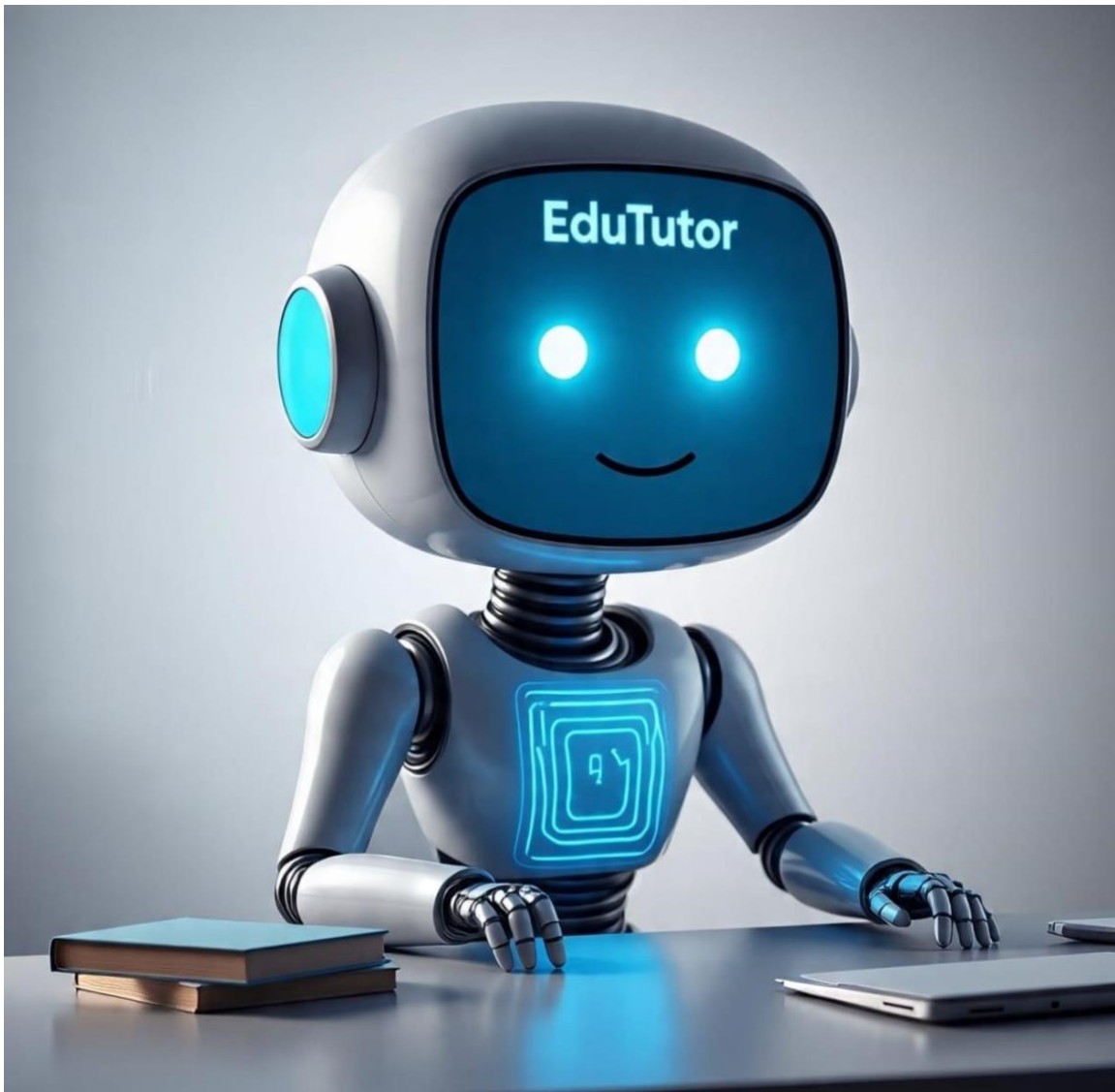# Edu Tutor AI: Personalized Learning

Generate AI with IBM

EduTutor AI

Project Documentation

1. Introduction

• Project Title: EduTutor AI

Team Member: KANAGADHARSHINI S

Team Member: KANNIKA PARAMESWARI R

Team Member: DHANA SRI M P

Team Member: MAITHILI R

2. Project Overview

Purpose:
EduTutor AI is designed to act as an intelligent educational assistant for learners. It leverages Large Language Models (LLMs) to explain academic concepts in simple language and generate quizzes dynamically. The assistant enhances self-learning by offering both explanations and assessments in an interactive way.

Features:
- Concept Explanation
  Key Point: AI-driven explanations
  Functionality: Provides detailed explanations of academic concepts with real-life examples.

- Quiz Generator
  Key Point: Assessment support
  Functionality: Generates 5 quiz questions on a given topic with multiple formats (MCQs, True/False, Short Answer) and provides an ANSWERS section.

- Conversational Interface
  Key Point: Natural language interaction
  Functionality: Learners can ask for explanations or quizzes in plain text.

- User-Friendly Gradio Interface

Key Point: Simple and interactive UI
Functionality: Web-based tabs for concept explanation and quiz generation.

3. Architecture

- Frontend (Gradio): Provides a clean and interactive web UI with tabs for "Concept Explanation" and "Quiz Generator."
- Backend (Transformers + PyTorch): Uses IBM Granite LLM for text generation. Handles tokenization, model inference, and GPU acceleration.
- LLM Integration (Hugging Face Transformers): Tokenizer ensures input formatting. Model generates structured responses.

4. Setup Instructions

Prerequisites:
- Python 3.9+
- pip and virtual environment tools
- Gradio
- Hugging Face Transformers
- Torch with CUDA support (optional)

Installation:
1. Clone the repository
2. Install dependencies: pip install -r requirements.txt
3. Run the app: python app.py
4. Access the Gradio link to interact with EduTutor AI

5. Folder Structure

```
EduTutor-AI/
│── app.py              # Main application script
│── requirements.txt       # Dependencies
│── docs/             # Documentation files
│── models/              # Pretrained model references
│── utils/           # Helper functions (if extended later)
```

6. Running the Application

- Launch the Gradio interface by running app.py
- Navigate between Concept Explanation and Quiz Generator tabs

- Input the desired topic and view the AI-generated output in real time

7. API Documentation

(Optional for future extension)
- POST /explain – Returns AI-generated explanation of a concept
- POST /quiz – Generates quiz questions with answers

8. Authentication

Currently open environment. Future
enhancements:
- Token-based authentication
- User session tracking for personalized learning

9. User Interface
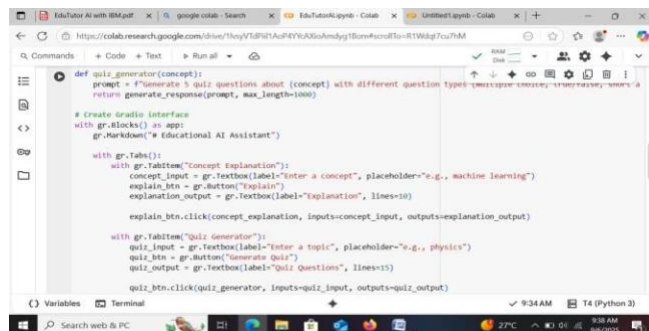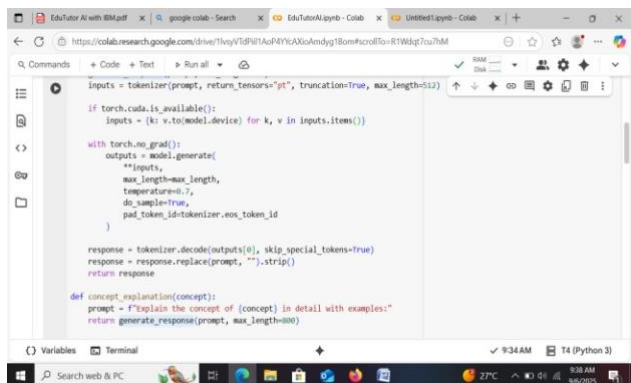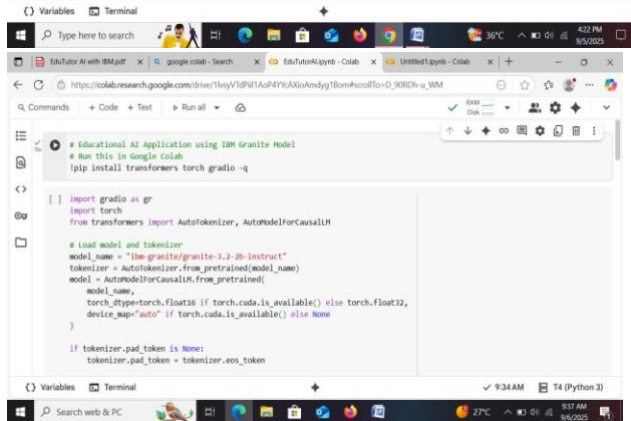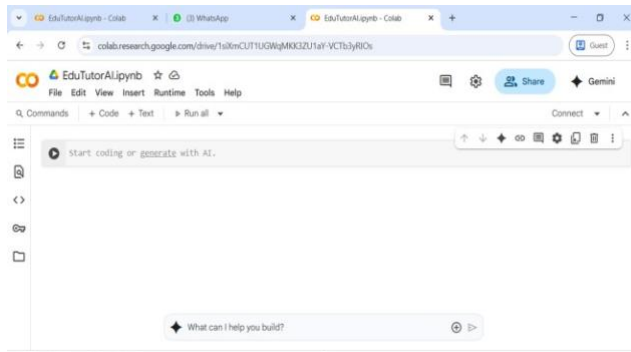
- Tabs for concept explanation and quiz generation
- Textbox inputs for user queries
- Textbox outputs for displaying explanations and quizzes
- Minimalist design for accessibility

10. Testing

- Unit Testing: Prompt generation and tokenizer handling
- Manual Testing: Interaction through Gradio UI
- Edge Cases: Empty inputs, large prompts, GPU/CPU compatibility

11. Screenshots

Screenshot 1 — Google Colab notebook "EduTutorAI.ipynb"

```
Start coding or generate with AI.
```

What can I help you build?

Screenshot 2 — Colab code cell

```python
# Educational AI Application using IBM Granite Model
# Run this in Google Colab
!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
```


Screenshot 3 — Colab code cell

```python
    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,
            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)
```


Screenshot 4 — Colab code cell

```python
def quiz_generator(concept):
    prompt = f"Generate 5 quiz questions about {concept} with different question types (multiple choice, true/false, short a
    return 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)
```

```python
# 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)
```



```python
        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)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
`torch_dtype` is deprecated! Use `dtype` instead!
Loading checkpoint shards: 100%     2/2 [00:40<00:00, 16.53s/it]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://bed73eba6cee17ae9a.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the
```
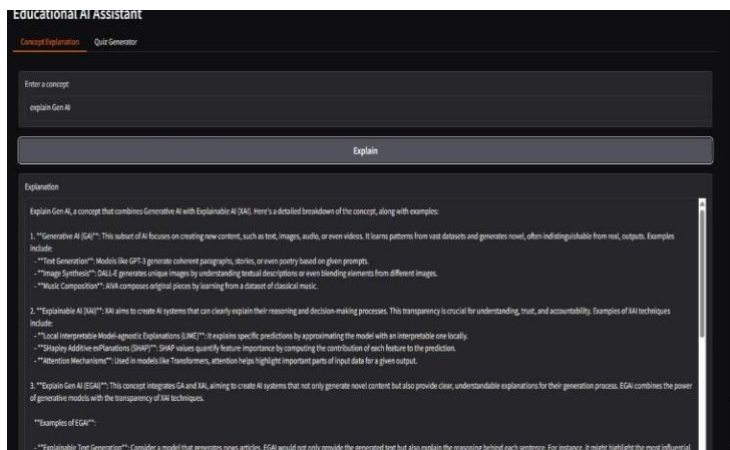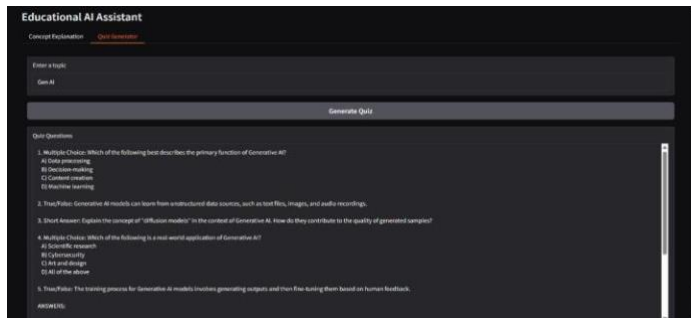

Educational AI Assistant

Concept Explanation   Quiz Generator

Enter a concept

explain Gen AI

Explain

Explanation

Explain Gen AI, a concept that combines Generative AI with Explainable AI (XAI). Here's a detailed breakdown of the concept, along with examples:

1. **"Generative AI (GAI)"**: This subset of AI focuses on creating new content, such as text, images, audio, or even videos. It learns patterns from vast datasets and generates novel, often indistinguishable from real, outputs. Examples include:
   - **"Text Generation"**: Models like GPT-3 generate coherent paragraphs, stories, or even poetry based on given prompts.
   - **"Image Synthesis"**: DALL-E generates unique images by understanding textual descriptions or even blending elements from different images.
   - **"Music Composition"**: AIVA composes original pieces by learning from a dataset of classical music.

2. **"Explainable AI (XAI)"**: XAI aims to create AI systems that can clearly explain their reasoning and decision-making processes. This transparency is crucial for understanding, trust, and accountability. Examples of XAI techniques include:
   - **"Local Interpretable Model-agnostic Explanations (LIME)"**: It explains specific predictions by approximating the model with an interpretable one locally.
   - **"SHapley Additive exPlanations (SHAP)"**: SHAP values quantify feature importance by computing the contribution of each feature to the prediction.
   - **"Attention Mechanisms"**: Used in models like Transformers, attention helps highlight important parts of input data for a given output.

3. **"Explain Gen AI (EGAI)"**: This concept integrates GA and XAI, aiming to create AI systems that not only generate novel content but also provide clear, understandable explanations for their generation process. EGAI combines the power of generative models with the transparency of XAI techniques.

   "Examples of EGAI":

   - **"Explainable Text Generation"**: Consider a model that generates news articles. EGAI would not only provide the generated text but also explain the reasoning behind each sentence. For instance, it might highlight the most influential

12. Known Issues

- Long explanations may exceed output box size
- Limited to topics supported by the pretrained Granite model

13. Future Enhancements

- Add voice-based queries
- Extend to subject-specific tutors (Math, Science, etc.)
- Track user progress with learning analytics dashboards
- Integrate FastAPI backend for API-based usage