EduTutor AI

Project Documentation

Team Leader: Jeffrey Leo A

Team Members:

1. Harikrishnan
2. Nandhakumar
3. Thanigaivel
4. Vikram

Table of Contents:

Introduction

EduTutor AI Pro is an intelligent educational assistant designed to help students and educators.

It provides detailed explanations of academic concepts and automatically generates quizzes to reinforce learning.

The system leverages AI models for natural language understanding and offers an interactive web interface powered by Gradio.

This version includes improvements in quiz diversity and response accuracy.

Team Leader: Jeffrey Leo A

Team Members:

1. Harikrishnan
2. Nandhakumar
3. Thanigaivel
4. Vikram

Project Overview

EduTutor AI Pro simplifies the learning process by combining automated explanations with practice-based assessments.

Users can enter a topic, receive AI-generated explanations, and test their understanding with dynamically generated quiz questions.

The Pro version focuses on faster execution and better accuracy of responses.

System Architecture

Frontend: Gradio-based web interface.

Backend: Python + Hugging Face Transformers.

AI Model: IBM Granite 3.2 2B Instruct.

Execution: PyTorch with optional CUDA/GPU support.

Setup Instructions 1.

Install dependencies:

```
pip install gradio torch transformers
```

2. Save the script as app.py.

3. Run the script to start the application.

Folder Structure

EduTutor-AI-Pro/

```
│-- app.py
│-- requirements.txt
│-- screenshots/
```

Running the Application Run

the application using:

python app.py

The Gradio interface will launch locally and generate a public shareable link.

API Documentation

generate_response(prompt, max_length=512): Generates AI-powered

responses. concept_explanation(concept): Explains a given concept.

quiz_generator(concept): Generates 5 quiz questions with answers.

Authentication

This version does not include authentication. Future releases may support login for personalized analytics and secure usage.

User Interface

Two tabs are available:

1. Concept Explanation: Enter a topic and get a detailed explanation.
2. Quiz Generator: Enter a topic and receive quiz questions with answers.

Testing

Manual testing was conducted with multiple topics to ensure meaningful explanations and valid quizzes.

Future work includes automated testing.

Known Issues

- Large model may run slowly on low-end systems.
- Requires internet for model loading.
- Limited quiz diversity for niche topics.

Future Enhancements -

Add voice input/output.

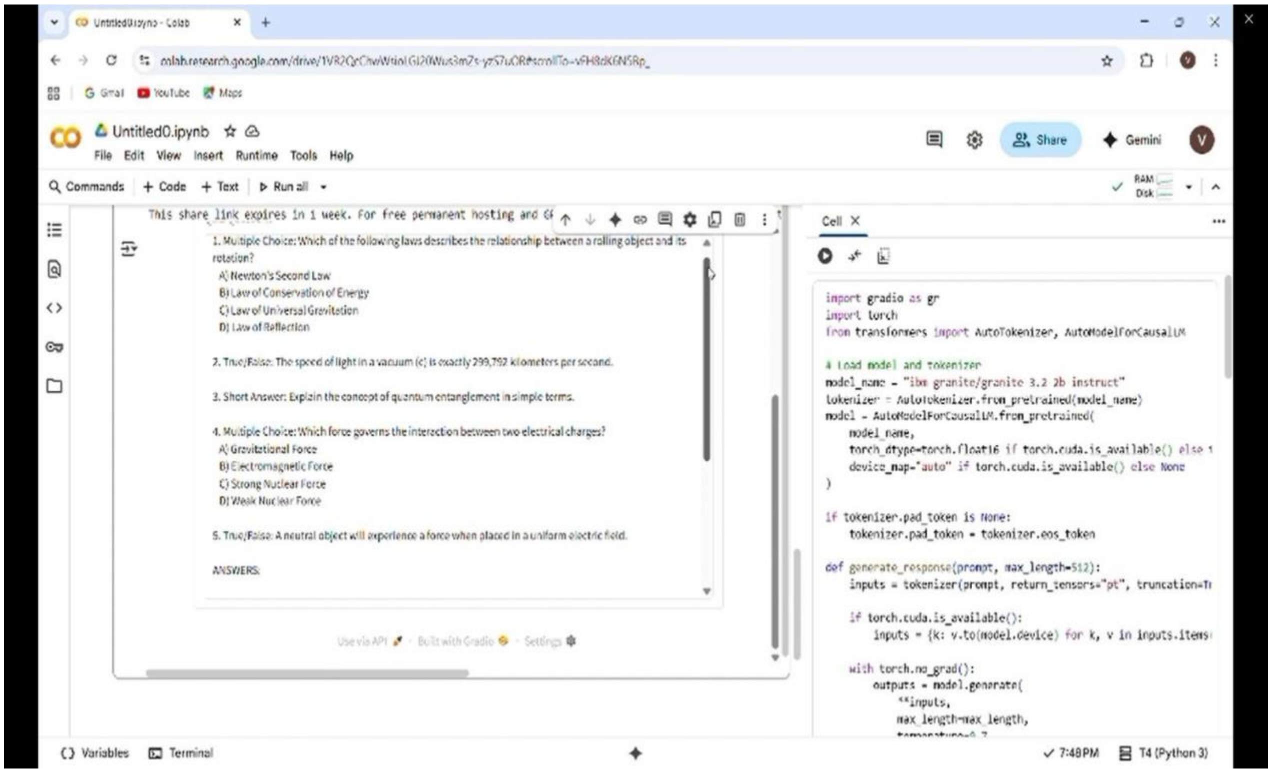
- Support multiple languages.
- Integrate with LMS platforms.
- Add quiz difficulty levels.

Program code

```
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)
    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 separate ANSWERS section:"
    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)
```

Screenshot



```
This share link expires in 1 week. For free permanent hosting and GF
```

1. Multiple Choice: Which of the following laws describes the relationship between a rolling object and its rotation?
   A) Newton's Second Law
   B) Law of Conservation of Energy
   C) Law of Universal Gravitation
   D) Law of Reflection

2. True/False: The speed of light in a vacuum (c) is exactly 299,792 kilometers per second.

3. Short Answer: Explain the concept of quantum entanglement in simple terms.

4. Multiple Choice: Which force governs the interaction between two electrical charges?
   A) Gravitational Force
   B) Electromagnetic Force
   C) Strong Nuclear Force
   D) Weak Nuclear Force

5. True/False: A neutral object will experience a force when placed in a uniform electric field.

ANSWERS:

```
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 t
    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=Tr
    
    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,
```