

EDUTUTOR AI

PROJECT DOCUMENTATION

INTRODUCTION

EDUTUTOR AI

TEAM MEMBER 1: K.P.THEIVANAI

TEAM MEMBER 2: THAMIZHINY S

TEAM MEMBER 3: K.VAISHNAVI

TEAM MEMBER 4: THANUJA V



PROJECT OVERVIEW.

Education isn't stuck in classrooms or books anymore. AI changes that. Students get personalized learning whenever they want, wherever they are. This project, an Educational AI Assistant using the IBM Granite Model, aims to make things interactive and engaging. It keeps it simple too.

The app works like a virtual tutor. It explains tough concepts in plain language. It even whips up quizzes so students can check what they know. We built it with IBM's Granite 3.2-2B Instruct Model. That combines AI and natural language processing for a smooth experience, kind of human-like you know.

Students can ask for detailed explanations on any topic. They generate practice quizzes in different formats. They view instant answers right away for self-assessment.

FEATURES.

1. CONCEPT EXPLANATION.

Input is just text, like "machine learning" or "photosynthesis". Output comes as a detailed natural language breakdown. This works great for students wanting quick summaries. Teachers use it for lesson prep too.

2. QUIZ GENERATION.

This lets users create quiz questions on the fly for any topic. The AI makes five questions in various styles. You get multiple choice ones. True or false. Short answers. At the end, there's an answers section for checking or grading easily.

Input is a topic, say "Newton's Laws" or "global warming". Output is the list of five questions plus answers. Teachers love it for test making. Students do self-checks with it.

3. AI Model Integration.

Backend runs on IBM's Granite-3.2-2B-Instruct model. We pull it through the Hugging Face transformers library. It's tuned for tasks like explanations and generating content.

It handles text generation automatically. Supports GPU for faster speed. Switches to CPU if no GPU around.

4. WEB-BASED INTERFACE WITH GRADIO.

We made a simple interface using Gradio. You interact with the AI straight from your browser. No hassle. It has two tabs. One for concept explanation. One for quiz generator.

Text boxes for topics. Buttons to start each feature. Output spots to see results. You don't need to code at all to use it.

5. DEPLOYMENT IN GOOGLE COLAB.

The whole thing runs easy in Google Colab. No local setup needed. It installs dependencies quick and fires up the Gradio interface in minutes.

CONVERSATIONAL INTERFACE DESIGN FOR EDUTUTOR AI

2. Interface Architecture and Components

2.1 Input Section

This part is where it all starts. Users type in their needs here. It's kept simple. Flexible too. So people can get to what they want without hassle.

Topic/Concept Input Field:

It's a basic text box. One line. You put in the topic or concept you're curious about. Like, that feeds straight to the AI.

Example placeholder: "Enter a concept or topic (e.g., machine learning, photosynthesis)"

Generate Quiz: It spits out quiz questions tied to that topic.

Submit Button:

Big button. Label changes with the mode. Says “Explain” or “Generate Quiz”. Hit it. And off it goes to the AI backend.

2.2 Output Section

Response Display Area:

Scrollable area. Like a chat window or big text box. What shows depends on the mode you picked.

Quiz Generation Mode:

It makes five questions. Mix of types. Multiple choice. True/false. Short answer. Then an answer key follows. All formatted clean. So you can read and use it without squinting.

The area wraps text. Scrolls if needed. Highlights key bits. Keeps you engaged. Makes it readable.

Dynamic Elements:

Things change with the mode. Input placeholder shifts to match. Button label updates too. Guides you along.

Responsiveness:

Works on anything. Desktops. Tablets. Phones. That's key for learning on the go.

Accessibility:

Full keyboard support. Focus shows on buttons and fields. Follows guidelines. Screen readers work fine. For users who need it.

3. USER INTERACTION FLOW

The flow feels natural. Like a conversation. Cycles through steps smoothly.

3.1 Starting an Interaction

Open it in your browser. EduTutorAI loads up.

Type in a topic. Say, "Newton's Laws".

Pick the mode. Explain or quiz. Whatever fits.

3.2 Submitting a Request

Click submit. It bundles your input and mode. Sends to the AI.

Spinner or loader shows. Lets you know it's thinking.

3.3 Receiving and Displaying the Response

AI sends back the text.

Interface puts it in the output spot. Formatted nice. Paragraphs for explanations. Lists for quizzes.

3.4 Follow-up and Iteration

Change your input. Switch modes. No page refresh needed. Quick and easy.

Old outputs stick around. Or clear them. Up to you. Good for comparing. Or jotting notes.

4. TECHNICAL IMPLEMENTATION CONSIDERATIONS

4.1 Backend Integration

Talks to the IBM Granite-3.2-2B-Instruct model. Hosted on Hugging Face transformers.

Requests go async. UI doesn't freeze.

GPU if you got it. Speeds things up.

4.2 Frontend Framework

Built with Gradio. Easy components for inputs. Buttons. Outputs. Minimal setup.

Great for quick builds. Deploys in Google Colab. Makes it reachable for everyone.

POLICY SUMMARIZATION DOCUMENT.

1. Introduction.

This part gives a quick rundown on the policy. You know, its background stuff, why they made it in the first place, and the big picture reason it fits into the organization or whatever context we're talking about. Basically, it sets the stage without getting too deep.

2. Purpose.

Here, you explain why this policy even exists. It covers the goals, the objectives, and what outcomes they're aiming for. This really helps lay out why it's important and relevant, you know.

3. Scope.

This section spells out the limits of the policy. Like, who it covers, which areas or systems are in play, and any parts that get left out.

EXTENDED EDUTUTORIAL. PROPOSED FEATURES AND ARCHITECTURE.

- ❖ Resource Forecasting. This would predict resources needed. Like, how many quiz questions users might request. Or what topics are hot right now. Server usage too, that kind of thing.
- ❖ Eco-Tipo Generator. Eco-Tipo, I mean, probably stands for ecological type or eco-template. It generates environmentally themed educational content. Or templates. Stuff tailored for eco-types, say sustainability or environmental science.

- ❖ Anomaly Detection. This detects unusual patterns. For example, weird user behavior. Or content misuse. Unexpected input or output. Maybe spotting when AI output goes off topic. Or seems likely incorrect.
- ❖ Multimodal Input Support. Accepts not just text. But images, audio. Possibly video too. To enrich explanations or quizzes. Like, user uploads a diagram. Or asks a voice question.

Streamlit or Gradio UI. Builds a user interface with a web framework. Streamlit or Gradio. Supports all the above features. In a clean, interactive way.

2. Purpose of these extensions.

- ❖ These make the application more robust. And intelligent. Forecasting helps plan capacity. Like model runs or server load. Eco-Tipo content expands domains. Anomaly detection ensures quality control.
- ❖ They increase engagement. And flexibility. Multimodal input allows varied learning. Visuals, audio, etc. Suits more user styles. Eco-Tipo content adds variety. And relevance.

3. Feature specifications and integration.

3.1 Resource Forecasting.

- ❖ What it might do. Track usage metrics over time. Number of requests for concept explanations. Quizzes. Topics requested, etc.
- ❖ Use past data to forecast. For example, how many quiz generations tomorrow. Or next week. To plan server or memory needs.
- ❖ Optionally forecast topic demand. Like, identify topics likely in demand.
- ❖ Inputs and data needed. Logs of user requests. Timestamp, type like explanation or quiz. Topic. Maybe mode, multimodal or text-only.
- ❖ System metrics. CPU or GPU usage. Memory per request.
- ❖ Optional topic metadata. Categories, counts.
- ❖ Model or methods. Time series forecasting. ARIMA, Prophet, LSTM. Or simpler moving average.
- ❖ Integration with your code. Logging module captures user requests. And metadata.

- ❖ Separate component trains forecasting model periodically.
- ❖ Expose forecasts to UI. Maybe a dashboard page or tab. Showing upcoming forecasted load.

3.2 Eco-Tipo Generator.

- ❖ What it might do. Generate educational content. Explanations, quizzes, templates. Themed around ecology, sustainability, environmental science.
- ❖ Maybe create eco-friendly project templates. Or prompts.
- ❖ Inputs. Topic or theme. Ecology, climate change, recycling, etc.
- ❖ Possibly multimedia input. Like images of ecosystems. To generate relevant content.
- ❖ Integration. Alongside quiz or explanation generation. Include a mode called Eco-Tipo Generator.
- ❖ Optionally provide image input. User uploads photo. AI explains ecosystem. Or generates quiz about that ecosystem.

3.3 Anomaly Detection.

- ❖ Methods. For text input, natural language processing checks. Similarity to training topics. Profanity. Or irrelevant vocabulary.
- ❖ For response quality, use heuristics. Response length, coherence. Or another model judges quality.
- ❖ For usage metrics, thresholding on request rates.
- ❖ Integration. Before processing user request. Run input through anomaly detection. Flag or reject problematic inputs.
- ❖ After generating output, check quality. Optionally request regeneration if flagged.
- ❖ Log anomalies for monitoring.

4. REVISED ARCHITECTURE.

- ❖ User Interface. Gradio or Streamlit. Tabs or pages. Concept Explanation. Quiz Generation. Eco-Tipo Generator. Dashboard for Forecasting and Anomaly.
- ❖ Preprocessing or Input Handling. Tokenize text. Encode images or audio. Modes like eco-Tipo, anomaly check.
- ❖ Core AI Modules. Granite model for explanations or quiz. Plus prompt templates. Eco-Tipo content module. Anomaly detection module.
- ❖ Output or Response. Explanation, quiz, eco content. With multimodal input. Errors or warnings if anomaly detected. Dashboard visuals. Forecast charts, anomaly logs.

5. IMPLEMENTATION PLAN AND STEPS.

Here's a suggested step-by-step plan. To build this out.

Step 1. Logging and Data Collection. Modify existing code. To log usage. Requests, topics, input types, timestamps. And output metrics. Response times, sizes.

Step 2. Build Forecasting Module. Use historical logs. Experiment with time series model. Build simple UI to visualize forecasts.

Step 3. Add Multimodal Input. Choose image and audio encoders. Adapt preprocessing. Test with simple examples

Step 4. Eco-Tipo Mode. Create prompt templates for eco content. Integrate into explanation or quizzes. Maybe allow image-based eco scenarios.

Step 5. Anomaly Detection. Define anomalies to catch. Build anomaly detection model or heuristics. Integrate checks before or after generation.

Step 6. Design UI. Decide on Gradio or Streamlit. Build interface with tabs or pages. Include dashboards. Integrate multimodal upload components.

Step 7. Testing and Validation. Test each mode. Test edge cases. Ensure models behave well with multimodal inputs. Check anomaly detection errors and false positives.

Step 8. Deployment and Monitoring. Deploy on server or Colab or cloud. Monitor resource usage. Forecasts versus actual. Adjust forecasting model. Refine anomaly rules.

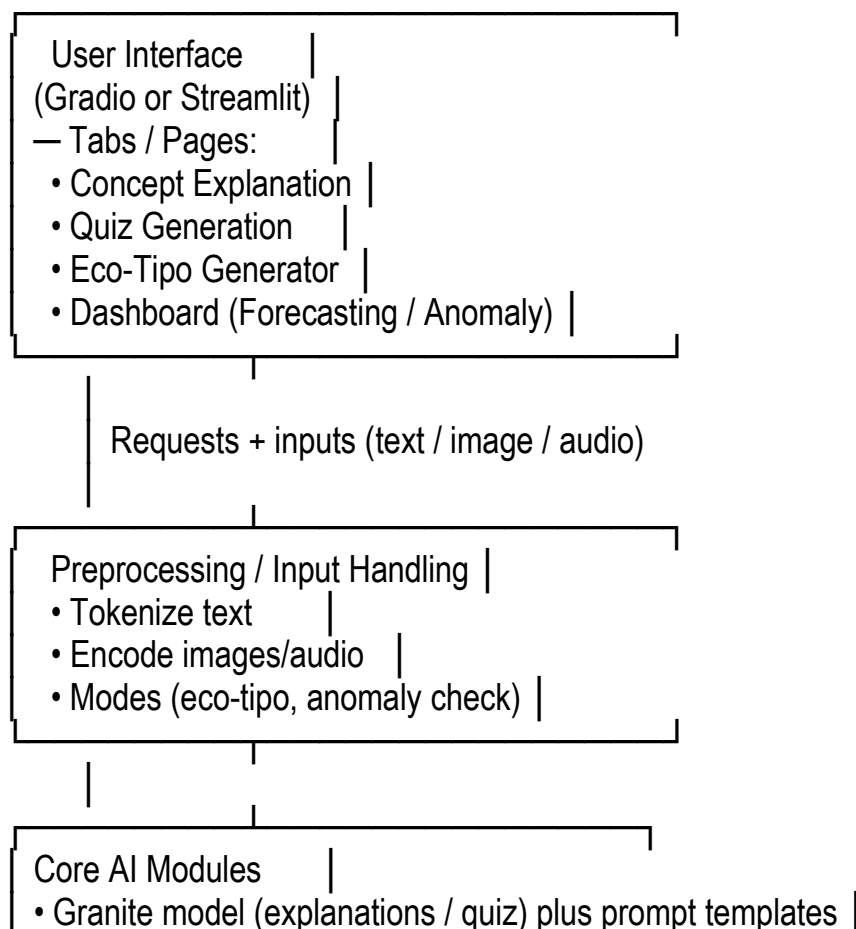
6. TECHNICAL AND RESOURCE CONSIDERATIONS.

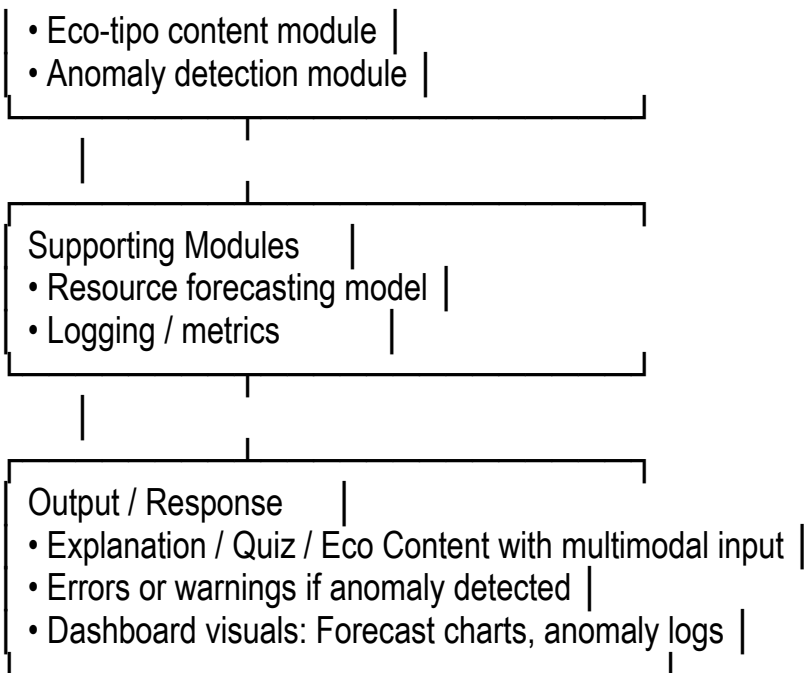
- ❖ Computational cost will increase. With multimodal models for images or audio. And anomaly detection. GPU usage or cloud resources may become necessary..
- ❖ Eco-Tipo content might require domain knowledge. Or curated data. To produce accurate content.
- ❖ UI complexity. Combining several tabs and modes increases maintenance.

7. FINALLY, OUTPUT AND RESPONSE LAYER.

- ❖ **It spits out explanations or quizzes or eco content, even with multimodal inputs.**
- ❖ **Errors or warnings pop up if an anomaly shows.**
- ❖ **Dashboard visuals include forecast charts and anomaly logs.**

Below is a high-level architecture sketch of how the extended system could be structured.





EDUTUTORAI SYSTEM ARCHITECTURE

DOCUMENTATION

1. ARCHITECTURAL OBJECTIVES

The goals here focus on scalability. That means forecasting resource demand ahead of time. It provides diverse educational content. Including eco-themed learning materials. Content reliability improves with anomaly detection. Multimodal inputs make the learning experience richer. And there's an accessible web-based interface. For both learners and educators.

2. High-Level System Architecture

The system has a modular design. It builds on core layers that flow one into the next. Starts with the user interface layer. Then input handling and preprocessing. Core AI engine uses IBM Granite. Followed by postprocessing and anomaly detection. Output generation comes after that. Finally, resource logging and forecasting engine ties it up.

3. COMPONENT BREAKDOWN

3.1 User Interface Layer

It uses tools like Gradio or Streamlit. The functionality includes a multi-tab interface. One tab for concept explanations. Another for quiz generation. Eco-tipo generator has its own spot. Dashboard shows forecasting and anomaly logs. Inputs cover text, images, and audio. Outputs include explanations, quizzes, eco-content, and graphs.

3.2 Input Handling and Preprocessing

This part handles accepting and validating input types. Text comes via textbox. Images through upload. Audio from recording or upload. It converts multimodal inputs into embeddings. Vision models like CLIP do the images. Speech-to-text models such as Whisper handle audio. There's a basic check for offensive or malformed input. You know, a pre-check for anomalies.

3.3 Core AI Engine

The model is IBM Granite 3.2-2B-Instruct. Functionality centers on text generation from prompt templates. It generates explanations for educational concepts. Multiple question types for quizzes. Eco-friendly content with a sustainability focus. It operates in different modes. Based on the selected tab and input type.

3.4 Prompt Composer

The purpose is to dynamically assemble prompt strings. It uses user input. Selected content mode, like standard or quiz or eco. Optional multimodal data if provided.

3.5 Anomaly Detection Module

Features include flagging inappropriate or irrelevant input. Nonsensical or extremely short output gets caught. Unusual spikes in usage patterns too. Technologies involve heuristics. Like length checks and keyword detection. Optional ML classifiers for content quality scoring.

3.6 Output Generator

Responsibilities cover cleaning and formatting model output for display. It adds headers and numbering. For example, for quiz questions. Content warnings apply if anomalies show up. Final results go to the UI display layer.

3.7 Resource Forecasting Engine

This predicts future request volumes. Topic popularity. Expected server load. Tools include Prophet, ARIMA, or LSTM if needed. Visualizations use Matplotlib or Plotly. Outputs appear in the dashboard tab.

3.8 Logging and Monitoring

The purpose is collecting user requests. Input and output size and type. Timestamps. It supports forecasting model training. Anomaly detection analytics as well.

Detect AI-generated content and give it a human touch with our AI Content Detector. Just paste your text, and you'll get accurate, human-like results in no time

HERE'S THE BREAKDOWN OF THE FOLDER STRUCTURE:

- ❖ EduTutorAI/ → This is the main project directory.
- ❖ app.py → The heart of the application, featuring the Gradio interface.
- ❖ requirements.txt → A handy list of all the dependencies you'll need.
- ❖ README.md → A quick overview of the project and how to use it.
- ❖ utils/ → A collection of helper functions to keep your code clean.
- ❖ tests/ → Scripts for unit and integration testing.
- ❖ docs/ → All the API documentation and guides you'll need.

Running the Application

- ❖ Step 1: Download or clone the project to your local machine.

- ❖ Step 2: Install the dependencies listed in the requirements file.
- ❖ Step 3: Start the application by running the main file.
- ❖ Step 4: Once it's up and running, the app will pop open in your browser.
- ❖ Step 5: If you're using Google Colab, run the notebook and use the shared link to access the tutor remotely.

AUTHENTICATION

- ❖ Make sure to secure the app with a login or token system.
- ❖ Keep your credentials safe in an environment file instead of hardcoding them.
- ❖ Start with a simple authentication method (like username and password).
- ❖ Later on, you can expand to include role-based access (like admin and student).

API DOCUMENTATION

- ❖ Concept Explanation
- ❖ Endpoint: /explain
- ❖ Input: A concept or subject you want to understand.

- ❖ Output: A clear, detailed explanation complete with examples.
- ❖ Quiz Generator
- ❖ Endpoint: /quiz
- ❖ Input: Any topic or subject area you choose.
- ❖ Output: Five diverse quiz questions along with a separate answer section.

TESTING

- ❖ Unit Testing
- ❖ Ensure that explanations are generated when a concept is provided.
- ❖ Check that at least five quiz questions are returned for any given topic.
- ❖ Make sure that invalid or blank inputs trigger a safe error message.
- ❖ Integration Testing
- ❖ Verify that the app launches successfully in a local environment.
- ❖ Confirm that inputs flow through the interface and reach the model.

- ❖ Ensure that outputs are displayed correctly in the user interface.
- ❖ User Testing
- ❖ Encourage users to try explaining a subject and generating a quiz.
- ❖ Gather feedback on clarity, accuracy, and overall ease of use.
- ❖ Make sure the app is responsive, reliable, and genuinely helpful.

KNOWN ISSUES – EDUCATIONAL AI ASSISTANT

1. Model Loading and Performance

- The AI model is large and requires significant resources (RAM/VRAM).
- First-time loading is slow due to heavy model download.
- On systems without a GPU, responses may be delayed.
- In low-spec environments, there is a risk of runtime crashes or “out of memory” errors.

2. Accuracy and Quality of Responses

- While explanations are generally helpful, some outputs may be:
 - Too generic, overly broad, or repetitive.
 - Slightly off-topic compared to the user's input.
- The app does not have built-in fact-checking, so minor inaccuracies may occur.
- Quiz difficulty is not always balanced (some too easy, some too advanced).

3. Authentication and Security

- Current authentication is very basic (simple token or password).
- No encryption or advanced user management is implemented yet.
- Risk of unauthorized access if shared publicly without secure login.

4. Error Handling

- Blank or invalid inputs may return incomplete or confusing outputs.
- Error messages are minimal and not always user-friendly.
- System does not yet support retry mechanisms when the model fails.

5. Usability and Interface Limitations

- Gradio provides a functional interface but:
 - Layout may feel cramped on mobile devices.

- No advanced customization for user experience yet.
- Application currently supports only text-based interaction (no voice or image inputs)

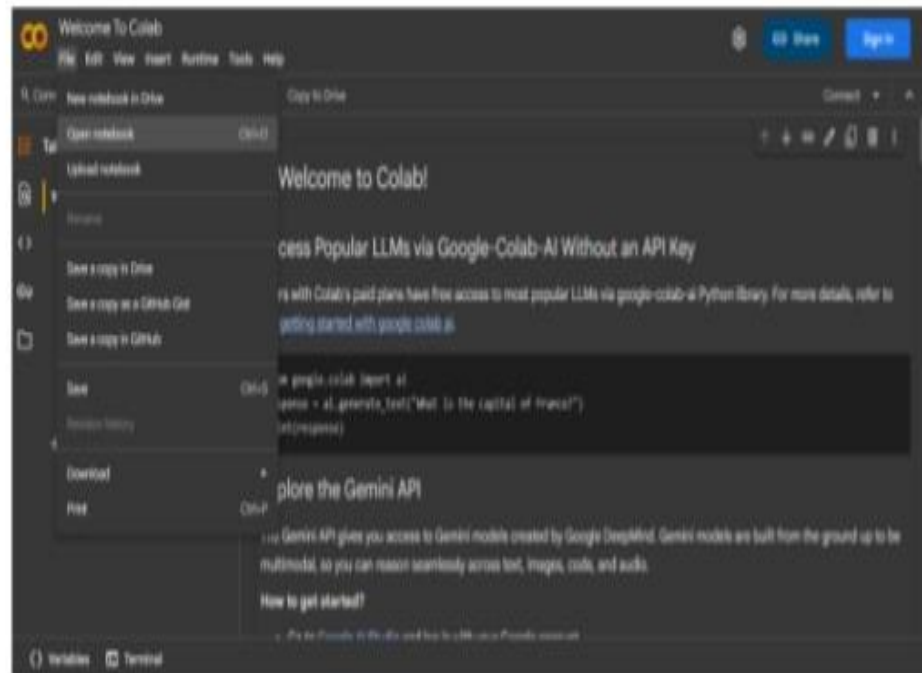
6. Limited Offline Use

- Requires internet connection for initial model download.
- Offline use is inconsistent and depends on local hardware capacity.

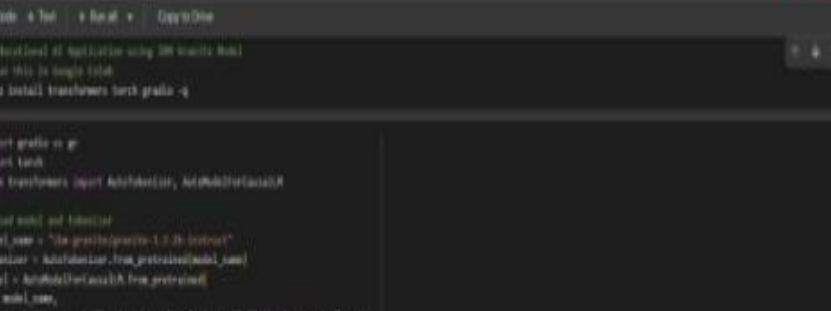
7. Testing Constraints

- Unit testing covers only basic functions (response generation, quiz count).
- No deep testing yet for performance under heavy user load.
- User testing feedback is limited to small groups, so broader edge cases may be missed.

SCREENSHORTS OF PROJECT



- 1.OPEN THE GOOGLE COLLAB
- 2.OPEN A NEW BOOK
- 3.ADD A TITLE



The screenshot shows the EduTutorAI Python IDE interface. The top bar includes the EduTutorAI logo, a search icon, and a 'Share' button. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The main workspace is divided into two panes. The left pane shows a file explorer with a single file named '0'. The right pane shows a Jupyter Notebook with a single cell containing Python code. The code is as follows:

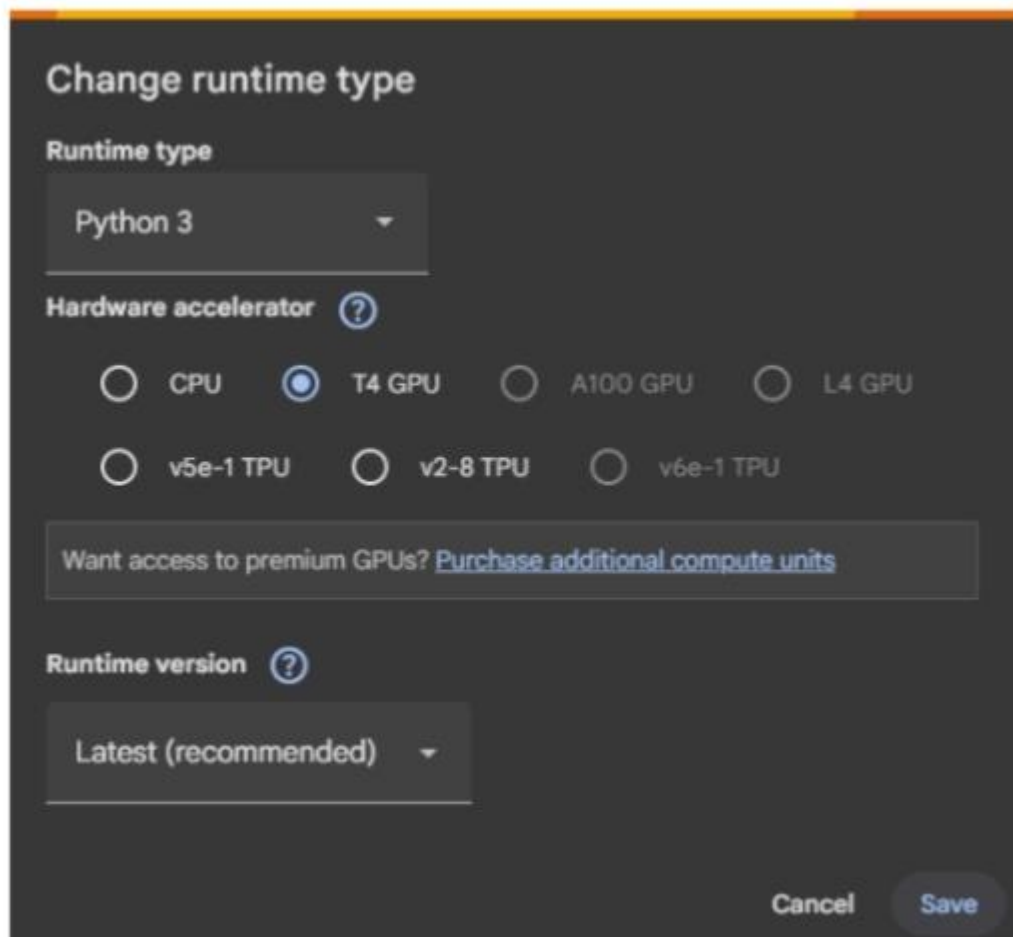
```
1 # Import necessary libraries
2 import torch
3 from transformers import AutoTokenizer, AutoModelForCausalLM
4
5 # Load model and tokenizer
6 model_name = "gpt3.5-turbo-1.5.0-instruct"
7 tokenizer = AutoTokenizer.from_pretrained(model_name)
8 model = AutoModelForCausalLM.from_pretrained(
9     model_name,
10     torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
11     device_map="auto" if torch.cuda.is_available() else None
12 )
13
14 # If tokenizer.pad_token is None:
15 tokenizer.pad_token = tokenizer.eos_token
16
17 def generate_response(prompt, max_length=100):
18     inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=100)
19
20     if torch.cuda.is_available():
21         inputs = {k: v.to(model.device) for k, v in inputs.items()}
22
23     with torch.no_grad():
24         outputs = model.generate(
25             *inputs,
26             max_length=max_length,
27             temperature=0.7,
28             do_sample=True,
29             pad_token=tokenizer.eos_token_id
30         )
31     return outputs[0][len(prompt):].tolist()
```

```

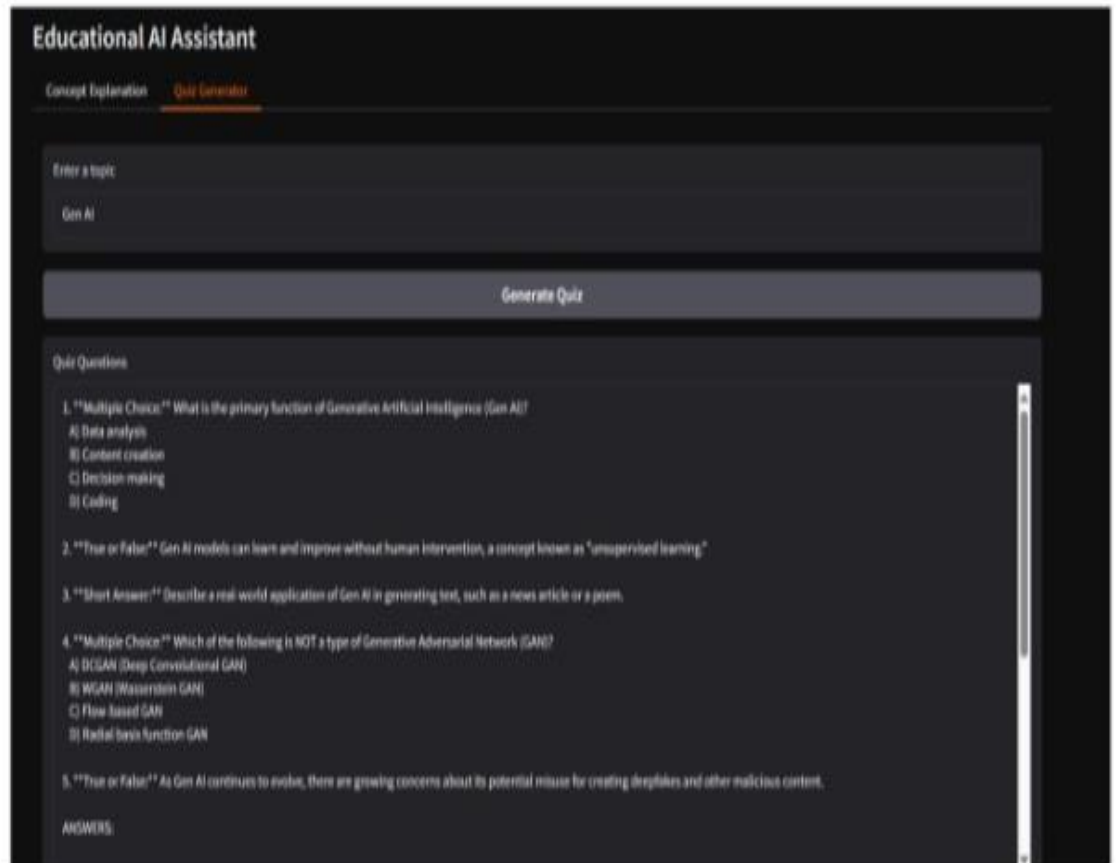
1 response = take(nizer.docx(outputs[0], skip_special_tokens=True))
2 response = response.replace(prompt, "").strip()
3 return response
4
5 def concept_explanation(concept):
6     prompt = f"Explain the concept of {concept} in detail with examples."
7     return generate_response(prompt, max_length=400)
8
9 def quiz_generator(concept):
10    prompt = f"Generate 5 quiz questions about {concept} with different question types (multiple choice, true/false, short answer). At the end, provide all the answers in a separate ANSWER section."
11    return generate_response(prompt, max_length=400)
12
13 # Create a simple interface
14 with gr.Blocks() as app:
15     gr.Markdown("A Educational AI Assistant")
16
17     with gr.Tab():
18         with gr.Labeled("Concept Explanation"):
19             concept_input = gr.Textbox(label="Enter a concept", placeholder="e.g., Newton's law")
20             explain_btn = gr.Button("Explain")
21             explanation_output = gr.Textbox(label="Explain here", lines=10)
22
23             explain_btn.click(concept_explanation, inputs=[concept_input], outputs=[explanation_output])
24
25         with gr.Labeled("Quiz Generator"):
26             quiz_input = gr.Textbox(label="Enter a topic", placeholder="e.g., Physics")
27             quiz_btn = gr.Button("Generate Quiz")
28             quiz_output = gr.Textbox(label="Quiz questions", lines=10)
29
30             quiz_btn.click(quiz_generator, inputs=[quiz_input], outputs=[quiz_output])
31
32 # Run the application

```

- ❖ DEPLOY THE CODE
- ❖ MAKE IT RUN THROUGH
- ❖ IN A GOOGLE COLLAB



- ❖ IN THIS WE NEED TO CHANGE THE RUNTYPE
- ❖ WE NEED TO SELECT T4 GPU
- ❖ SELECT THE TYPO FOR RUNTIME THE CODE



THE OUTPUT IS VISIBLE THROUGH THIS SLIDES

❖ THE OUTPUT TO PLATES WITH THE LINK THIS GENERATED IN STUDIO HEDGE

FUTURE ENHANCEMENTS – EDUCATIONAL AI ASSISTANT

The Educational AI Assistant has a lot of room to grow beyond what it currently offers. Here's a look at the exciting enhancements and improvements we're planning to tackle existing challenges and broaden its capabilities:

1. Improved Model Performance

- ❖ We're working on optimizing model loading and cutting down on resource usage through quantization and lighter versions.
- ❖ We'll also support smaller, offline-friendly models for those using low-end devices.
- ❖ Plus, we'll enable caching for frequently accessed topics to make repeated queries faster.

2. Advanced Accuracy and Personalization

- ❖ The quiz generator will be enhanced to offer a balanced range of difficulty levels (easy, medium, hard)
- ❖ We'll introduce adaptive learning, allowing the AI to tailor explanations and quizzes based on how users are progressing.
- ❖ Responses will be personalized to match the user's study level, whether they're a beginner, intermediate, or advanced learner.

3. Robust Authentication and Security

- ❖ We'll implement multi-user login with role-based access (Admin, Teacher, Student).
- ❖ For secure API authentication, we'll use JWT tokens or OAuth2.
- ❖ All communication will be protected with end-to-end encryption.
- ❖ We'll also set up session management and user activity logs for better oversight.

4. Enhanced User Experience (UX/UI)

- ❖ A responsive, mobile-first UI will be developed to improve learning on phones and tablets.
- ❖ We'll add voice input/output support to make the experience more interactive and accessible.
- ❖ Users will have the option to choose between dark and light themes, along with customizable layouts.
- ❖ And we'll allow users to download explanations and quizzes in PDF or Word formats.

5. Extended Learning Features

- ❖ A progress tracker will be added to help users monitor their learning history and quiz performance.
- ❖ We'll introduce flashcards for quick revision.
- ❖ Interactive diagrams, charts, or videos will accompany text explanations for a richer learning experience.
- ❖ Multilingual support will be enabled so learners can study in their preferred language.

6. Offline and Hybrid Mode

- ❖ We're enhancing offline functionality by bundling lighter models within the app.
- ❖ In a hybrid setup, users can utilize a local model for simple tasks while connecting to a cloud model for more complex ones.

7. Scalability and Collaboration

- ❖ We're focusing on scalability to ensure the assistant can grow with user needs and facilitate collaboration among users.

FLOWCHART

