Edu Tutor AI: Personalized Learning

Project Documentation

1.Introduction

- **Project title : Personalized Learning Project Documentation**
- **Team ID :** NM2025TMID02145
- **Team Leader :** PUVANA T
- **Team member :** RAMYA K
- **Team member :** SANGAVI S
- **Team member :** SANDHIYA M

2.Project Overview 💻

The **Educational AI Assistant** is a web-based application designed to help students and lifelong learners grasp new concepts and test their knowledge. The application leverages a large language model (LLM) to provide detailed explanations and generate customized quizzes on a wide range of topics. The user interface is built using Gradio, allowing for a simple and intuitive experience that runs directly from a Python script.

3.Architecture 🏛️

The application follows a client-server architecture, although it is contained within a single Python file.

- **Frontend (UI Layer):** The user interface is handled entirely by the **Gradio** library. It creates a web-based frontend with interactive components (textboxes, buttons, tabs) that automatically communicate with the backend Python functions.
- **Backend (Application Logic):** The core logic is a single Python script that manages the Gradio interface and the LLM interactions.
- **Core AI Component:** The application utilizes the **ibm-granite/granite-3.2-2b-instruct** pre-trained language model from the Hugging Face transformers library. The model is loaded into memory at startup.
- **Data Flow:** User input from the Gradio UI is passed as a string to the concept_explanation or quiz_generator function. These functions format the input into a specific prompt for the LLM. The generate_response function then executes the model's inference and returns the generated text, which is displayed back to the user via the Gradio interface.

4.Setup Instructions 🛠

To set up and run the application, you need to have Python and the necessary libraries installed.

Prerequisites

- Python 3.8 or higher
- pip (Python package installer)

Installation

1. **Create a Virtual Environment** (Recommended):
   ```
   python -m venv venv
   source venv/bin/activate  # On Windows, use `venv\Scripts\activate`
   ```

2. **Install Required Libraries:**
   ```
   pip install gradio torch transformers
   ```

   Note: The torch library can be large. If you have an NVIDIA GPU, you may want to install the GPU-enabled version for faster performance. Check the PyTorch website for instructions.

5.Folder Structure 📁

The project has a minimal and flat folder structure, as all the code is contained within a single file. For a more robust project, a requirements.txt file should also be included.

```
/educational_ai_assistant/
├── app.py
├── requirements.txt
└── README.md
```

6.Running the Application ▶

To start the Educational AI Assistant, simply execute the Python script from your terminal.

1. Navigate to the directory containing app.py.
2. Run the following command:
   ```
   python app.py
   ```

3. The console will display a local URL (e.g., http://127.0.0.1:7860).

4. Since the Gradio app is launched with share=True, it will also provide a public shareable URL. Open either URL in your web browser to access the application.
5.

7.API Documentation 📜

This application does not expose a public REST API for external consumption. The functions concept_explanation and quiz_generator serve as internal "endpoints" that are called by the Gradio interface. The communication is handled automatically by the Gradio library's backend.

8.Authentication 🔑

The application currently has no user authentication or authorization. It is a publicly accessible tool, and any user with the Gradio share link can use its features. Future enhancements could include adding an authentication layer to enable user-specific features like saving explanations or tracking progress.

9.User Interface 🎨

The user interface is a simple, tab-based layout built with Gradio Blocks.

- **Main Title:** The app is titled **"Educational AI Assistant"**.
- **Tabs:** There are two main tabs:
  - **Concept Explanation:** Contains a text input field for the concept and a multi-line text output field to display the explanation. A button labeled **"Explain"** triggers the generation.
  - **Quiz Generator:** Contains a text input field for the topic and a multi-line text output field for the quiz questions and answers. A button labeled **"Generate Quiz"** starts the quiz generation.

10.Testing

Since no automated test suite is provided, the following manual tests should be performed to ensure the application is functioning correctly.

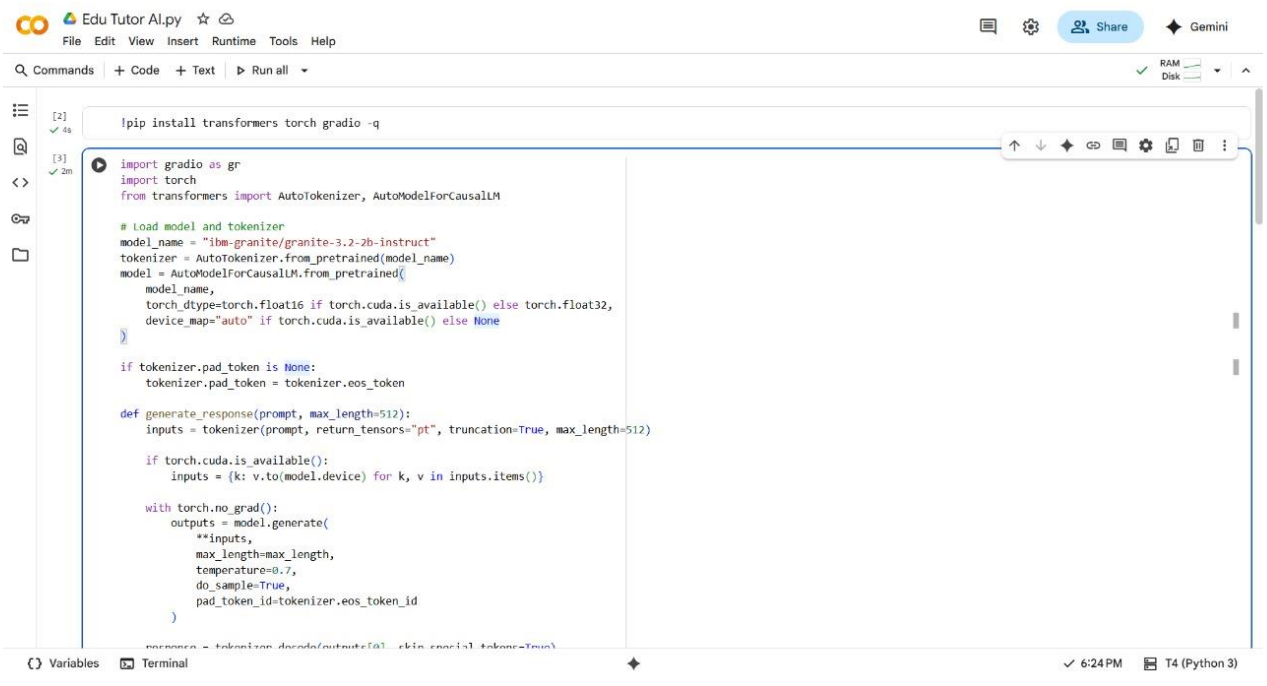
- **Functional Testing:**

- **Concept Explanation Tab:**
  - Enter a simple concept (e.g., "photosynthesis") and click "Explain". Verify that a detailed explanation is generated.
  - Enter a more complex or abstract concept (e.g., "quantum entanglement") and verify the response.
- **Quiz Generator Tab:**
  - Enter a topic (e.g., "solar system") and click "Generate Quiz". Verify that 5 distinct questions are generated.
  - Confirm that the questions include different types (multiple choice, true/false, short answer).
  - Ensure that an **"ANSWERS"** section is present at the end of the generated text.
- **Performance Testing:**
  - Monitor the time it takes for a response to be generated. This will vary significantly between a CPU and GPU setup.
  - Verify that the application does not crash or freeze when generating longer responses.

---

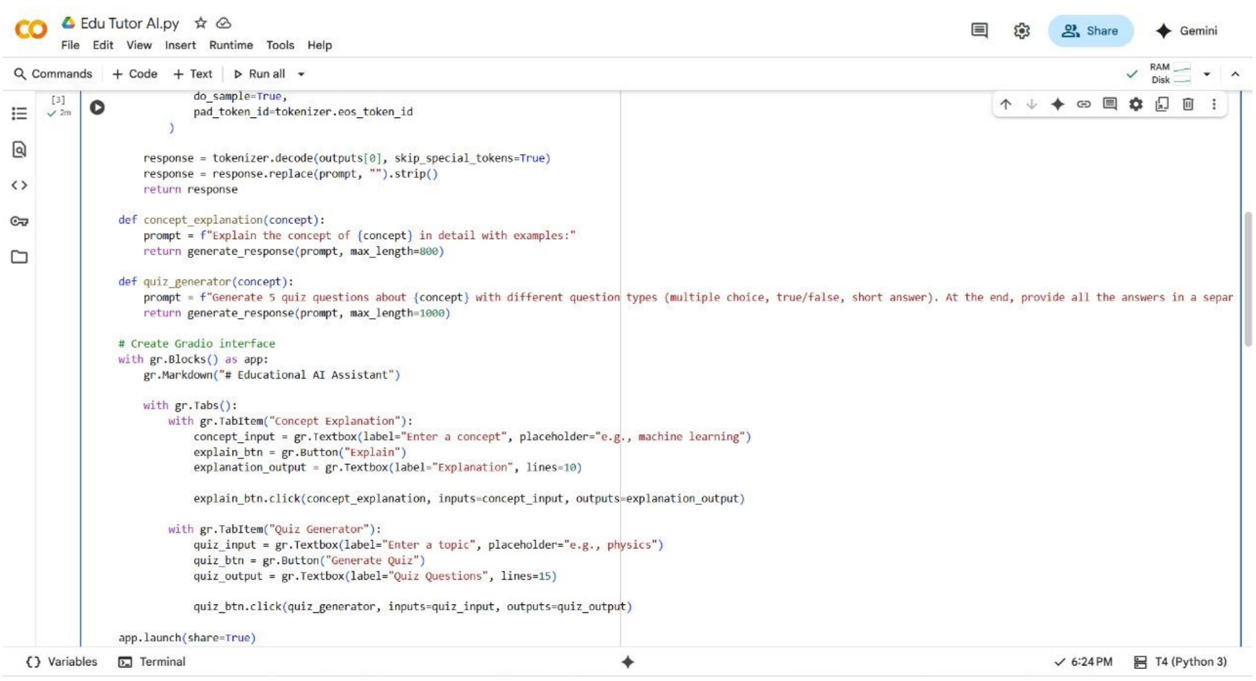
11. Screen shots

1.Input

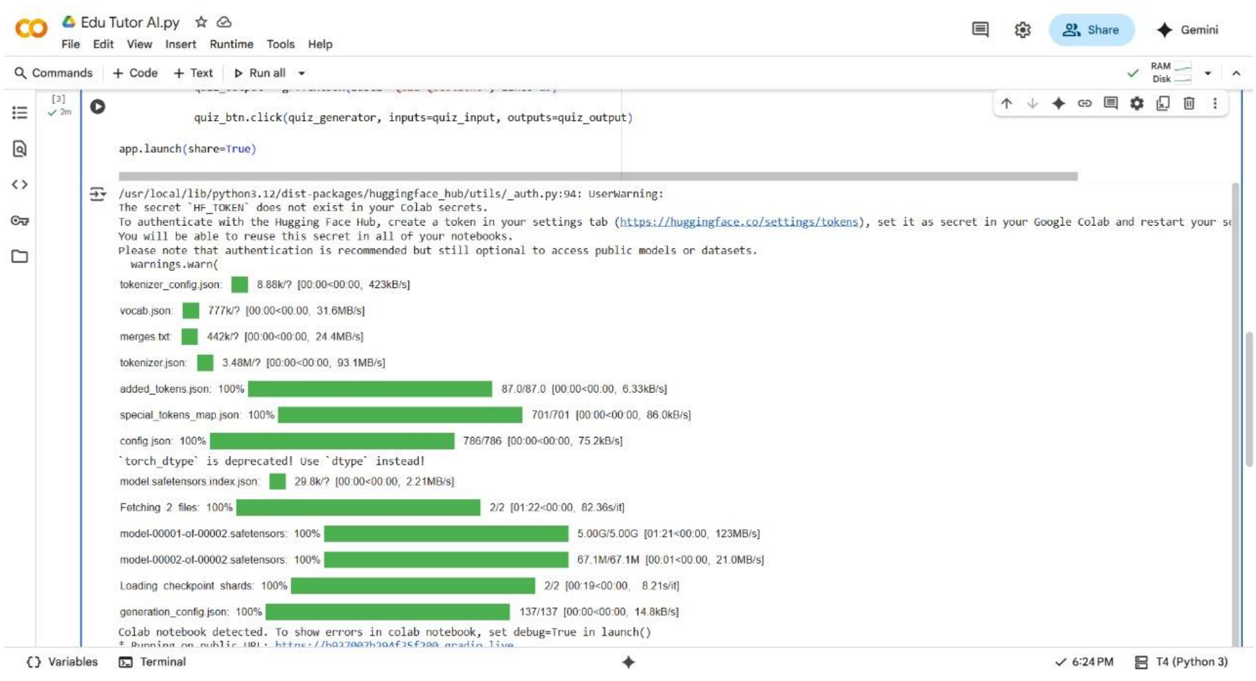


```
!pip install transformers torch gradio -q
```

```
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,
            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)
```

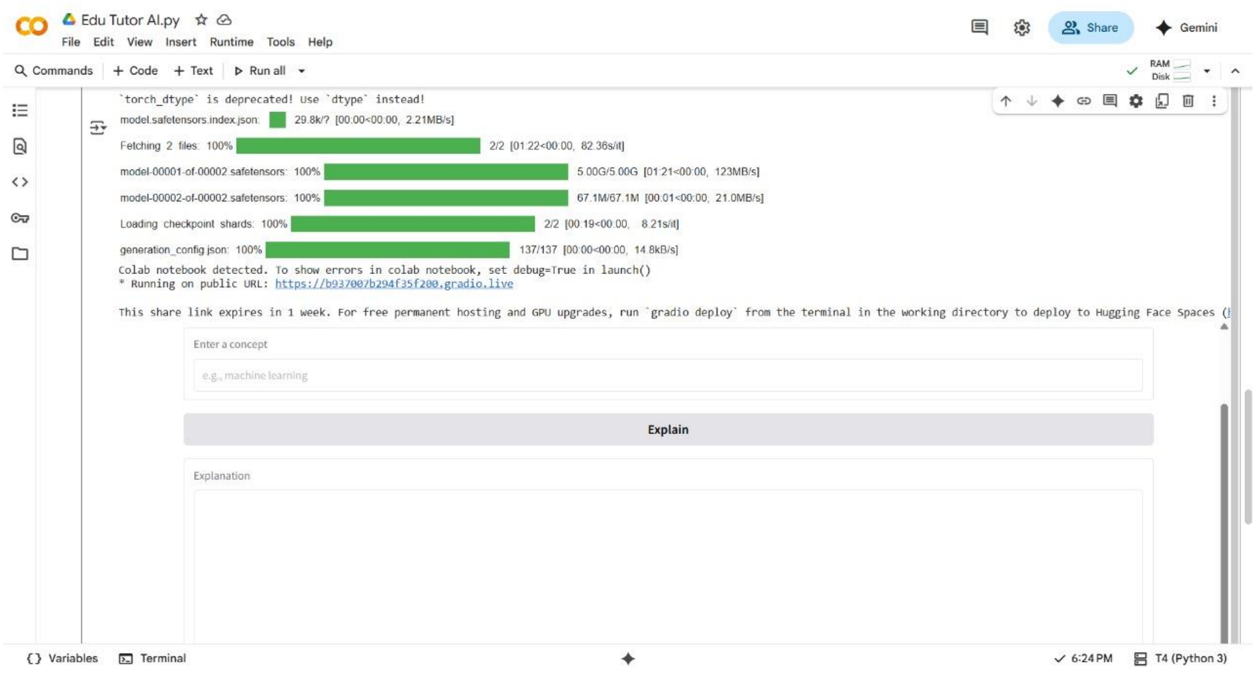


```
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    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 separ
    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)

app.launch(share=True)
```



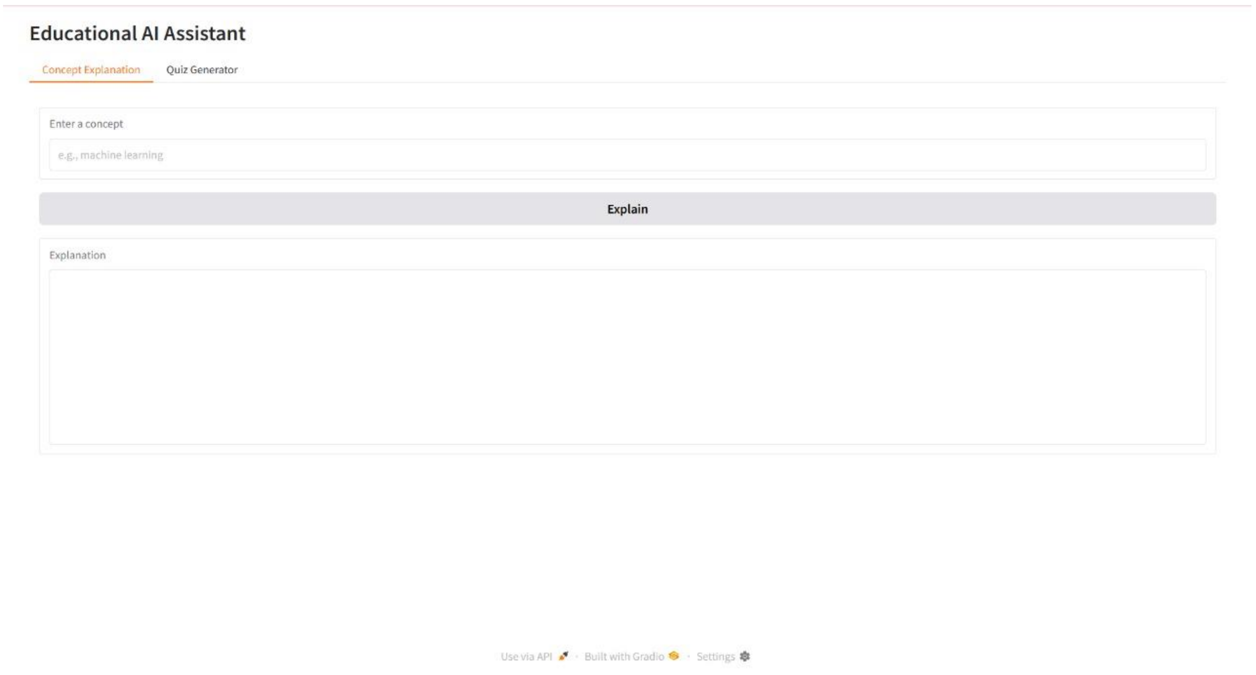
```
            quiz_btn.click(quiz_generator, inputs=quiz_input, outputs=quiz_output)

app.launch(share=True)
```

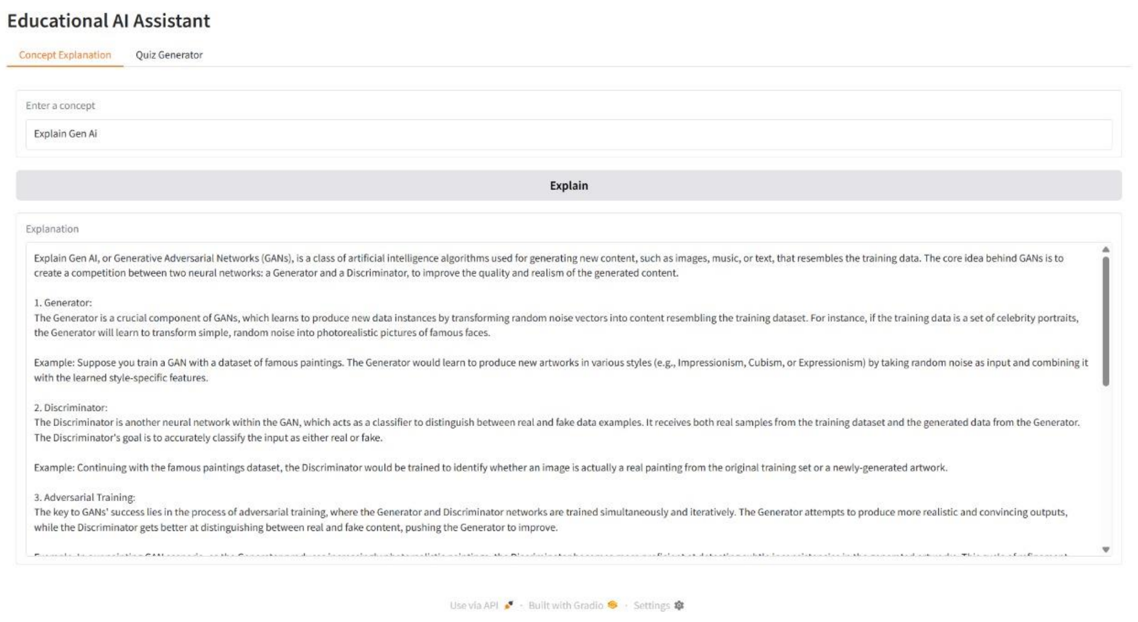
```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your se
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(
tokenizer_config.json: 🟩 8.88k/? [00:00<00:00, 423kB/s]
vocab.json: 🟩 777k/? [00:00<00:00, 31.6MB/s]
merges.txt: 🟩 442k/? [00:00<00:00, 24.4MB/s]
tokenizer.json: 🟩 3.48M/? [00:00<00:00, 93.1MB/s]
added_tokens.json: 100% 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩 87.0/87.0 [00:00<00:00, 6.33kB/s]
special_tokens_map.json: 100% 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩 701/701 [00:00<00:00, 86.0kB/s]
config.json: 100% 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩 786/786 [00:00<00:00, 75.2kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 🟩 29.8k/? [00:00<00:00, 2.21MB/s]
Fetching 2 files: 100% 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩 2/2 [01:22<00:00, 82.36s/it]
model-00001-of-00002.safetensors: 100% 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩 5.00G/5.00G [01:21<00:00, 123MB/s]
model-00002-of-00002.safetensors: 100% 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩 67.1M/67.1M [00:01<00:00, 21.0MB/s]
Loading checkpoint shards: 100% 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩 2/2 [00:19<00:00, 8.21s/it]
generation_config.json: 100% 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩 137/137 [00:00<00:00, 14.8kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://h927007b294f35f200.gradio.live
```



```
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json:  29.8k/? [00:00<00:00, 2.21MB/s]
Fetching 2 files: 100%  2/2 [01:22<00:00, 82.36s/it]
model-00001-of-00002.safetensors: 100%  5.00G/5.00G [01:21<00:00, 123MB/s]
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:01<00:00, 21.0MB/s]
Loading checkpoint shards: 100%  2/2 [00:19<00:00,  8.21s/it]
generation_config.json: 100%  137/137 [00:00<00:00, 14.8kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://b937007b294f35f200.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (
```

2.Output





Explain Gen Ai

Explain Gen AI, or Generative Adversarial Networks (GANs), is a class of artificial intelligence algorithms used for generating new content, such as images, music, or text, that resembles the training data. The core idea behind GANs is to create a competition between two neural networks: a Generator and a Discriminator, to improve the quality and realism of the generated content.

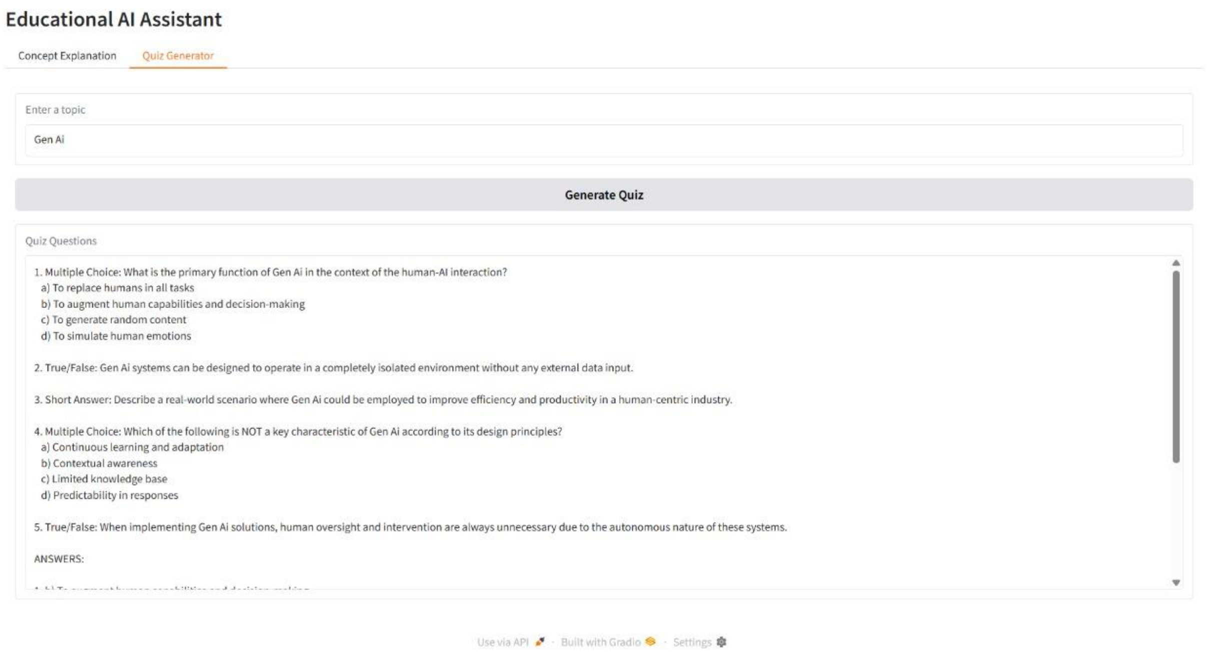
1. Generator:
The Generator is a crucial component of GANs, which learns to produce new data instances by transforming random noise vectors into content resembling the training dataset. For instance, if the training data is a set of celebrity portraits, the Generator will learn to transform simple, random noise into photorealistic pictures of famous faces.

Example: Suppose you train a GAN with a dataset of famous paintings. The Generator would learn to produce new artworks in various styles (e.g., Impressionism, Cubism, or Expressionism) by taking random noise as input and combining it with the learned style-specific features.

2. Discriminator:
The Discriminator is another neural network within the GAN, which acts as a classifier to distinguish between real and fake data examples. It receives both real samples from the training dataset and the generated data from the Generator. The Discriminator's goal is to accurately classify the input as either real or fake.

Example: Continuing with the famous paintings dataset, the Discriminator would be trained to identify whether an image is actually a real painting from the original training set or a newly-generated artwork.

3. Adversarial Training:
The key to GANs' success lies in the process of adversarial training, where the Generator and Discriminator networks are trained simultaneously and iteratively. The Generator attempts to produce more realistic and convincing outputs, while the Discriminator gets better at distinguishing between real and fake content, pushing the Generator to improve.



Gen Ai

1. Multiple Choice: What is the primary function of Gen Ai in the context of the human-AI interaction?
  a) To replace humans in all tasks
  b) To augment human capabilities and decision-making
  c) To generate random content
  d) To simulate human emotions

2. True/False: Gen Ai systems can be designed to operate in a completely isolated environment without any external data input.

3. Short Answer: Describe a real-world scenario where Gen Ai could be employed to improve efficiency and productivity in a human-centric industry.

4. Multiple Choice: Which of the following is NOT a key characteristic of Gen Ai according to its design principles?
  a) Continuous learning and adaptation
  b) Contextual awareness
  c) Limited knowledge base
  d) Predictability in responses

5. True/False: When implementing Gen Ai solutions, human oversight and intervention are always unnecessary due to the autonomous nature of these systems.

ANSWERS:

12. Conclusion

The Educational AI Assistant is a successful proof-of-concept that demonstrates the power of integrating large language models into a simple, user-friendly interface. It provides a foundational framework for a more comprehensive educational tool. Future improvements could include conversational capabilities, the ability to save generated content to a database, and personalized learning paths.