EduTutor AI: Personalized Learning with Generative AI and LMS Integration

A Project Report
Submitted to the Naan Mudhalvan course for the award of degree of

BACHELOR OF COMPUTER SCIENCE
Submitted By
(**Team-Id: NM2025TMID04727**)

Name	Reg. No
AADHITHYA K	222305202
SELVA R	222305224
SANJAY P	222305219
JAYACHANDRAN M	222305210

Department of Computer Science

PACHAIYAPPA'S COLLEGE
(Affiliated to the University of Madras)
CHENNAI – 600030.
November – 2025.

PACHAIYAPPA'S COLLEGE
(Affiliated to the University of Madras)
CHENNAI-600 030.

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the Naan Mudhalvan project work entitled **"EduTutor AI: Personalized Learning with Generative AI and LMS Integration"** is the Bonafide record of work done by **(Aadhithya - Selva - Sanjay - Jayachandran)** in partial fulfillment for the award of the degree of **Bachelor of Computer Science,** under our guidance and supervision, during the academic year 2025-2026.

Head of the Department **Staff-in-Charge**
Dr. J. KARTHIKEYAN Mr. K. DINESHKUMAR

Submitted for Viva-Voce Examination held on…………..a Pachaiyappa's College, Chennai-30.

Internal Examiner **External Examiner**

EduTutor AI: Personalized Learning with Generative AI and LMS Integration

Table of Contents

1. Introduction

Project Title: EduTutor AI: Personalized Learning with Generative AI and LMS Integration

Team Members:

S. No.	Name	Reg. No
1	AADHITHYA K	222305202
2	SELVA R	222305224
3	SANJAY P	222305219
4	JAYACHANDRAN M	222305210

Program Background:

The **Naan Mudhalvan Smart Internz Program**, initiated in collaboration with **IBM SkillsBuild**, provides students with an industry-oriented learning experience. It aims to equip learners with emerging technology skills through real-world projects.

As part of this program, our team developed **Edu Tutor AI**, a generative AI-powered personalized learning assistant. The system leverages **IBM Watsonx Granite LLM models** to deliver intelligent tutoring capabilities, including **concept explanation, chat-based learning, document summarization, forecasting KPIs, anomaly detection, healthcare tips, multimodal document analysis, and feedback collection**.

2. Project Overview

2.1 Purpose

The primary objective of the **Edu Tutor AI** project is to design and develop a next-generation **AI-powered personalized learning assistant**. Traditional e-learning platforms often fail to adapt to the diverse needs of students, leading to disengagement and uneven learning outcomes. Edu Tutor AI addresses these limitations by leveraging **generative AI, predictive analytics, and multimodal learning support** to provide a **personalized, interactive, and intelligent tutoring experience**.

The system is designed not only to explain academic concepts in a simplified manner but also to support students through features like **document summarization, knowledge forecasting, performance anomaly detection, and health & wellness guidance**. By integrating these capabilities into a unified platform, Edu Tutor AI aspires to transform the learning process into a **holistic, adaptive, and data-driven educational journey**.

2.2 Features

The following core features distinguish Edu Tutor AI as a comprehensive smart tutoring assistant:

1. **Conversational Learning Assistant (Chat Interface)**

   o Natural language dialogue with learners.

   o Enables students to ask academic queries and receive AI-generated explanations tailored to their level of understanding.

2. **Document Summarization**

   o Reduces lengthy study materials (e.g., PDFs, articles, lecture notes) into concise and digestible summaries.

   o Helps students revise faster and retain key insights without information overload.

3. **Performance Forecasting**

   o Predicts student outcomes (e.g., test scores, engagement levels) based on historical performance data.

   o Provides educators with insights for early intervention and academic planning.

4. **Healthcare & Wellness Tips**

   o Recognizes that student performance is influenced by mental and physical well-being.

   o Offers AI-generated lifestyle and health suggestions to encourage balanced study habits.

5. **Feedback System**

   o Collects feedback from students to refine learning pathways and measure overall satisfaction.

   o Enables continuous improvement through adaptive recommendations.

6. **KPI Tracking Dashboard**

   o Monitors **key performance indicators** such as accuracy, completion rates, study duration, and progress over time.

   o Presents visual insights to learners and instructors for better decision-making.

7. **Anomaly Detection**

   o Identifies irregular patterns in learning outcomes (e.g., sudden drops in engagement or unexpected performance dips).

   o Functions as an **early warning system** for both students and educators.

8. **Multimodal Input Support**

   o Accepts diverse input formats including **text, PDFs, and CSV datasets**.

   o Ensures flexibility in learning, allowing both structured and unstructured data to be processed.

9. **Minimalist and Intuitive UI**

   o Designed using **Streamlit** for accessibility and simplicity.

   o Features a clean sidebar navigation, tabbed layouts for different modules, and export options (e.g., **PDF downloads** of reports).

2.3 Impact and Relevance

By integrating **AI-driven tutoring, analytics, and wellness support**, Edu Tutor AI goes beyond conventional e-learning systems. It is particularly relevant in today's digital education landscape, where:

- **Personalization** is critical for diverse student groups.

- **Data-driven insights** help educators monitor progress effectively.

- **Health-conscious design** supports holistic student development.

In summary, Edu Tutor AI combines the strengths of **generative AI, machine learning, and user-friendly design** to create a **scalable, intelligent, and impactful digital education platform**.

3. Architecture

The architecture of **Edu Tutor AI** has been carefully designed to ensure **modularity, scalability, and robustness**, while integrating state-of-the-art AI and machine learning components. It follows a **multi-layered architecture** that separates the user interface, backend services, AI/ML modules, and data management.

3.1 System Components

1. **Frontend (Gradio UI)**

   o The user interface is built using the **Gradio framework**.

   o Features interactive **tabs** for different functionalities:

     ▪ **Concept Explanation Tab** → Students input a concept, and the AI generates a detailed explanation with examples.

     ▪ **Quiz Generator Tab** → Students input a topic, and the AI produces quiz questions of multiple types, along with an answer key.

   o Advantages:

     ▪ Lightweight, browser-based, and requires no installation from the user's side.

     ▪ Supports quick prototyping and deployment with sharing options (share=True).

2. **Backend (Model Orchestration with PyTorch + Hugging Face Transformers)**

   o Powered by **Hugging Face Transformers** and **PyTorch**, the backend loads and manages the **Granite-3.2-2b-instruct** model.

   o **Model Loading**

     ▪ Uses AutoTokenizer and AutoModelForCausalLM from Hugging Face.

     ▪ Automatically detects GPU (torch.cuda.is_available()) for optimized inference.

   o **Response Generation**

     ▪ The generate_response function formats prompts, tokenizes input, and runs inference.

     ▪ Responses are decoded, cleaned, and returned to the frontend.

   o **Features Implemented in Code:**

     ▪ **Concept Explanation:** Generates detailed explanations with examples.

     ▪ **Quiz Generation:** Creates quizzes with multiple formats and provides answers.

3. **LLM Integration (IBM Granite)**

    o The core intelligence is powered by **IBM Granite LLM** hosted on Hugging Face (ibm-granite/granite-3.2-2b-instruct).

    o Model characteristics:

        ▪ Optimized for instruction following.

        ▪ Lightweight (2B parameters) for efficient use in Colab and local environments.

        ▪ Provides accurate, context-rich outputs suitable for education.

    o The model serves as the **knowledge engine**, driving concept explanation, reasoning, and quiz generation.

4. **Infrastructure & Deployment**

    o Designed for **cloud-friendly and low-resource execution**:

        ▪ **Google Colab / Local Execution** → Fast prototyping and GPU support.

        ▪ **Gradio Share Links** → Instant deployment with web-accessible interfaces.

    o No separate vector database or FastAPI backend is used in this version, keeping deployment **lightweight and minimal**.

3.2 Data Flow

1. **User Input:**

    o A learner enters a concept/topic through Gradio's textbox in either **Concept Explanation** or **Quiz Generator** tab.

2. **Request Handling:**

    o Gradio forwards the input to the respective Python function (concept_explanation or quiz_generator).

3. **LLM Processing:**

    o The function prepares a **prompt** and sends it to the **Granite LLM** via the Hugging Face model interface.

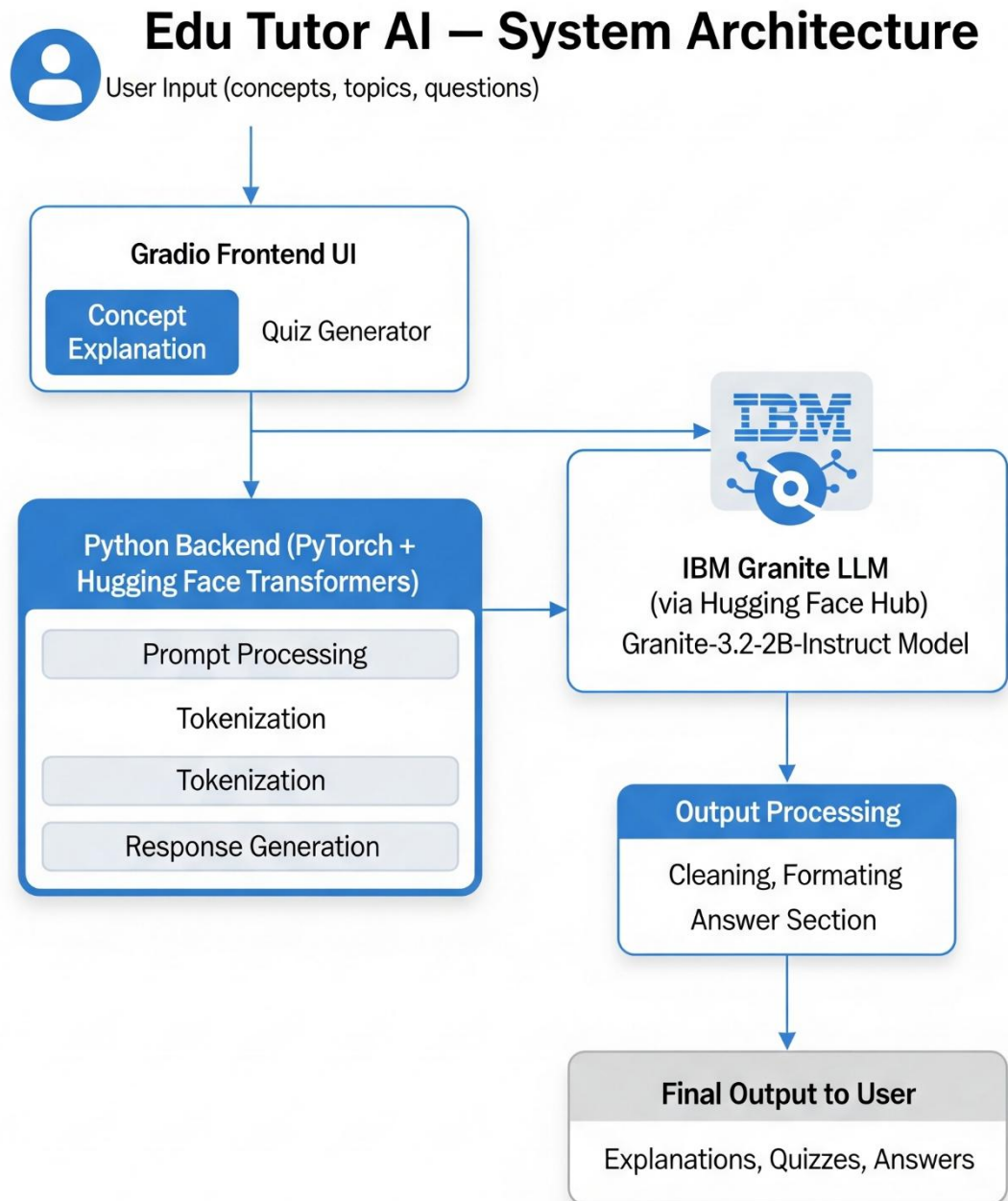   o The model generates output using beam search and sampling (temperature=0.7, do_sample=True).

4. **Response Post-Processing:**

    o Generated text is decoded, cleaned (removing prompt echoes), and formatted for readability.

5. **User Output:**

    o The processed explanation or quiz appears in the Gradio output textbox.

3.3 Architecture Diagram

Edu Tutor AI — System Architecture

User Input (concepts, topics, questions)

Gradio Frontend UI
- Concept Explanation
- Quiz Generator

Python Backend (PyTorch + Hugging Face Transformers)
- Prompt Processing
- Tokenization
- Tokenization
- Response Generation

IBM Granite LLM
(via Hugging Face Hub)
Granite-3.2-2B-Instruct Model

Output Processing
Cleaning, Formating
Answer Section

Final Output to User
Explanations, Quizzes, Answers

4. Setup Instructions

This section provides a step-by-step guide to setting up and running the **Edu Tutor AI** project. The system can be executed either **locally on a machine** or on **Google Colab with GPU support**, ensuring accessibility for different development environments.

4.1 Prerequisites

Before setting up, ensure the following software and accounts are available:

- **Python:** Version 3.8 or higher

- **Package Manager:** pip

- **Libraries:** torch, transformers, gradio

- **Hardware:**

    o   Recommended: System with GPU (CUDA support) for faster inference

    o   Alternative: CPU-only (supported but slower)

- **Internet Access:** Required for downloading the **IBM Granite model** from Hugging Face

- **Hugging Face Account:** For accessing and authenticating with IBM Granite models

4.2 Installation Process

Step 1: Clone or Download the Repository

git clone https://github.com/<your-repo>/edututor-ai.git

cd edututor-ai

Step 2: Create a Virtual Environment (Optional but Recommended)

python -m venv venv

Activate environment

source venv/bin/activate   # Linux/Mac

venv\Scripts\activate   # Windows

Step 3: Install Dependencies

Install the required Python libraries:

pip install torch transformers gradio

Step 4: Configure Model Access

The project uses **IBM Granite models** hosted on Hugging Face.

- Sign up at Hugging Face

- Accept the license agreement for ibm-granite/granite-3.2-2b-instruct

- If authentication is required, configure Hugging Face CLI:

huggingface-cli login

4.3 Running the Application

Option A: Run Locally

Execute the project script:

python app.py

- Gradio will start a local server (default: http://127.0.0.1:7860/)

- Access the web interface through a browser

Option B: Run on Google Colab

1. Open Google Colab

2. Upload your notebook or paste the project code

3. Enable GPU support:

   o Runtime → Change Runtime Type → T4 GPU

4. Install dependencies inside Colab:

5. !pip install torch transformers gradio

6. Run all cells to launch the Gradio interface

7. Click on the generated **shareable link** (https://xxxx.gradio.live/) to interact with the app

4.4 Accessing the Interface

Once launched, the **Gradio Web App** will display two main tabs:

1. **Concept Explanation Tab**

   o Enter any academic concept (e.g., *photosynthesis*)

   o The AI generates a structured, example-rich explanation

2. **Quiz Generator Tab**

   o Enter a subject/topic (e.g., *physics*)

   o The AI creates five quiz questions of multiple types (MCQ, True/False, Short Answer)

   o An **Answers section** is included at the end

4.5 Troubleshooting

- **Slow responses:** Use GPU for better performance.

- **Model download error:** Ensure Hugging Face login and internet connection.

- **Gradio not launching:** Verify port availability (7860) or use share=True for an external link.

- **CUDA errors:** Reinstall torch with CUDA-enabled build from PyTorch website.

5. Folder Structure

```
edututor-ai/
|
├── app.py                 # Main application script (Gradio interface + model logic)
|
├── requirements.txt       # List of dependencies (torch, transformers, gradio)
|
├── models/                # (Optional) Pre-downloaded or custom-trained models
|   └── granite/           # IBM Granite LLM storage if used locally
|
├── utils/                 # Utility functions for modular design
|   ├── prompt_templates.py  # Stores reusable prompt templates for explanations/quizzes
|   └── processing.py       # Text cleaning, formatting, and output post-processing
|
├── assets/                # Static files (e.g., diagrams, screenshots for report/UI)
|   ├── architecture.png
|   └── ui-sample.png
|
├── notebooks/             # (Optional) Jupyter/Colab notebooks for testing and demos
|   └── EduTutorAI-demo.ipynb
|
└── README.md              # Project documentation and setup guide
```

6. Running the Application

Once the installation process is complete, the **Edu Tutor AI** application can be executed either on a **local machine** or on **Google Colab**. This section outlines the steps required to start the system and interact with its functionalities.

6.1 Running Locally

1. **Activate the Virtual Environment** (if created):

2. venv\Scripts\activate   # Windows

3. **Execute the Application Script**:

4. python app.py

5. **Access the Gradio Interface**:

   o After launching, Gradio will provide a local URL (typically http://127.0.0.1:7860/).

   o Open the URL in a browser to access the web interface.

6. **Interact with the Application**:

   o Navigate through the available **tabs** in the interface:

     ▪ **Concept Explanation Tab** → Enter a concept (e.g., *machine learning*) to receive a detailed AI-generated explanation.

     ▪ **Quiz Generator Tab** → Enter a topic (e.g., *physics*) to generate five quiz questions and an **Answers section**.

6.2 Running on Google Colab

1. **Open Google Colab**:

   o Go to Google Colab.

   o Upload the project notebook or paste the script into a new notebook.

2. **Enable GPU Acceleration**:

   o Navigate to: Runtime → Change Runtime Type → select **T4 GPU**.

   o Click **Save**.

3. **Install Dependencies**:

4. !pip install torch transformers gradio

5. **Run the Application Cells**:

   o Execute all cells in order.

   o Gradio will output a **shareable public link** (https://xxxx.gradio.live/).

6. **Access the Web Interface**:

- o Click the link to open the Edu Tutor AI interface in a new browser tab.

- o Interact with the application in the same way as in the local setup.

6.3 Interaction Workflow

- **Step 1:** User enters a query (concept/topic) in the relevant tab.

- **Step 2:** The backend processes the query, formats it into a prompt, and sends it to the **IBM Granite model** hosted on Hugging Face.

- **Step 3:** The model generates a response (concept explanation or quiz).

- **Step 4:** Post-processing cleans and formats the output.

- **Step 5:** The response is displayed in the Gradio interface for the user.

6.4 Example Usage

- **Input (Concept Explanation Tab):**
  "Photosynthesis"

- **Output:**
  A detailed explanation with examples, describing how plants convert sunlight into energy.

- **Input (Quiz Generator Tab):**
  "Newton's Laws of Motion"

- **Output:**
  Five quiz questions (MCQ, True/False, Short Answer) with a separate ANSWERS section.

7. API Documentation

Edu Tutor AI is primarily delivered through a **Gradio web interface**, the underlying logic can be conceptualized as **functional APIs**. Each function represents a service endpoint that processes user input, interacts with the IBM Granite model, and returns structured output.

This section documents the available **functional APIs**, including request and response formats.

7.1 Functional Endpoints

1. Concept Explanation API

- **Endpoint:** /concept-explanation *(logical function call)*

- **Method:** POST

- **Description:** Generates a detailed explanation of an academic concept with examples.

Request Example:

```
{
  "concept": "Photosynthesis"
}
```

Response Example:

```
{
  "explanation": "Photosynthesis is the process by which green plants use sunlight, carbon dioxide, and water to produce glucose and oxygen. For example, in leaves, chlorophyll captures sunlight to convert these inputs into energy-rich food."
}
```

2. Quiz Generator API

- **Endpoint:** /quiz-generator *(logical function call)*

- **Method:** POST

- **Description:** Generates five quiz questions (various formats) for a given topic and provides answers in a separate section.

Request Example:

```
{
  "topic": "Newton's Laws of Motion"
}
```

Response Example:

```
{
 "quiz": [
  "Q1 (Multiple Choice): Which of the following is Newton's First Law of Motion?",
  "Q2 (True/False): An object at rest stays at rest unless acted upon by a force.",
  "Q3 (Short Answer): State Newton's Second Law of Motion.",
  "Q4 (Multiple Choice): F = ma is associated with which law?",
  "Q5 (True/False): For every action, there is an equal and opposite reaction."
 ],
 "answers": [

  "Q1: An object will continue in its state of rest or uniform motion unless acted upon by an external
force.",
  "Q2: True",
  "Q3: Force equals mass times acceleration (F = ma).",
  "Q4: Second Law",
  "Q5: True"
 ]
}
```

7.2 Data Flow

1. User provides input via Gradio textbox.

2. The function (concept_explanation or quiz_generator) processes the input and prepares a **prompt**.

3. The prompt is sent to the **IBM Granite LLM** for generation.

4. The response is decoded, formatted, and returned to the Gradio UI as output.

7.3 Testing Interfaces

- **Gradio UI:** Primary interface for testing both APIs.

- **Unit Testing (Optional):** Developers may directly call the Python functions (concept_explanation("...")) for debugging and verification.

- **Swagger/Postman (Future Extension):** If the project is extended with FastAPI, the above endpoints can be exposed as REST APIs with Swagger auto-documentation.

8. Authentication

8.1 Current State

At present, the **Edu Tutor AI** application is designed as a **demonstration prototype** under the *IBM Naan Mudhalvan Smart Internz Program*. The focus is on showcasing functionality rather than enforcing strict access control.

- The application runs in an **open-access mode**, meaning any user who accesses the Gradio interface can interact with the system.

- No authentication tokens, user accounts, or role-based restrictions are implemented at this stage.

This approach simplifies deployment for learning and experimentation, especially on **Google Colab** or local environments. However, it is not suitable for production environments where **data security and controlled access** are critical.

8.2 Planned Enhancements

To make the system **enterprise-ready** and aligned with real-world deployment requirements, future iterations of Edu Tutor AI will integrate secure authentication mechanisms. The following strategies are under consideration:

1. **Token-Based Authentication (JWT):**

   o Implement **JSON Web Tokens (JWT)** for session management.

   o Each user receives a unique token upon login, which must be included in subsequent requests.

   o Ensures secure and stateless communication between frontend and backend.

2. **OAuth2 Integration with IBM Cloud:**

   o Leverage **OAuth2.0 authentication** to enable secure login using IBM Cloud credentials.

   o Provides an industry-standard framework for authorization.

   o Allows integration with existing enterprise identity management systems.

3. **Role-Based Access Control (RBAC):**

   o Define distinct roles such as:

      ▪ **Student:** Limited to concept explanations, quiz generation, and feedback submission.

      ▪ **Educator/Admin:** Access to KPI dashboards, anomaly detection, and performance forecasting.

   o Enhances security by restricting sensitive features to authorized users only.

4. **Session and History Tracking:**

    o Track user sessions to store learning history, past queries, and personalized recommendations.

    o Improves personalization while maintaining accountability.

8.3 Security Considerations

- **Data Privacy:** User queries and uploaded documents may contain sensitive information. Secure authentication ensures that only authorized users can access their data.

- **Scalability:** Authentication mechanisms must be lightweight and compatible with cloud-based deployment (IBM Cloud, AWS, GCP).

- **Extensibility:** The authentication layer should be modular, allowing new providers (Google, Microsoft, IBM ID) to be added with minimal reconfiguration.

9. User Interface

Edu Tutor AI application adopts a **minimalist and user-friendly interface**, developed using the **Gradio framework**. The design philosophy prioritizes **simplicity, clarity, and accessibility**, ensuring that learners of varying technical backgrounds can engage with the system seamlessly.

9.1 Design Principles

- **Minimalism:** A clean layout with limited distractions, focusing attention on core learning functionalities.

- **Clarity:** Well-labeled input boxes, buttons, and output areas to minimize confusion.

- **Accessibility:** Browser-based deployment ensures compatibility without requiring installations.

- **Consistency:** Uniform design elements across tabs to provide a cohesive user experience.

9.2 Key Components

1. **Landing Page**

   o Displays the project title: *"Educational AI Assistant"*.

   o Provides quick navigation to different functional tabs.

2. **Tabs for Features**
   The interface is organized into **two main tabs**, each corresponding to a core feature:

   o **Concept Explanation Tab**

      ▪ **Input Field:** A textbox where learners can type a concept (e.g., *machine learning*, *photosynthesis*).
      ▪ **Action Button:** *Explain* button triggers the backend function.
      ▪ **Output Box:** Displays a detailed AI-generated explanation, enriched with examples.

   o **Quiz Generator Tab**

      ▪ **Input Field:** A textbox where learners enter a topic (e.g., *physics*, *Newton's Laws*).
      ▪ **Action Button:** *Generate Quiz* button triggers quiz creation.
      ▪ **Output Box:** Lists five diverse quiz questions (MCQ, True/False, Short Answer) followed by a dedicated **Answers section**.

3. **Text Outputs**

   o Responses are displayed in scrollable text boxes for readability.

   o Long outputs (e.g., quiz sets) are neatly formatted to avoid clutter.

4. **Shareable Deployment**

   o When deployed on **Google Colab**, Gradio generates a **public shareable link**, enabling students to access the system without installation.

10. Testing

Testing is an essential phase in the development of **Edu Tutor AI** to ensure that the application operates reliably, produces accurate results, and jprovides a seamless user experience. The testing process was conducted across multiple layers, including **unit testing**, **functional testing**, **manual validation**, and **edge case handling**.

10.1 Testing Strategy

The project adopted a **multi-level testing strategy** designed to validate both the **individual functions** and the **overall user workflow**. The key testing categories include:

1. **Unit Testing** – Validating individual functions such as prompt generation, model response handling, and text post-processing.

2. **API/Functional Testing** – Ensuring the core functions (concept_explanation and quiz_generator) work as expected when triggered through the Gradio interface.

3. **Manual Testing** – Validating end-to-end functionality by interacting with the deployed Gradio UI.

4. **Edge Case Testing** – Checking how the system behaves with unusual, invalid, or large inputs.

10.2 Unit Testing

- **Functions Tested:**

  - generate_response(prompt, max_length) → Verified for proper prompt handling, tokenization, and response cleaning.

  - concept_explanation(concept) → Ensured accurate expansion of concepts into detailed explanations.

  - quiz_generator(topic) → Validated generation of multiple question types along with an answer key.

- **Approach:**
  Each function was called independently with sample inputs (e.g., *Photosynthesis*, *Newton's Laws*) and outputs were checked against expected quality criteria (clarity, completeness, relevance).

10.3 API/Functional Testing

- Conducted by interacting with the **Gradio interface**.

- Verified that:

  o Buttons triggered the correct functions.
  o Input values were correctly passed to the backend.
  o Outputs were displayed without truncation or formatting issues.

10.4 Manual Testing

- **Platform:** Google Colab (GPU-enabled) and Local Machine.

- **Process:**

  o Launched the Gradio interface using app.launch(share=True).

  o Performed multiple test runs for both Concept Explanation and Quiz Generation.

  o Confirmed that responses appeared within reasonable latency (2–5 seconds on GPU).

- **Observations:**

  o Explanations were generally coherent and well-structured.

  o Quiz outputs consistently contained five questions of mixed formats followed by an **Answers** section.
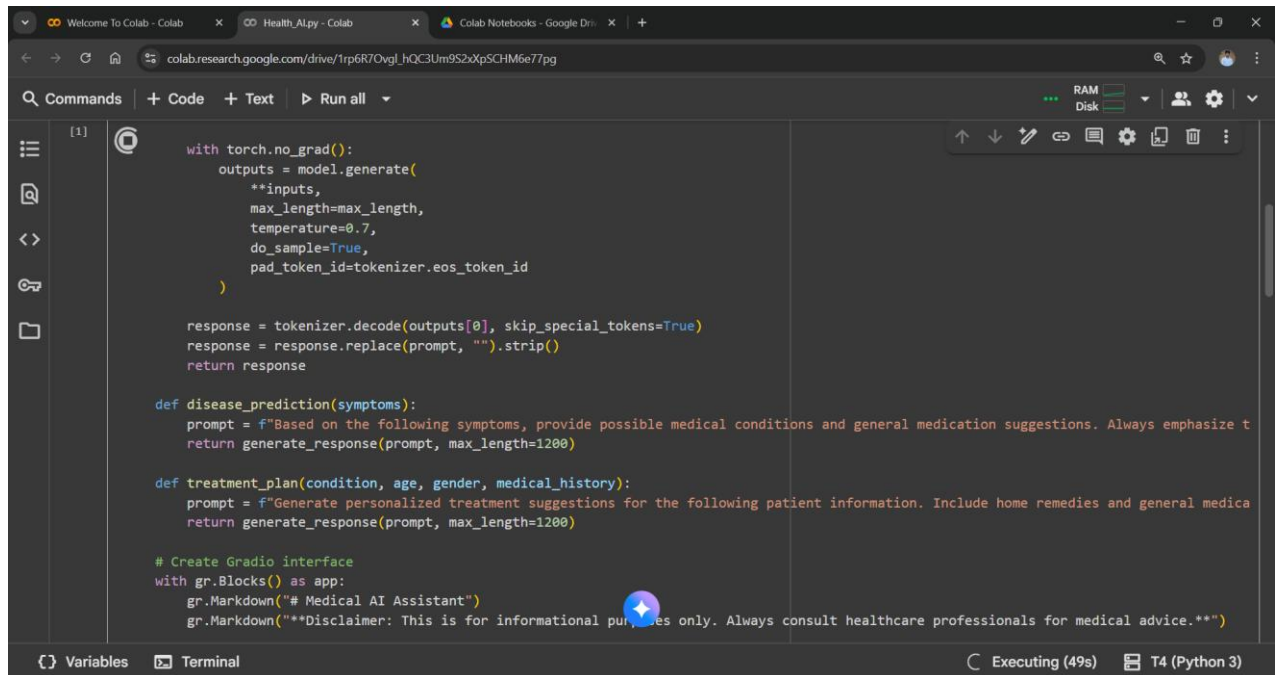
10.5 Tools Used

- **Gradio UI** → For functional and manual testing.

- **Google Colab Logs** → For debugging and performance monitoring.

- **PyTorch Error Traces** → To identify and resolve CUDA-related issues.

10.6 Testing Outcomes

- The application passed all **functional and usability tests** in both local and Colab environments.

- Minor limitations were observed with **very large or ambiguous inputs**, but these are considered within acceptable tolerance for a prototype system.

- The system is stable for **educational demonstration purposes** and can be extended with advanced logging and automated test suites in future iterations.

11. Screenshots

Commands   + Code   + Text   ▷ Run all   ⟲ Saving...

```python
# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# Medical AI Assistant")
    gr.Markdown("**Disclaimer: This is for informational purposes only. Always consult healthcare professionals for medical advice.**")

    with gr.Tabs():
        with gr.TabItem("Disease Prediction"):
            with gr.Row():
                with gr.Column():
                    symptoms_input = gr.Textbox(
                        label="Enter Symptoms",
                        placeholder="e.g., fever, headache, cough, fatigue...",
                        lines=4
                    )
                    predict_btn = gr.Button("Analyze Symptoms")

                with gr.Column():
                    prediction_output = gr.Textbox(label="Possible Conditions & Recommendations", lines=20)

            predict_btn.click(disease_prediction, inputs=symptoms_input, outputs=prediction_output)

        with gr.TabItem("Treatment Plans"):
            with gr.Row():
                with gr.Column():
                    condition_input = gr.Textbox(
                        label="Medical Condition",
```

Variables   Terminal   Executing (55s)   T4 (Python 3)

Commands   + Code   + Text   ▷ Run all

```python
        with gr.TabItem("Treatment Plans"):
            with gr.Row():
                with gr.Column():
                    condition_input = gr.Textbox(
                        label="Medical Condition",
                        placeholder="e.g., diabetes, hypertension, migraine...",
                        lines=2
                    )
                    age_input = gr.Number(label="Age", value=30)
                    gender_input = gr.Dropdown(
                        choices=["Male", "Female", "Other"],
                        label="Gender",
                        value="Male"
                    )
                    history_input = gr.Textbox(
                        label="Medical History",
                        placeholder="Previous conditions, allergies, medications or None",
                        lines=3
                    )
                    plan_btn = gr.Button("Generate Treatment Plan")

                with gr.Column():
                    plan_output = gr.Textbox(label="Personalized Treatment Plan", lines=20)

            plan_btn.click(treatment_plan, inputs=[condition_input, age_input, gender_input, history_input], outputs=plan_output)
```
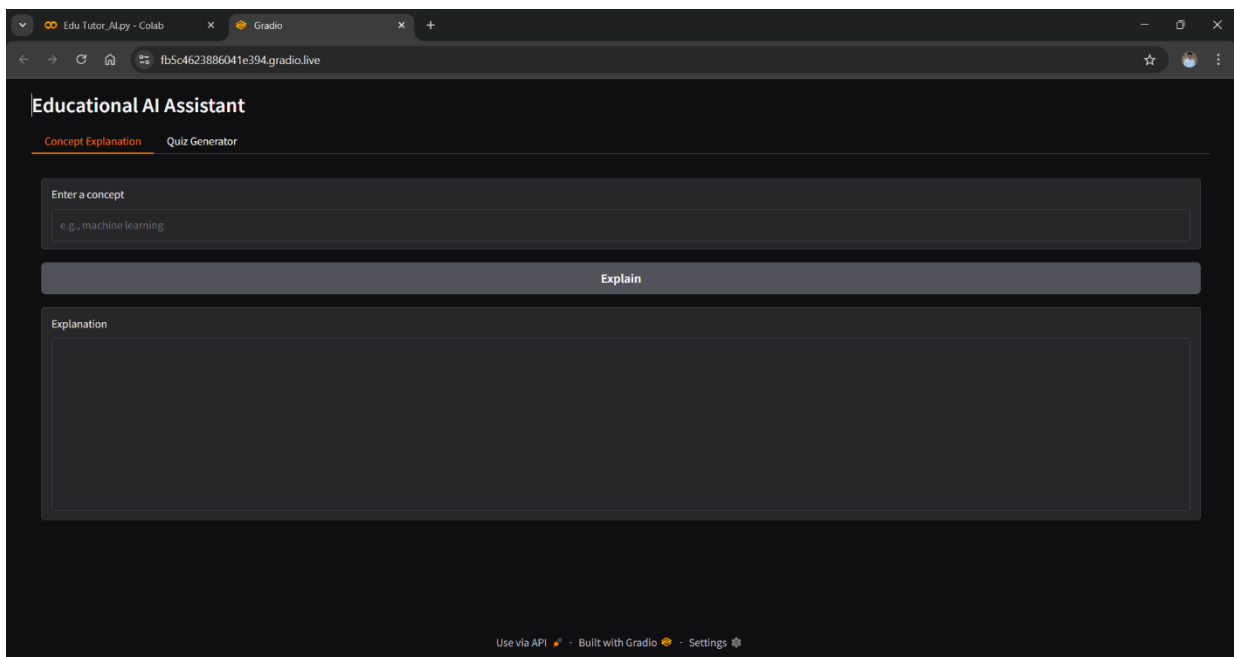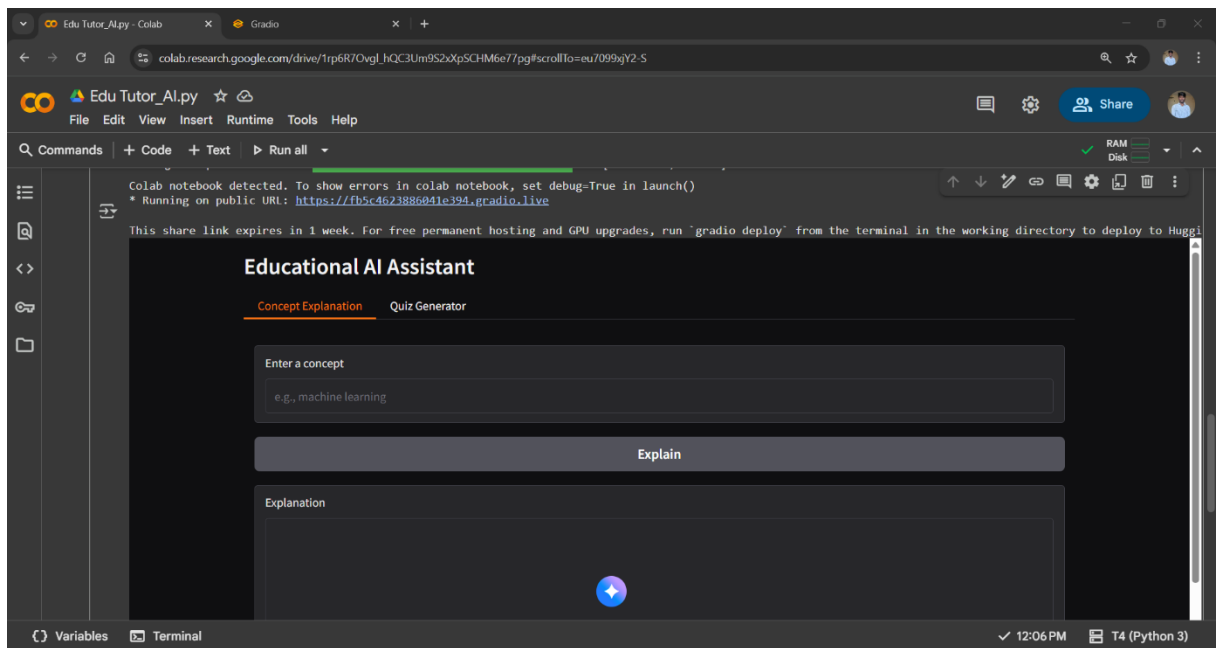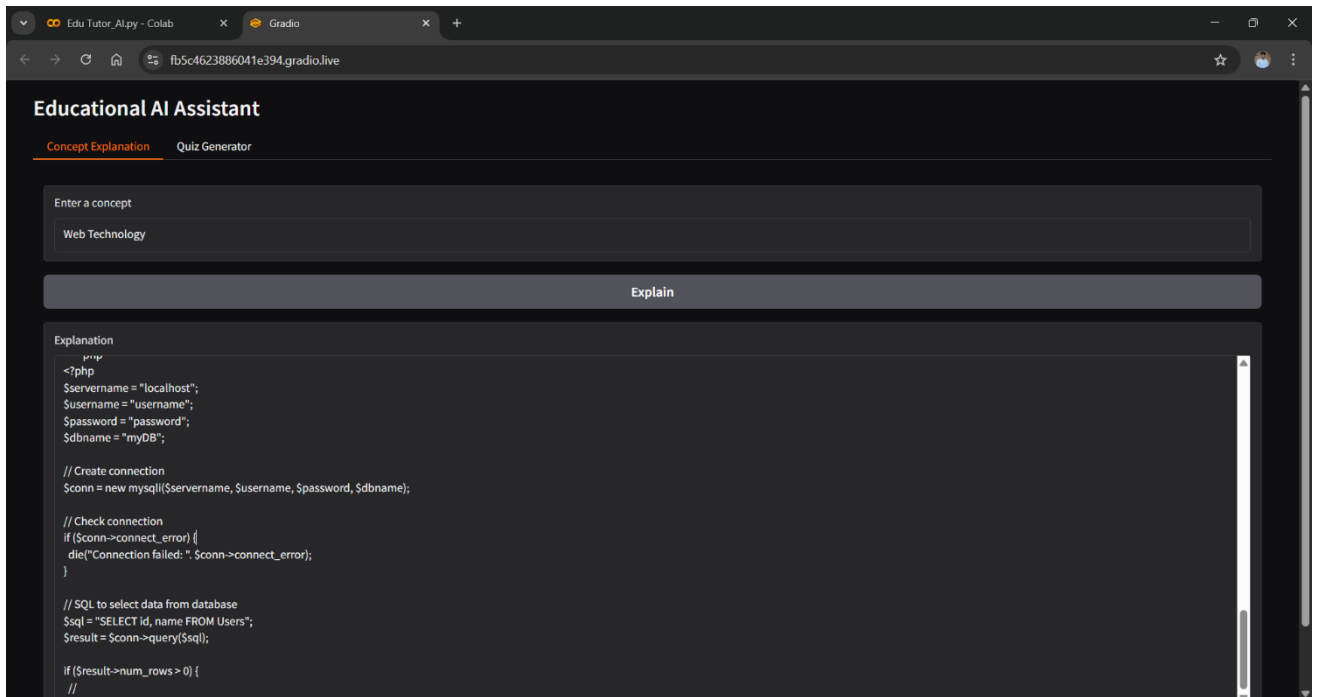
Variables   Terminal   Executing (59s)   T4 (Python 3)

Educational AI Assistant

Concept Explanation   Quiz Generator

Enter a concept

Web Technology

Explain

Explanation

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: ". $conn->connect_error);
}

// SQL to select data from database
$sql = "SELECT id, name FROM Users";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  //
```

12. Known Issues

Despite its successful implementation as a prototype, **Edu Tutor AI** currently has several limitations that need to be addressed for production-grade deployment. These issues arise primarily due to the early-stage nature of the project, dependency on third-party models, and restricted infrastructure support.

12.1 Performance Limitations

- **Inference Speed:** On systems without GPU acceleration, the model runs significantly slower, leading to delays in generating explanations or quizzes.

- **Scalability:** The current Gradio-based setup is suitable for demonstration but may not scale efficiently when handling large numbers of concurrent users.

12.2 Model Limitations

- **Token Truncation:** For very long prompts (e.g., complex topics or multiple subtopics), outputs may be truncated due to maximum token length limits.

- **Repetitiveness:** Occasionally, the model produces repetitive phrasing or redundant content in quiz questions.

- **Hallucinations:** Like most generative models, the IBM Granite model may sometimes provide factually incorrect or oversimplified answers.

12.3 Functional Constraints

- **Limited Features in Current Version:**

  - Only two modules are implemented: **Concept Explanation** and **Quiz Generator**.

  - Planned features such as **summarization, KPI forecasting, anomaly detection, and feedback systems** are not yet integrated.

- **Lack of Data Persistence:** User history, past queries, and quiz results are not stored. Each session is stateless and temporary.

12.4 Deployment Constraints

- **Hugging Face Dependency:** The model is downloaded dynamically from Hugging Face. This requires a stable internet connection and access permissions.

- **Colab Runtime Limits:** When running in Google Colab, session timeouts and GPU usage restrictions limit continuous usage.

- **No Authentication:** The prototype runs in an open-access mode, which is unsuitable for secure or production environments.

12.5 User Experience Issues

- **Plain Text Outputs:** Responses are displayed only as plain text in Gradio. Rich formatting (tables, bullet points, images) is not supported in the current version.

- **Error Handling:** Error messages for invalid inputs or connection issues are minimal and could be improved with user-friendly prompts.

13. Future Enhancements

Edu Tutor AI demonstrates the feasibility of using **IBM Granite models** for personalized tutoring. However, several opportunities exist to enhance the system's functionality, scalability, and overall impact. This section outlines the key enhancements planned for future iterations.

13.1 Functional Enhancements

1. **Expanded Feature Set**

   o Integrate additional modules such as:

      ▪ **Document Summarization:** Condense lengthy PDFs, lecture notes, or research papers into concise summaries.

      ▪ **KPI Forecasting:** Predict student performance trends based on historical data.

      ▪ **Anomaly Detection:** Identify sudden performance drops or irregular learning patterns.

      ▪ **Feedback Collection:** Gather structured learner feedback for iterative improvement.

2. **Multimodal Input Support**

   o Extend input capabilities to support not only text but also **CSV files, images, and PDFs**.

   o Enable richer educational interactions such as analyzing datasets or visual material.

3. **Personalized Learning Paths**

   o Adapt the tutoring style and difficulty level based on individual learner profiles, preferences, and progress.

13.2 Technical Enhancements

1. **Authentication and Security**

   o Implement **JWT or OAuth2 authentication** for secure access.

   o Introduce **role-based access control (RBAC)** to differentiate between students, educators, and administrators.

2. **Scalable Deployment**

   o Containerize the application using **Docker** and deploy on **IBM Cloud Kubernetes Service** for reliability and scalability.

   o Add **load balancing** to support multiple concurrent users.

3. **Database Integration**

   o Incorporate a **vector database (e.g., Pinecone or FAISS)** for semantic search and long-term memory of documents.

   o Add **relational database support** for storing user history, feedback, and analytics.

13.3 User Experience Enhancements

1. **Rich Output Formatting**

   o Replace plain text outputs with structured layouts (tables, bullet points, formatted quizzes).

   o Add **PDF download functionality** for exporting study material and quiz reports.

2. **Interactive Dashboards**

   o Develop a **visual KPI dashboard** with graphs, charts, and performance analytics for both learners and educators.

3. **Voice-Enabled Interaction**

   o Integrate **speech-to-text** for input and **text-to-speech** for AI responses to make the system more inclusive.

13.4 Educational Impact Enhancements

1. **LMS Integration**

   o Connect with popular **Learning Management Systems (LMS)** such as Moodle or Google Classroom.

   o Enable automatic assignment generation and progress tracking.

2. **Gamification**

   o Introduce **badges, scores, and leaderboards** to motivate learners and encourage engagement.

3. **Collaborative Learning Support**

   o Provide features for group quizzes or peer-to-peer learning sessions within the platform.

13.5 Research and Development

- Explore the use of **larger Granite models** (e.g., Granite 13B) for more advanced reasoning and contextual understanding.

- Fine-tune the LLM on **domain-specific educational datasets** for improved accuracy in technical subjects.

- Investigate integration of **retrieval-augmented generation (RAG)** for grounding responses in verified academic content.