Edu Tutor AI: Personalized Learning

Project Documentation

1. Introduction

Project Title: Edu Tutor AI: Personalized Learning

Team Members:

N. Kiruthika

R. Kamalapriya

P. Joys Malini

A. Anisha

2. Project Overview

Purpose: Edu Tutor AI is designed to create a personalized learning environment using AI. The assistant leverages real-time data, natural language understanding, and predictive analytics to provide students and educators with tools for smarter, eco-conscious, and efficient learning support.

It can: - Summarize complex documents for easier learning. - Provide personalized study tips. - Forecast performance trends. - Detect anomalies in learning patterns. - Encourage continuous engagement through feedback loops.

Key Features: - Conversational Interface: Ask questions and get simple answers. - Policy/Lesson Summarization: Converts lengthy study content into concise summaries. - Resource Forecasting: Predicts usage of time/effort in study. - Eco-Tip Generator (Learning Tip Generator): Personalized advice for improvement. - Feedback Loop: Collects learner input to refine suggestions. - KPI Forecasting: Tracks learning performance indicators. - Anomaly Detection: Flags unusual study behaviors or performance drops. - Multimodal Input Support: Accepts PDFs, text, CSV for analysis. - Streamlit/Gradio UI: Interactive dashboard for users.

3. Architecture

Frontend (Streamlit): User dashboard with uploads, chat, feedback, reports. Backend (FastAPI): API framework for chat, tips, forecasting, embeddings. LLM Integration (IBM Watsonx Granite): For summarization, Q&A;, and learning tips. Vector Search (Pinecone): Stores and retrieves embedded study documents. ML Modules: Performance forecasting & anomaly detection using scikit-learn.

4. Setup Instructions

Prerequisites: - Python 3.9+ - pip & virtual environment - API keys (IBM Watsonx, Pinecone) - Internet access

Steps: 1. Clone the repository. 2. Install dependencies from requirements.txt. 3. Configure .env file with credentials. 4. Run FastAPI backend. 5. Launch Streamlit frontend. 6. Upload data and interact with modules.

5. Folder Structure

app/ – FastAPI backend logic app/api/ – Modular API routes (chat, feedback, reports, embeddings) ui/ – Streamlit frontend components smart_dashboard.py – Entry point for dashboard granite_llm.py – IBM Watsonx integration document_embedder.py – Document embeddings via Pinecone kpi_file_forecaster.py – Performance forecasting anomaly_file_checker.py – Anomaly detection report_generator.py – AI-generated reports

6. Running the Application

1. Start FastAPI server. 2. Run Streamlit dashboard. 3. Navigate using sidebar. 4. Upload documents/CSVs. 5. Chat with the assistant, generate reports, view predictions. 6. Interact in real-time with backend APIs.

7. API Documentation

POST /chat/ask – AI Q&A; POST /upload-doc – Upload & embed docs GET /search-docs – Search similar study content GET /get-eco-tips – Study/learning tips POST /submit-feedback – Learner feedback

8. Authentication

Open demo mode by default. Secure version supports: - JWT / API keys - OAuth2 (IBM Cloud) - Role-based access (student, teacher, admin) - Planned: session & history tracking

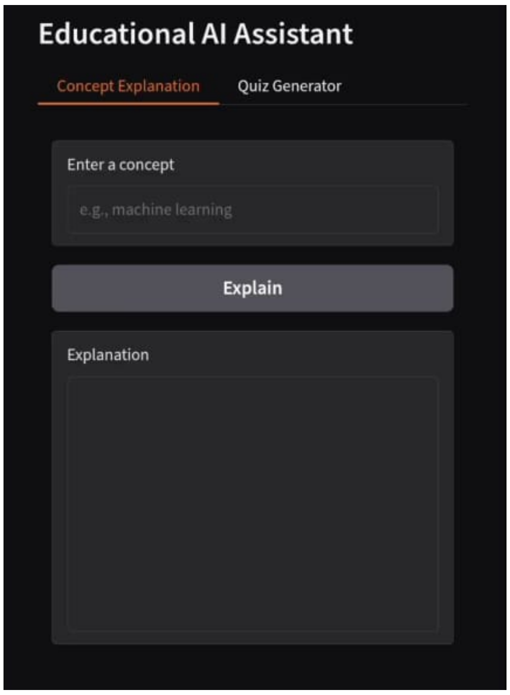
9. User Interface

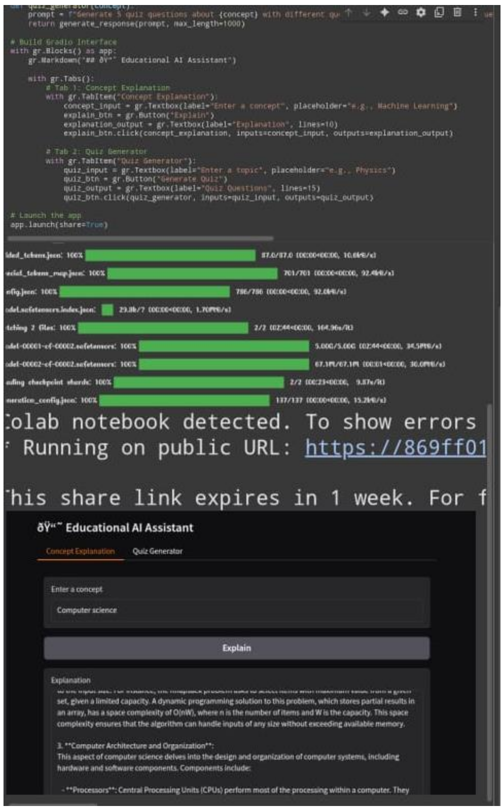
Sidebar navigation Visual learning performance dashboards Tabs for chat, tips, forecasting Real-time forms Report downloads (PDF) Clean, user-friendly design

10. Testing

Unit Testing: Prompt functions, utilities API Testing: Swagger UI, Postman Manual Testing: File uploads, chat accuracy, output consistency Edge Cases: Large files, malformed inputs, invalid keys

11. Screenshots and Demolink




```
    prompt = f"Generate 5 quiz questions about {concept} with different qu
    return generate_response(prompt, max_length=1000)

# Build Gradio Interface
with gr.Blocks() as app:
    gr.Markdown("## ðŸ"˜ Educational AI Assistant")

    with gr.Tabs():
        # Tab 1: Concept Explanation
        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)

        # Tab 2: Quiz Generator
        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)

# Launch the app
app.launch(share=True)
```

```
ided_tokens.json: 100%  87.0/87.0 [00:00<00:00, 10.6kB/s]
ecial_tokens_map.json: 100%  701/701 [00:00<00:00, 92.4kB/s]
nfig.json: 100%  786/786 [00:00<00:00, 92.0kB/s]
odel.safetensors.index.json:  29.8k/? [00:00<00:00, 1.70MB/s]
tching 2 files: 100%  2/2 [02:44<00:00, 164.96s/it]
odel-00001-of-00002.safetensors: 100%  5.00G/5.00G [02:44<00:00, 34.5MB/s]
odel-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:01<00:00, 30.0MB/s]
ading checkpoint shards: 100%  2/2 [00:23<00:00, 9.87s/it]
neration_config.json: 100%  137/137 [00:00<00:00, 15.2kB/s]
Colab notebook detected. To show errors
 Running on public URL: https://869ff01

This share link expires in 1 week. For f
```

Computer science

set, given a limited capacity. A dynamic programming solution to this problem, which stores partial results in an array, has a space complexity of O(nW), where n is the number of items and W is the capacity. This space complexity ensures that the algorithm can handle inputs of any size without exceeding available memory.

3. **Computer Architecture and Organization**:
This aspect of computer science delves into the design and organization of computer systems, including hardware and software components. Components include:

- **Processors**: Central Processing Units (CPUs) perform most of the processing within a computer. They

Demo link:
https://drive.google.com/file/d/15IkTu4zyalbKp_OrAEXnVh9Mur4-5kF9/view?usp=drivesdk

GitHub link:
https://github.com/Kiruthi718/IBM-Project.git

12. Future Enhancement

- **Adaptive Learning**: Adjusts content difficulty based on student performance.
- **Voice Interaction**: Adds speech-to-text and text-to-speech support.
- **Gamification**: Rewards, badges, and leaderboards for motivation.
- **LMS Integration**: Connect with Google Classroom, Moodle, etc.
- **Multilingual Support**: Learning in regional & global languages.
- **AI Assessment**: Automated test creation, grading, and feedback.
- **AR/VR Learning**: Immersive virtual classrooms and simulations.
- **Emotional Analytics**: Detects stress/engagement for better guidance.