

# Edu Tutor AI Personalized Learning

## 1. Introduction

- Project title : Edu Tutor AI Assistant
- Team leader : SUBASH M
- Team member : SARAVANAN P
- Team member : RAYAPPAN B
- Team member : SRIDHARAN D

## 2. Project overview

- Purpose:

□ The purpose of this program is to provide an AI-powered educational assistant that supports both learners and educators. It helps users understand complex concepts by generating clear and detailed explanations with relevant examples. The program also creates quizzes on various topics, offering multiple types of questions along with answers for self-assessment. By combining concept learning with

interactive testing, it enhances engagement and reinforces knowledge retention. Overall, it aims to make learning more accessible, effective, and personalized through AI assistance.

- **Feature:**

1. Concept Explanation – Provides detailed explanations of any concept with examples to aid understanding.
2. Quiz Generator – Creates 5 quiz questions on a given topic with multiple formats (MCQ, true/false, short answer).
3. Answer Key Section – Supplies correct answers to generated quizzes for easy self-assessment.
4. Interactive User Interface – Built with Gradio, offering tabs for concept learning and quiz generation in a simple layout.
5. AI-Powered Responses – Uses the IBM Granite model to generate accurate, context-aware, and natural language outputs.
6. Customizable Output – Users can input any topic or concept, and the assistant adapts responses

accordingly.

7. Cloud Sharing Option – The app can be launched with a shareable link, making it accessible across devices.

## 3. Architecture

### 1. User Interface Layer (Frontend)

Built with Gradio Blocks.

Provides two main tabs:

Concept Explanation Tab for entering concepts and viewing explanations.

Quiz Generator Tab for entering topics and generating quizzes.

### 2. Application Layer (Backend Logic)

Contains Python functions:

`concept_explanation()` → Builds a prompt and requests a detailed explanation.

`quiz_generator()` → Builds a prompt and generates quiz questions + answers.

`generate_response()` → Core function that interacts with the AI model to produce text.

### 3. AI Model Layer

Uses Hugging Face Transformers with the IBM Granite 3.2 2B Instruct model.

Tokenizer processes user input into model-readable format.

Model generates responses based on prompts, then outputs natural text.

## 4. Processing & Response Handling

Model outputs are decoded back into human-readable text.

Post-processing removes unnecessary tokens and cleans results before sending to the UI.

## 5. Deployment Layer

Runs locally or on GPU (if available).

Supports Gradio share=True, enabling public access via a shareable link.

# 3. Setup Instructions

## 1. System Requirements

OS: Windows, macOS, or Linux

Python: 3.9 or later

Hardware:

CPU (works fine)

GPU (recommended for faster inference, supports CUDA)

Internet connection (required to download model + run Gradio app)

## 2. Install Required Packages

Open a terminal (or Google Colab cell) and run:

```
pip install torch transformers gradio -q
```

torch → Deep learning framework for running models.

transformers → Hugging Face library to load IBM Granite model.

gradio → To create the web-based interface.

## 5. Folder Structure

app.py → Entry point for running the Gradio app.

src/ → Contains all logic neatly separated (model loading, explanation, quiz).

tests/ → For testing and debugging.

docs/ → For project reports, architecture diagrams, etc.

## 6. Running the Application

### 1. User Interface Layer (Frontend)

Built with Gradio Blocks.

Provides two main tabs:

Concept Explanation Tab for entering concepts and viewing explanations.

Quiz Generator Tab for entering topics and generating quizzes.

### 2. Application Layer (Backend Logic)

Contains Python functions:

`concept_explanation()` → Builds a prompt and requests a detailed explanation.

`quiz_generator()` → Builds a prompt and generates quiz questions + answers.

`generate_response()` → Core function that interacts with the AI model to produce text.

## 7. API Documentation

### 1. Concept Explanation

Input: Textbox for concept name

Output: Textbox with detailed explanation

Button: "Explain" triggers `concept_explanation()`

## 2. Quiz Generator

Input: Textbox for topic name

Output: Textbox with quiz questions and answers

Button: "Generate Quiz" triggers `quiz_generator()`.

## 8. Authentication

`auth` parameter in `app.launch()` takes a dictionary of username/password pairs.

When a user opens the app, they will be prompted to enter username and password.

Only users with valid credentials can access the application.

For production, consider hashing passwords instead of storing plain text.

You can integrate with Google OAuth or other authentication providers for more secure login.

You can create role-based access (e.g., students vs teachers) to allow different features.

## 9. User Interface

### ► Concept Explanation

Purpose: To allow users to enter a concept and receive a detailed explanation.

Components:

Component	Type	Description
Textbox	Input	Users enter the concept name (e.g., "Machine Learning").
Button	Action	"Explain" button triggers the explanation generation.
Textbox	Output	Displays the AI-generated explanation of the concept.

Interaction Flow:

1. User types a concept in the input textbox.
2. User clicks Explain.
3. AI generates a detailed explanation with examples.
4. Explanation appears in the output textbox.

### ► Quiz Generator

Purpose: To generate quiz questions and answers for a given topic.

Components:



Component	Type	Description
Textbox	Input	Users enter a topic name (e.g., "Physics").
Button	Action	"Generate Quiz" button triggers quiz generation.
Textbox	Output	Displays the AI-generated quiz questions along with answers.

Interaction Flow:

1. User enters a topic in the input textbox.
2. User clicks Generate Quiz.
3. AI generates 5 quiz questions with different formats (MCQ, True/False, Short Answer).
4. Quiz and answer section appear in the output textbox.

## 10. Testing

Integration / Functional Testing

Open the Gradio interface in a browser.

Concept Explanation Tab:

1. Enter a topic (e.g., "Machine Learning")
2. Click Explain
3. Verify the output is relevant, readable, and

complete.

Quiz Generator Tab:

1. Enter a topic (e.g., "Physics")
2. Click Generate Quiz
3. Verify that 5 questions appear with an answers section.
4. Performance Testing

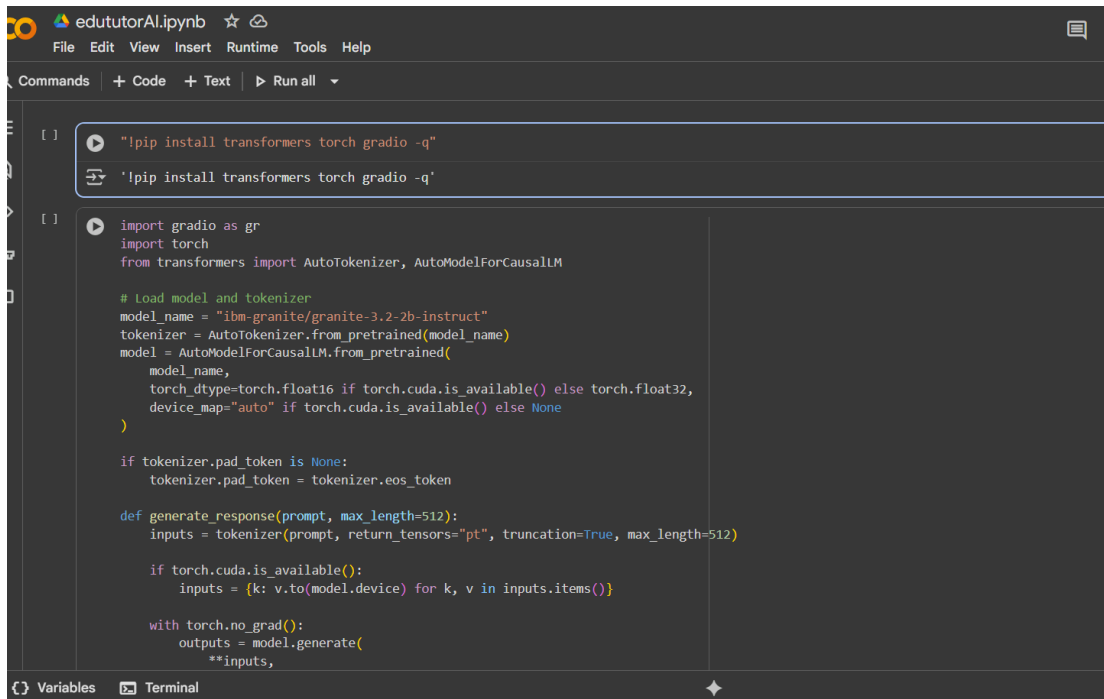
Check response time on CPU vs GPU.

Monitor memory usage for large prompts (max\_length > 1000).

Adjust max\_length or temperature if generation is too slow.r.

## **11. screen shots**

### **Input**



```
[ ]: !pip install transformers torch gradio -q

[ ]: import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

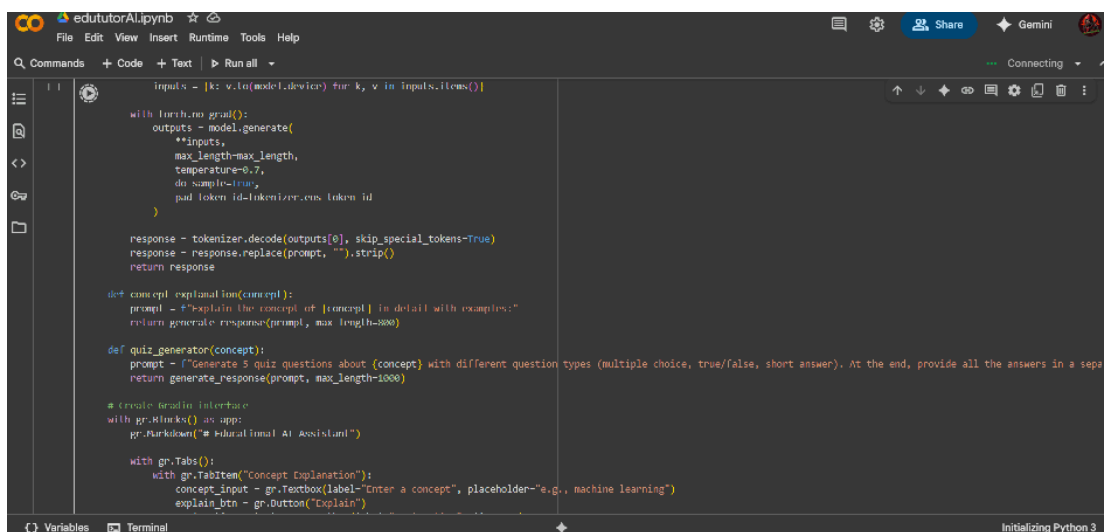
# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=512):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
```



```
inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

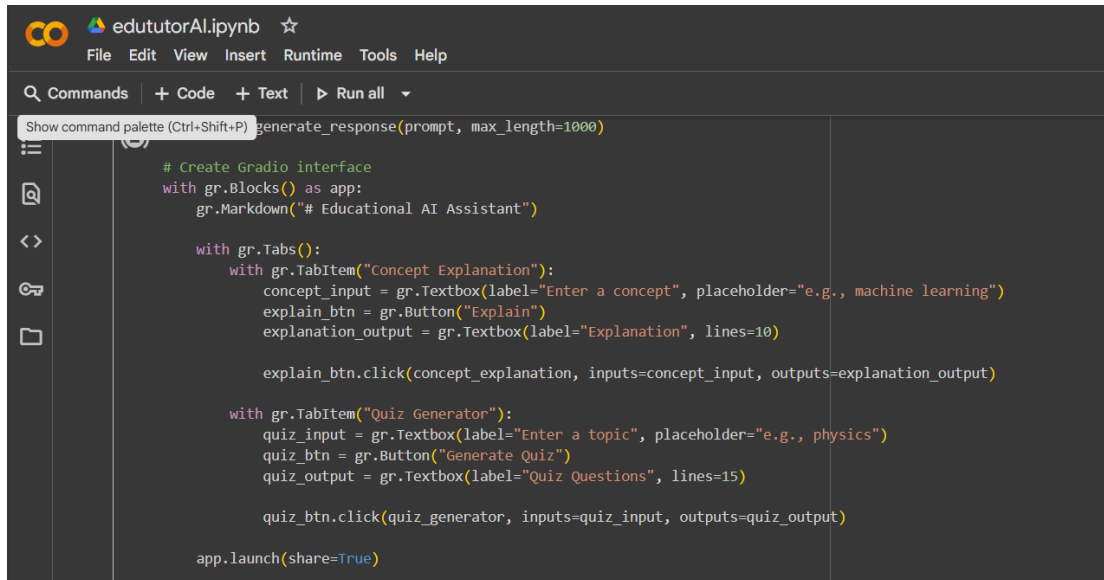
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

def concept_explanation(concept):
    prompt = f"explain the concept of {concept} in detail with examples:"
    return generate_response(prompt, max_length=800)

def quiz_generator(concept):
    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 sepa"
    return generate_response(prompt, max_length=1000)

# Create Gradio Interface
with gr.Blocks() as app:
    gr.Markdown("# Edututorial AI Assistant")

    with gr.Tab():
        with gr.Tabitem("Concept Explanation"):
            concept_input = gr.Textbox(label="Enter a concept", placeholder="e.g., machine learning")
            explain_btn = gr.Button("Explain")
```



The screenshot shows a Jupyter Notebook titled "edututorAI.ipynb" with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar (Commands, Code, Text, Run all). The code in the notebook is as follows:

```
generate_response(prompt, max_length=1000)

# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# Educational AI Assistant")

    with gr.Tabs():
        with gr.TabItem("Concept Explanation"):
            concept_input = gr.Textbox(label="Enter a concept", placeholder="e.g., machine learning")
            explain_btn = gr.Button("Explain")
            explanation_output = gr.Textbox(label="Explanation", lines=10)

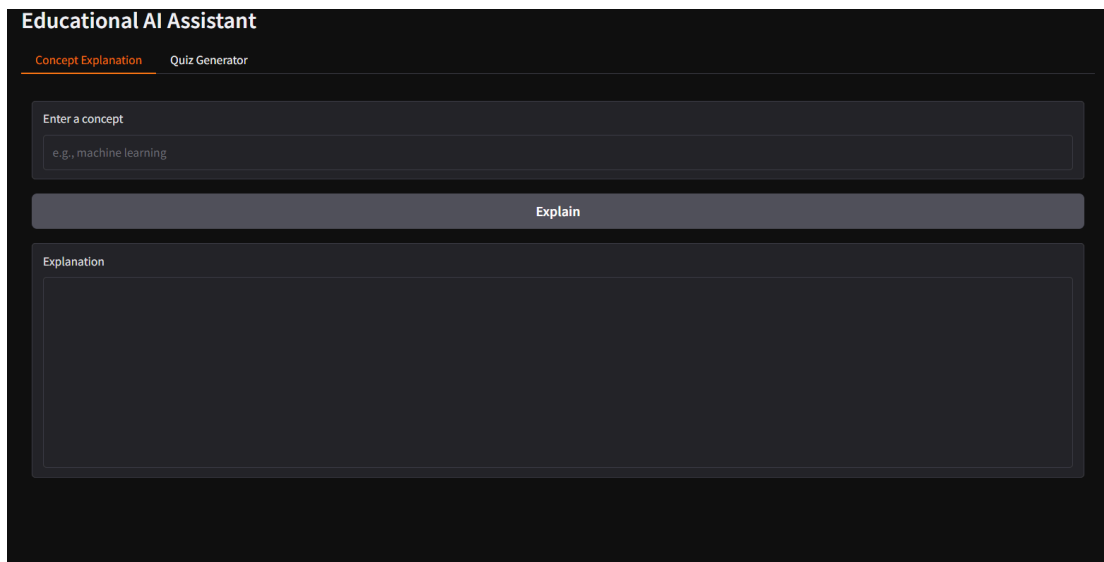
            explain_btn.click(concept_explanation, inputs=concept_input, outputs=explanation_output)

        with gr.TabItem("Quiz Generator"):
            quiz_input = gr.Textbox(label="Enter a topic", placeholder="e.g., physics")
            quiz_btn = gr.Button("Generate Quiz")
            quiz_output = gr.Textbox(label="Quiz Questions", lines=15)

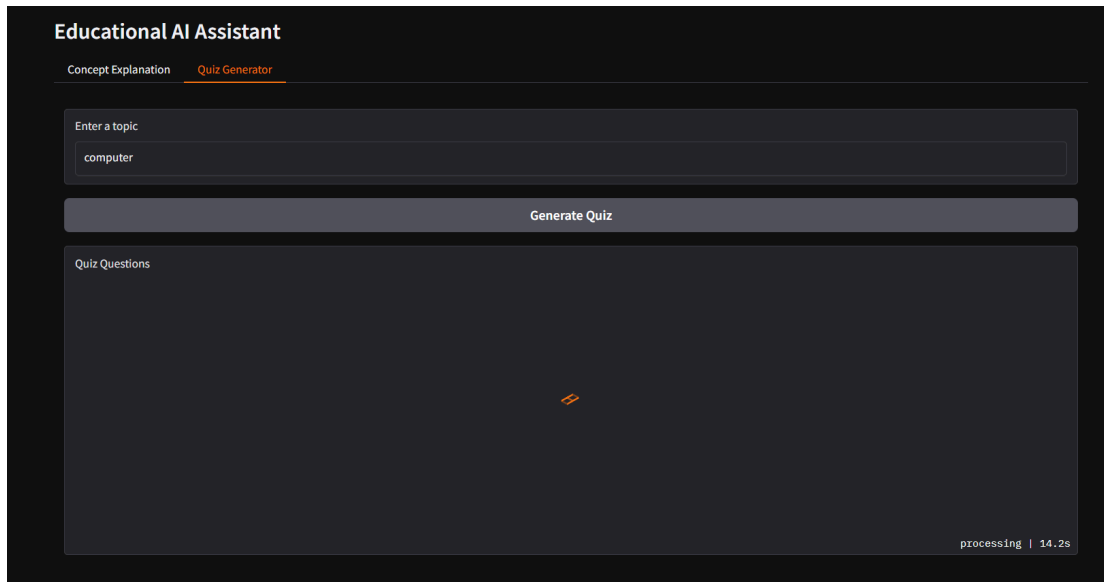
            quiz_btn.click(quiz_generator, inputs=quiz_input, outputs=quiz_output)

    app.launch(share=True)
```

# Output



The screenshot shows the web interface of the Educational AI Assistant. It has a dark theme and a title bar "Educational AI Assistant". There are two tabs: "Concept Explanation" (active) and "Quiz Generator". The "Concept Explanation" tab contains a text input field labeled "Enter a concept" with a placeholder "e.g., machine learning". Below the input field is a button labeled "Explain". Below the button is a large text area labeled "Explanation" for the output.



## 12. Known Issues

- ◆ Large Model Size and Memory Usage
- ◆ Response Accuracy / Relevance
- ◆ Limited Input Length
- ◆ Gradio UI Limitations
- ◆ Internet Connection for Model Download
- ◆ No Advanced Error Handling
- ◆ Quiz Answer Verification

## 13. Future enhancement

- ◆ User Authentication & Profiles
- ◆ Expanded Question Types for Quizzes

- ◆ Improved AI Model Integration
- ◆ Rich Text and Multimedia Support
- ◆ Offline Mode
- ◆ Analytics and Progress Reports
- ◆ Mobile-Friendly Interface
- ◆ Multilingual Support
- ◆ Error Handling and Logging
- ◆ API Access for Developers