1. **Introduction**

This project implements an Educational AI Assistant using Gradio, Hugging Face Transformers, and PyTorch.

It allows users to:

Explain academic concepts in detail with examples.

Generate quiz questions (MCQ, True/False, Short Answer) with answers.

This assistant is designed for students, teachers, and lifelong learners who want interactive learning support.

2. **Technology Stack**

Python → Main programming language.

Gradio → Simple web-based interface.

PyTorch → Deep learning backend.

Transformers (Hugging Face) → To load and run a pre-trained Large Language Model (LLM).

Model Used: ibm-granite/granite-3.2-2b-instruct (instruction-tuned language model).

3. **Code Walkthrough**

3.1 Importing Dependencies

import gradio as gr

import torch

from transformers import AutoTokenizer, AutoModelForCausalLM

gradio: For UI.

torch: Needed for deep learning computations.

transformers: Loads the model & tokenizer.

3.2 Loading 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

)

Loads tokenizer & model from Hugging Face.

Uses GPU if available (float16 for efficiency).

Defaults to CPU (float32) otherwise.

3.3 Handling Padding Token

if tokenizer.pad\_token is None:

tokenizer.pad\_token = tokenizer.eos\_token

If no pad token is defined, uses the end-of-sequence (EOS) token.

3.4 Response Generation Function

def generate\_response(prompt, max\_length=512):

inputs = tokenizer(prompt, return\_tensors="pt", truncation=True, max\_length=51

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

How it works:

1. Converts prompt into tokens.

2. Moves input to GPU (if available).

3. Generates text using:

temperature=0.7 → balanced creativity.

do\_sample=True → adds randomness for varied outputs.

4. Decodes tokens → human-readable text.

5. Removes the prompt from the output to keep only the answer.

3.5 Concept Explanation

def concept\_explanation(concept):

prompt = f"Explain the concept of {concept} in detail with examples:"

return generate\_response(prompt, max\_length=800)

Generates detailed explanations of academic concepts.

3.6 Quiz Generator

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 separate ANSWERS section:

return generate\_response(prompt, max\_length=1000)

Creates 5 quiz questions + answers section.

Covers multiple formats: MCQ, True/False, Short Answer.

3.7 Building the 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")

transformers: Loads the model & tokenizer.

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)

Uses Gradio Blocks to build structured UI.

Two Tabs:

Concept Explanation: User enters a topic → AI generates explanation.

Quiz Generator: User enters a topic → AI generates quiz.

share=True → Creates a public shareable link

**4. Features**

✔️ Explain any academic concept with examples.

✔️ Generate quizzes with answers for self-testing.

✔️ User-friendly web UI.

✔️ Works on CPU & GPU.

✔️ Shareable public link for collaboration.

**-5. Use Cases**

Students → For studying and self-testing.

Teachers → To prepare teaching material & quizzes.

EdTech Platforms → Integrate as a learning tool.

Researchers → Quick kno

Improvements wledge

**6. Future explanations**

* Add chat-style mode using gr.Chatbot.
* Enable PDF/Word export of quizzes.
* Integrate speech-to-text & text-to-speech for accessibility.
* Improve response quality with top\_p sampling.
* Add error handling for long/invalid inputs.

**7. Conclusion**

This project demonstrates how Large Language Models (LLMs) and Gradio can be combined to create an AI-powered educational assistant.

It simplifies learning and teaching by offering:

On-demand explanations.

Auto-generated quizzes for revision.

It’s a step toward personalized, AI-driven education.

Would you like me to now convert this documentation into a formatted PDF (with headings, bullet points, and clean layout) so you can directly submit or attach it to your project?