (RBAC) to customize feature availability based on user roles (student, teacher, admin).

4. **Granular Permission Sets**
    - Students: Allowed to submit topics, upload documents, download summaries/audio files, and translate notes.
    - Teachers: Enabled to create custom summaries for lessons, oversee group usage, and suggest content.
    - Admins: Responsible for managing user registrations, analyzing usage data, and overseeing subscription settings.

5. **Subscription & Payment Integration**
    - Integration with payment platforms like Stripe or PayPal for secure transactions.
    - Support for monthly or annual plans, including trial periods, billing history, and options for upgrading or downgrading.

6. **Advertising Revenue (Free Tier Only)**
    - Non-intrusive advertisements displayed solely to free-tier users, ideally sourced from academic publishers, EdTech products, or educational book providers.
    - All advertisements would adhere to data privacy guidelines and remain relevant to student interests.

7. **Institutional Partnerships**
    - Implement BioMentor in schools and tutoring centers through strategic onboarding collaborations.
    - Provide white-labeled versions for centers wanting customized learning platforms.

- Include training manuals and onboarding resources for teachers to facilitate easy integration.

8. Marketing & Outreach

- Market through school networks, online A/L discussion groups, social media platforms, and academic events.
- Highlight features such as curriculum-aligned summarization, note generation, audio support, and translation to build credibility and attract interest.
- Share testimonials from students and case studies to demonstrate practical effectiveness.

9. Feedback & Product Evolution

- Incorporate in-app feedback mechanisms for students and teachers to submit suggestions or report issues.
- Utilize feedback to foster enhancements and steer future development such as expanding to A/L Chemistry and Physics or adding Sinhala/Tamil audio options.

3. RESULTS & DISCUSSION

The development and deployment of the BioMentor summarization feature represented an important advancement in improving learning assistance for Advanced Level (A/L) Biology courses. This feature was rigorously assessed across multiple aspects, such as architectural effectiveness, computational efficiency, and the quality of summarization. The findings from these assessments highlight the system's reliability, significance to the educational setting, and feasibility for application in actual learning environments.

1. Architectural Assessment: Monolithic versus Microservices

To evaluate the deployment strategy and behavior of the BioMentor summarization component, two architectural models were implemented and analyzed: monolithic and microservices. Both architectures were containerized and deployed using Docker to ensure consistency, portability, and isolation. These were assessed based on several important criteria, including response times, resource utilization, deployment efficiency, debugging challenges, and fault tolerance.

The monolithic model combines all system modules such as summarization, note generation, translation, and audio synthesis into a single application, which provides simplicity and speed. On the other hand, the microservices architecture breaks these functionalities into distinct, independently deployable services, offering modularity and scalability.

The comparative performance and trade-offs of both architectures are summarized below in Table 10:

Table 10: Comparison of Monolithic and Microservices Architecture

Feature	Monolithic Architecture	Microservices Architecture
Response Time	85% faster, no API overhead. (Figure 62)	Slower, inter-service delays. (Figure 63)
Deployment Speed	47% faster (37.8 min). (Figure 64)	Slower (71.5 min). (Figure 65)
CPU and Memory Usage	Lower (~34% CPU, 28-36% RAM). (Figure 66)	Higher (`43-62% CPU, 37-40% RAM). (Figure 67)
Debugging	Easier, centralized logs. (Figure 68)	Harder, distributed logs. (Figure 69)
Infrastructure	Simple, single container.	Complex, multiple containers.
Fault Tolerance	Lower, single failure risk.	Higher, independent services.

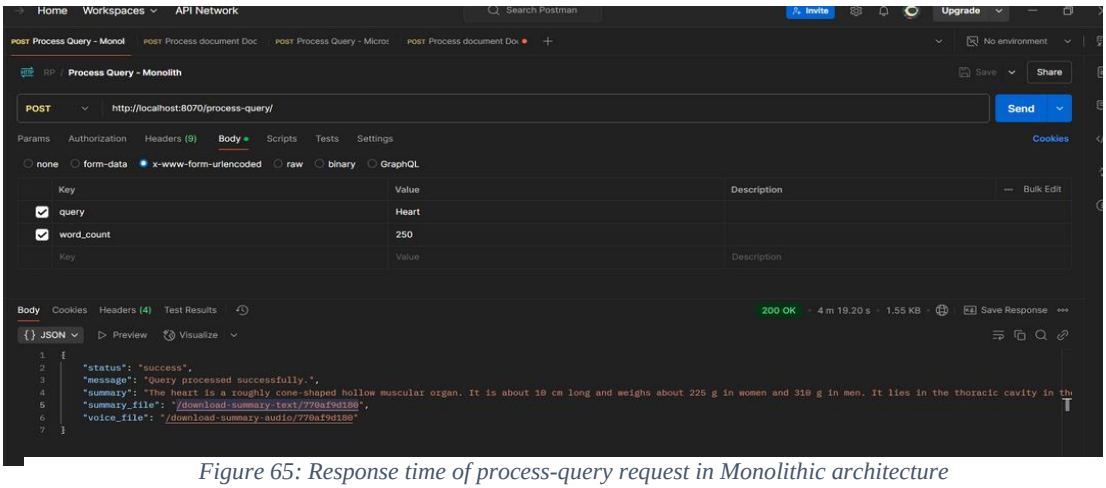


Figure 65: Response time of process-query request in Monolithic architecture

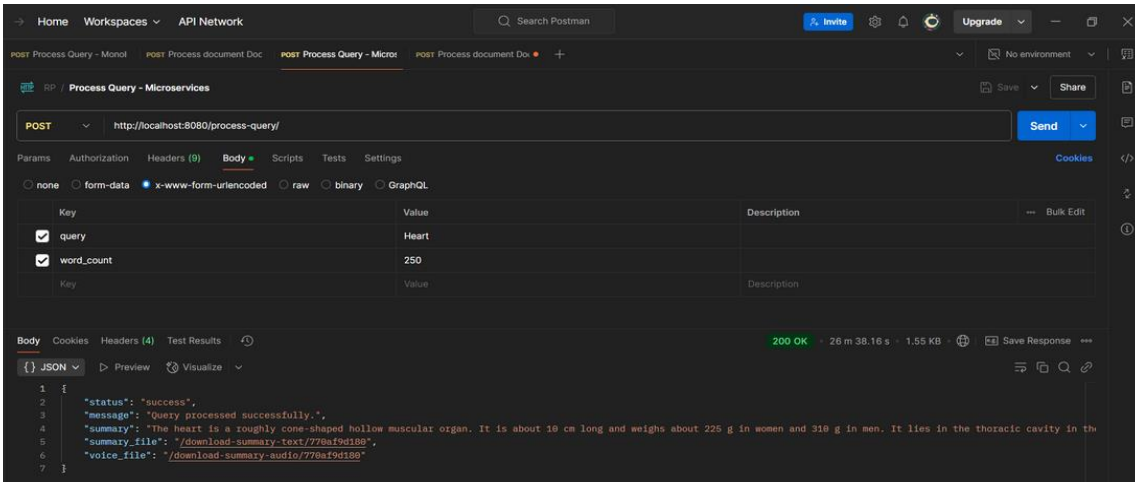


Figure 66: Response time of process-query request in Microservices architecture

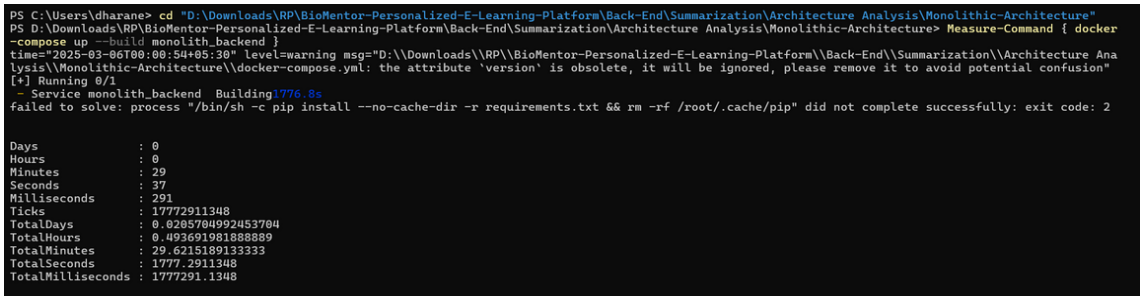


Figure 67: Deployment speed of Monolithic Architecture

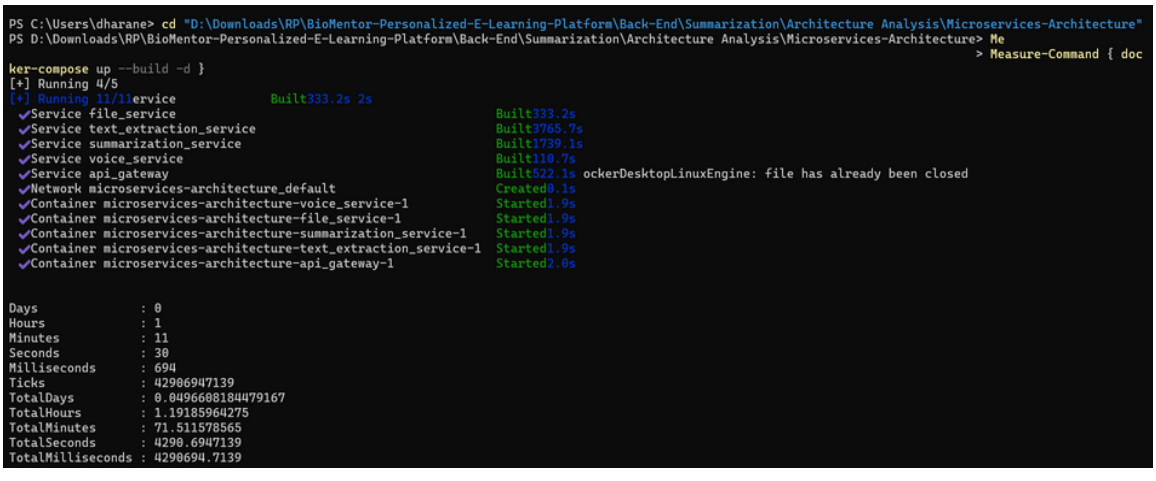


Figure 68: Deployment speed of Microservices Architecture

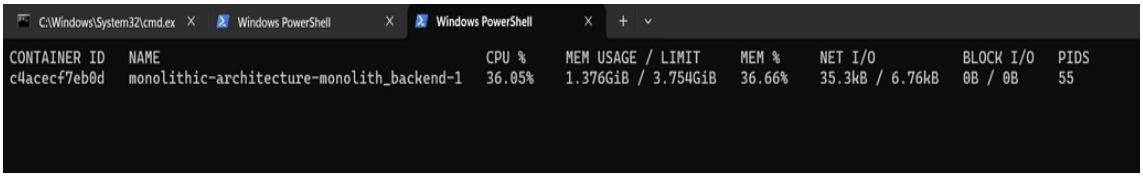


Figure 69: CPU & Memory usage of Monolithic Architecture

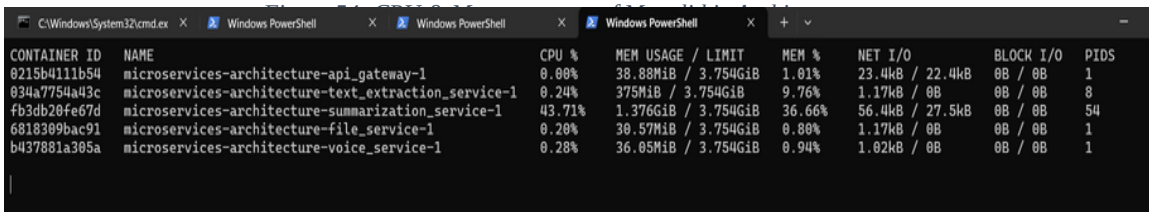


Figure 70: CPU & Memory usage of Microservices Architecture

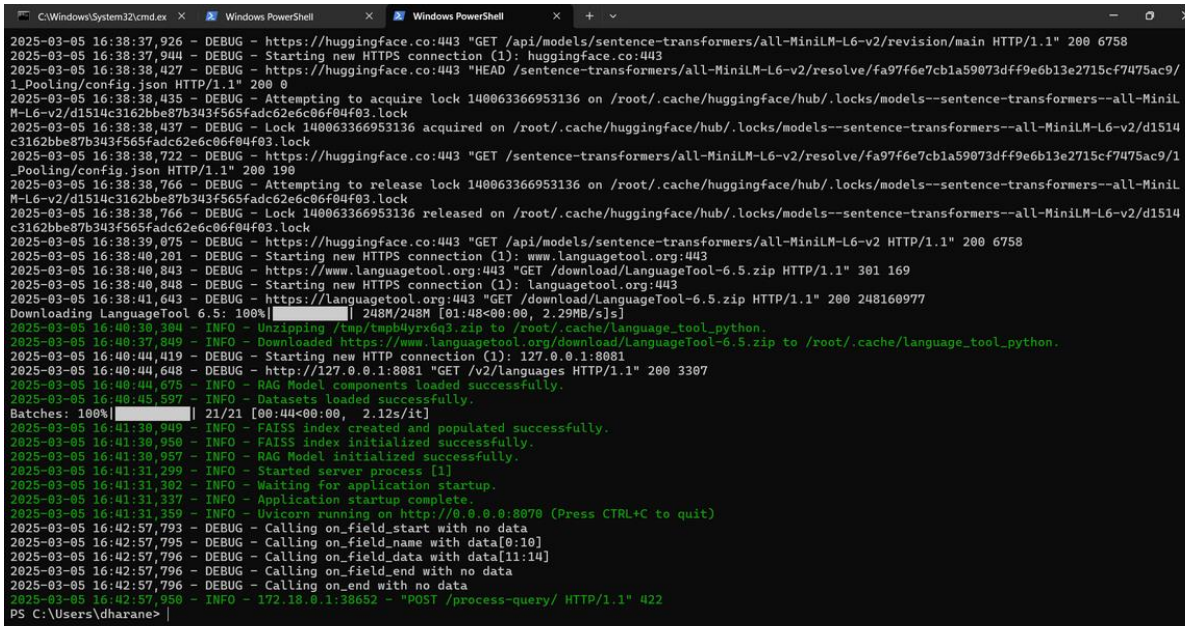


Figure 71: Centralized logs of Monolithic Architecture

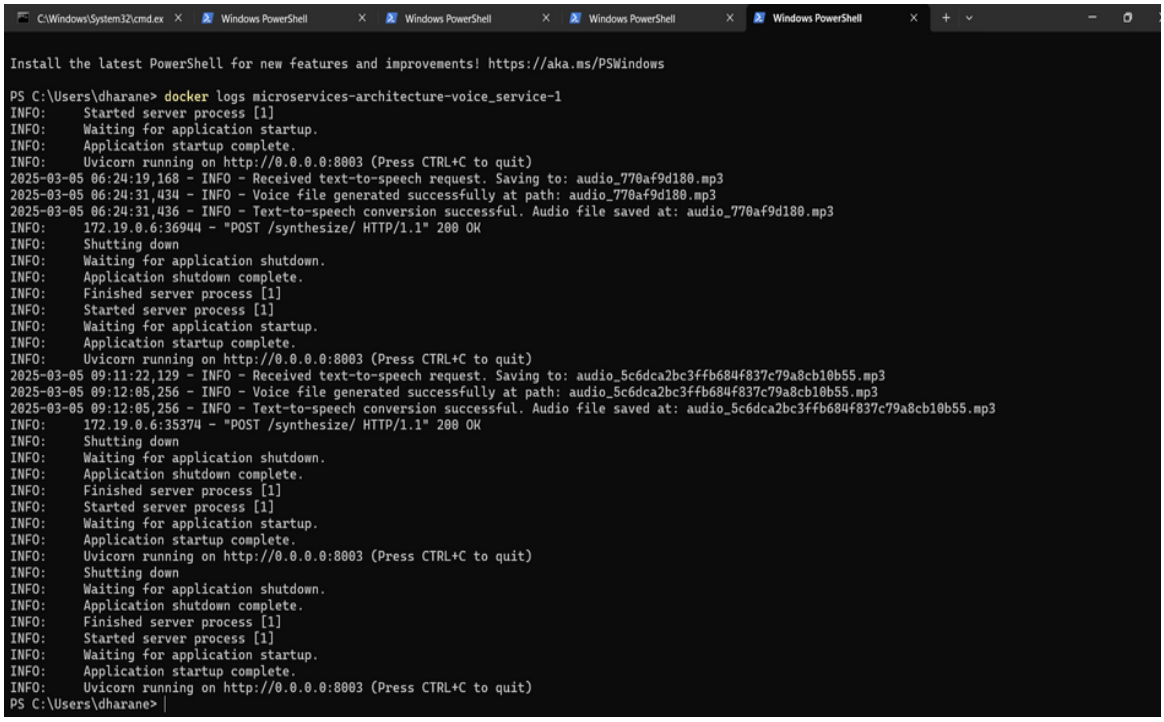


Figure 72: Distributed logs of Microservices Architecture

2. Evaluation of Summarization Quality Using ROUGE Metrics

To assess the quality, precision, and relevance of the summaries produced by the fine-tuned Flan-T5-Base model, the component was evaluated using ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metrics. ROUGE is a commonly used framework for evaluating summaries, emphasizing semantic similarity over exact word correspondence, which makes it especially appropriate for educational materials that may include paraphrasing or rewording.

ROUGE evaluates summaries based on different levels of textual overlap:

- **ROUGE-1** evaluates unigram (single word) matches, assessing how well the model captures key terms.
- **ROUGE-2** focuses on bigram (two-word phrase) continuity, indicating fluency and phrase-level cohesion.

- **ROUGE-L** measures the Longest Common Subsequence, reflecting overall sentence structure and logical flow.

The ROUGE results for the BioMentor summarization component are presented in Table 11 below:

Table 11: Rouge Score Evaluation

Metric	Definition	Score	Interpretation
ROUGE-1 (Unigram Overlap)	Overlap of individual words (unigrams)	0.74	High keyword retention; core concepts captured effectively
ROUGE-2 (Bigram Overlap)	Overlap of consecutive two-word phrases (bigrams)	0.40	Maintains phrase continuity and preserves relationships between terms
ROUGE-L (Longest Common Subsequence)	Longest common subsequence match (structural alignment)	0.54	Indicates good sentence structure and readable, logically arranged summaries

3. Practical Deployment Outcome and Educational Relevance

After development and evaluation, the BioMentor summarization system was evaluated using a varied collection of biology-related educational resources, such as textbook materials, uploaded PDF files, and user-generated queries. The model effectively produced summaries in real-time and offered outputs in multiple formats, including downloadable PDF summaries and audio files (in MP3 format).

Importantly, the system's multilingual capabilities facilitated the translation of notes into Tamil and Sinhala, broadening access for a wider array of students throughout Sri Lanka. The clarity and responsiveness of the summaries along with the generation of structured notes enabled students to review extensive amounts of material more

efficiently. The support for topic-specific queries allowed for targeted summarization, while the document upload feature enabled comprehensive summarization for customized study resources.

The implementation of both monolithic and microservices versions provided flexibility in application: smaller installations benefited from speed and simplicity, whereas larger institutional deployments could take advantage of the scalability and separation offered by microservices.

The findings from both architectural performance assessments and evaluations of summarization quality confirm the technical strength and educational significance of the BioMentor summarization feature. Monolithic architecture is ideal for quick deployments and targeted usage, whereas the microservices approach sets the groundwork for future expansion and integration with larger systems. The ROUGE scores demonstrate that the model produces summaries that are informative, coherent, and in line with academic expectations. Acting as a digital learning assistant, BioMentor successfully connects AI with the A/L Biology curriculum, offering personalized academic support aligned with the curriculum for students with varying learning requirements.

4. FUTURE SCOPE

The effective development and implementation of the BioMentor summarization feature for A/L Biology represents a solid foundation for further advancements in intelligent educational support. As the platform develops, multiple promising avenues can be explored to enhance its impact, efficiency, and versatility in various academic settings.

1. Broader Subject Coverage

Although the present version of BioMentor centers on A/L Biology, the underlying architecture and summarization process can be adapted for other key subjects, such as A/L Chemistry, Physics, and General English. This will enable students from diverse fields to receive high-quality summarization and academic assistance.

2. Customizable Summarization

Future versions could incorporate customizable summarization, where the language complexity, or format of the summary adjusts in real-time to suit the user's grade level, learning preferences, or previous performance. This would ensure that the content remains relevant and suitably challenging for each student.

3. Offline Access

Providing offline functionality will allow students in rural or low-connectivity regions to access summaries and notes without depending on a constant internet connection. It could further improve accessibility and user experience.

4. Enhanced Translation and Multilingual Features

While current support includes translations into Tamil and Sinhala, future enhancements could focus on grammar corrections, transliteration, and context-sensitive translation improvements, particularly for scientific terminology. This would further tailor the learning experience for students who are non-native English speakers.

5. Integration with Learning Management Systems (LMS)

BioMentor could be connected with widely used LMS platforms in schools and tutoring centers to automatically summarize lesson plans, class notes, or uploaded content, simplifying teacher tasks and increasing lesson engagement.

6. User Data Analytics and Tailored Feedback

The integration of user analytics will enable the platform to provide feedback loops, such as monitoring common topics summarized, pinpointing areas of student misunderstanding, or suggesting study schedules based on user activity.

7. Commitment to Data Privacy and Ethical Standards

As BioMentor manages content generated by students, future upgrades should strengthen data privacy measures, including end-to-end encryption and clarity regarding how student information is processed and stored.

8. Shared Note Collaboration

Students could collaborate on collective summaries or notes, especially during group study sessions or class-wide revision activities. This collaborative element could foster peer learning within the BioMentor environment.

9. Cross-Platform API Integration

Expanding BioMentor as a plug-and-play summarization API could allow other educational applications, digital libraries, or content repositories to incorporate its core functionalities, enabling a wider academic impact across different platforms.

5. CONCLUSION

The introduction of BioMentor’ s summarization feature represents a major step forward in providing curriculum-aligned, accessible, and tailored academic assistance for A/L Biology learners. Aimed at breaking down complex material and encouraging self-directed learning, the feature effectively incorporates essential elements such as document summarization, topic-based summarization, audio synthesis, structured notetaking, and multilingual support all within an easy-to-navigate, student-oriented platform.

Featuring a dual architectural setup, monolithic for ease of use and performance, and microservices for enhanced scalability and reliability the feature underwent assessment regarding system performance, response times, and deployment efficiency. These architectural evaluations not only improved the backend’ s flexibility but also laid the groundwork for future scaling advancements.

Using a fine-tuned Flan-T5 model, BioMentor’ s summarization system proved to be highly efficient, as confirmed by impressive ROUGE scores across unigram, bigram, and structural indicators. This validated the system's ability to generate brief, coherent, and informative summaries that are appropriate for academic learning and review. The addition of multilingual features (Tamil and Sinhala) for notes and audio outputs for summaries further ensured that the system caters to a variety of learners, enhancing inclusivity and access.

Operated on an Azure Virtual Machine with the model hosted through the Hugging Face Hub, the summarization component has demonstrated technical robustness, educational significance, and commercial viability.

In summary, BioMentor’ s summarization feature is a dependable and significant solution in the dynamic field of digital education. It illustrates how well-crafted intelligent content delivery can improve comprehension, support personalized learning experiences, and contribute to a more inclusive and effective educational environment.

REFERENCES

[1] A. Venkataramana et al., "Abstractive Text Summarization Using BART," *IEEE MysuruCon*, 2022.

[2] A. Jabbar et al., "Abstractive Text Summarization: A Transformer-Based Approach," *IEEE Access*, vol. 11, 2023.

[3] D. Sudharson et al., "An Abstractive Summarization and Conversation Bot Using T5 and Its Variants," *Proc. ICAICCIT*, 2023.

[4] K. Maurya et al., "NLP-Enhanced Long Document Summarization: A Comprehensive Approach for Information Condensation," *InCACCT*, 2024.

[5] M. Ramina et al., "Topic Level Summary Generation Using BERT Induced Abstractive Summarization Model," *ICICCS*, IEEE, 2020.

[6] R. Ramachandran et al., "A Novel Method for Text Summarization and Clustering of Documents," *GCAT*, IEEE, 2022.

[7] A. Goyal et al., "TalkifyPy: The Pythonic Voice Assistant," *Proc. ACET*, 2024.

[8] J. Christian et al., "Analyzing Microservices and Monolithic Systems: Key Factors in Architecture, Development, and Operations," *Proc. IC2IE*, 2023.

[9] H. Thapliyal, "Unveiling the Past: AI-Powered Historical Book Question Answering," *HBQA Thesis*, dasarpai.com, 2023. [Online]. Available: https://dasarpai.com/assets/images/projects/hbqas/HBQA_Thesis-18-Nov-23.pdf

[10] J. Xu, "GenAI and LLM for Financial Institutions: A Corporate Strategic Survey," *SSRN*, 2024. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4988118

APPENDICES