

Edu Tutor AI: Personalized Learning with Generative AI and LMS Integration

Project Documentation

• Introduction

- Project title : Edu Tutor AI : Personalized Learning with Generative AI and LMS Integration
- Team member : SAI SAKTHIVEL .A
- Team member : SAI SATHIYAMOORTHY .A
- Team member : SURIYA .S
- Team member : SIVA KANDAN .V

Project overview

Title: Edu Tutor AI - Personalized Learning with Generative AI and LMS Integration

Objective: Create an AI-powered tutor that provides personalized lessons, quizzes, and doubt-solving for students,

integrated with a Learning Management System (LMS) for tracking and managing learning progress.

Problem Solved: Traditional teaching is uniform and cannot adapt to each student's pace or learning style. This system ensures customized, efficient, and trackable learning.

Key Features

- Personalized learning paths using Generative AI
- Instant doubt-solving via AI-powered chat
- Adaptive quizzes and assignments
- Multi-language supportcourse management
- LMS Integration for progress tracking and

Architecture :

1. Frontend (Streamlit/Web UI)

User interface for students to ask questions, take quizzes, and view progress.

2. Backend (Flask/Django Server)

Handles API requests, user authentication, and connects frontend to AI and LMS.

3. AI Module (Generative AI/NLP)

Processes student queries, generates personalized content, lessons, and quizzes.

4. Database / LMS

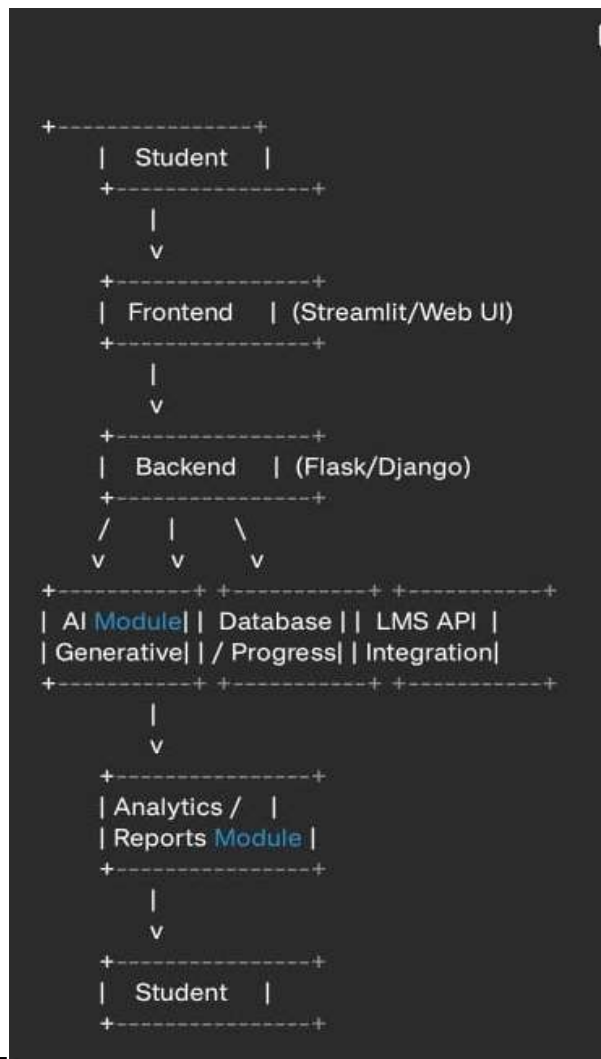
Stores student profiles, course data, quiz results, and learning progress.

5. Analytics Module

Monitors performance and generates reports for students and teachers.

Flow:

Student → Frontend → Backend → AI Module →
LMS/Database → Response → Student



Explanation:

- Students interact with the frontend.
- Requests go to the backend, which communicates with the AI module, database, and LMS.
- AI generates personalized content/quizzes.
- Progress is stored and analytics/reports are provided to both students and teachers.

Project Flow

1. Student Access

- Students log in via the frontend interface (Streamlit or Web).
- They can ask questions, view courses, or take quizzes.

2. Request Handling

- Frontend sends student queries or requests to the backend server.
- Backend manages authentication, data storage, and AI requests.

3. AI Processing

- Generative AI module analyzes the student's input.
- Generates personalized explanations, content, or quiz questions.

4. Database/LMS Interaction

- Student progress, quiz results, and learning history are stored in the LMS or database.
- Backend updates the records for tracking and reporting.

5. Response Delivery

- Backend sends AI-generated content or quiz results back to the frontend.
- Students view responses, complete quizzes, and continue learning.

6. Analytics & Feedback

- Progress and performance data are analyzed.
- Teachers and students can access reports and dashboards for insights.

Flow Diagram :

Student Frontend → Backend → AI Module →
LMS/Database → Response → Student

Milestone 1: Requirement Specification

Objective: Define what the system must do and the resources needed for development.

Requirements:

1. Functional Requirements:

- Personalized learning content generation using AI
- Instant doubt-solving for students
- Adaptive quizzes and assessments
- Progress tracking and reporting via LMS
- Multi-language support

2. Non-Functional Requirements:

- User-friendly web interface (Streamlit/Web)
- Fast and responsive AI processing
- Secure storage of student data
- Scalable architecture for multiple users

3. System Requirements:

- Python 3.8+
- Libraries: Stream ask/Django, NLP/AI libraries
- Database: MySQL / Firebase
- Internet connection for AI APIs

Outcome:

A clear understanding of what the system must achieve and the tools needed to build it.

Milestone 2: Initialization of Environment Variables

Objective: Set up the development environment and configure necessary variables for smooth project execution.

Steps:

1. Install Required Software:

- Python 3.8+, IDE (VS Code/PyCharm), Database
- (MySQL/Firebase)

2. Install Python Libraries:

Bash

- pip install streamlit flask numpy pandas nitk openai requests

3. Configure Environment Variables:

- API keys for Generative AI (e.g., Open AI API key)
- Database credentials (host, username, password, database name)
- LMS API keys or tokens
- Optional: Paths for assets , logs , or temporary files

Outcome:

A fully prepared environment where the frontend, backend, AI module, and LMS integration can run smoothly without configuration issues.

Milestone 3: AI Integration with IBM Watsonx

Objective: Integrate IBM Watsonx generative AI to provide personalized learning, doubt-solving, and adaptive quizzes.

Steps:

1. Set Up IBM Watsonx Account:
 - Create an IBM Cloud account and get API credentials for Watsonx.
2. Connect AI Module to Backend:
 - Use IBM Watsonx APIs to send student queries from the backend.
 - Receive AI-generated responses (lessons, explanations, quizzes).
3. Process AI Responses:
 - Parse responses and format them for frontend display.
 - Ensure personalized content based on student's learning history.
4. Testing & Validation:

Test AI responses for accuracy and relevance.
quality.

Outcome:

A fully functional AI module integrated with IBM Watsonx, delivering personalized learning content, instant doubt-solving, and adaptive assessments to students.

Milestone 4: Google Classroom Sync

Objective: Integrate Edu Tutor AI with Google Classroom to manage courses, assignments, and track student progress.

Steps:

1. Set Up Google Classroom API:

- Enable Google Classroom API in Google Cloud Console.
- Generate OAuth credentials for secure access.

2. Connect Backend to Google Classroom:

- Use API to fetch courses, assignments, and student data.
- Sync Edu Tutor AI quizzes and assignments with Google Classroom.

3. Update Student Progress:

- After AI-generated quizzes or lessons, automatically update grades and progress in google classroom.

4. Testing & Validation:

- Ensure all data syncs correctly between EduTutorAI and Google Classroom.
- Validate that assignments, quizzes, and scores appear accurately.

Outcome:

Seamless synchronization with Google Classroom, allowing teachers and students to track learning progress, assignments, and performance efficiently. Progress in Google Classroom.

Milestone 5: Pinecone Vector DB Integration

Objective: Integrate a Pinecone vector database to store and retrieve embeddings for personalized learning and AI responses efficiently.

Steps:

1. Set Up Pinecone Account:

- Create an account on Pinecone.io and generate API keys.

2. Prepare Data for Vectorization:

- Convert lessons, quizzes, and study materials into vector
- embeddings using NLP models.

3. Connect Backend to Pinecone:

- Store embeddings in Pinecone for fast similarity search.
- Query Pinecone during student interactions to retrieve relevant content.

4. Integrate with AI Module:

- Use Pinecone results to enhance AI-generated responses.
- Ensure personalized and context-aware learning suggestions.

5. Testing & Validation:

- Check retrieval accuracy and response relevance.
- Optimize embedding and search parameters for better performance.

Outcome:

A high-performance vector-based database system that enables Edu Tutor AI to provide context-aware, personalized content efficiently and in real-time

Milestone 6: Streamlit Frontend UI

Objective: Build an interactive and user-friendly web interface using Streamlit for students to access AI tutoring features.

Steps:

1. Set Up Streamlit:

- Install Streamlit library:

Bash

```
pip install streamlit
```

- Initialize the project frontend script (app.py).

2. Design UI Components:

- Chat interface for asking questions (st.text_input, st.button).
- Quiz interface for adaptive tests (st.radio, st.form).
- Dashboard for progress tracking (st.dataframe, st.line_chart).

3. Connect Frontend to Backend:

- Use API calls to send student inputs to backend AI module..
- Display AI responses and LMS data on the frontend.

4. Testing & Validation:

- Ensure smooth interaction and real-time updates.

- Test responsiveness and usability on different devices.

Outcome:

A fully functional, interactive Streamlit frontend where students can:

- Ask questions
- Take quizzes
- View progress dashboards
- Access personalized learning content

Milestone 7: Functional Verification

Objective: Test the entire Edu Tutor AI system to ensure all components work correctly and meet project requirements.

Steps:

1. Verify Frontend:

- Check Streamlit UI for smooth navigation, chat, quizzes, and dashboards.

2. Verify Backend & AI Module:

- Ensure student queries are correctly processed by the AI.
- Validate personalized content and quiz generation.

3. Verify LMS & Database Integration:

- Check that student progress, assignments, and quiz results are synced with Google Classroom or LMS.
- Confirm database records are correctly stored and retrievable.

4. Test Pinecone Integration:

- Verify vector search retrieves relevant lessons and study materials.

5. End-to-End Testing:

- Simulate student interactions: ask questions, take quizzes, track progress.
- Confirm AI responses, LMS updates, and analytics dashboards work seamlessly.

Outcome:

Edu Tutor AI is fully functional and reliable, providing:

Personalized learning

- Real-time doubt-solving
- Adaptive quizzes
- LMS progress tracking
- Accurate analytics

ScreenShots:

Program:

```
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

def generate_response(prompt, max_length=512):
    if tokenizer.pad_token is None:
        tokenizer.pad_token = tokenizer.eos_token

    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):"
    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)
```

Concept Explanation:

Educational AI Assistant

Concept Explanation

Quiz Generator

Enter a concept

Explain Gen AI

Exp

Explanation

short story about a robot, GPT-3 can create a narrative with characters, plot, and themes, demonstrating its ability to understand and generate human-like text.

***Explained version*:** To generate a story, GPT-3 first analyzes the input "Write a short story about a robot." It identifies key elements like "robot" and "story," then draws from its pretraining to generate a coherent narrative. The model's initial lack of understanding of human emotions, which sets the stage for an emotional journey in the story.

2. ***Image Generation*:** In the realm of visual content, Explain Gen AI uses models like DALL-E 2 (developed by OpenAI) to generate images from text prompts.

***Explained version*:** When presented with the text "A serene landscape at dusk," DALL-E 2 might generate an image of a calm ocean waves, distant mountains, and a lone seagull. The model explains its decisions by considering the visual elements associated with "serene" and "dusk." For example, it might choose warm, dusky hues to evoke serenity, and positioning the seagull at the horizon to suggest a vast, peaceful landscape.

3. ***Interpretable Decision-Making*:** Explain Gen AI models aim to provide insights into their internal decision-making processes, often through techniques like attention mechanisms and counterfactual explanations.

***Attention Mechanisms*:** Models like ViT (Vision Transformer) use attention mechanisms to highlight the most relevant parts of the input (image or text) the model focuses on. ViT provides transparency into its reasoning by showing which parts of the input are most influential. For example, when asked "What is the cat's favorite food?" and shown an image of a cat, the model might explain that it primarily considers the cat's ears, eyes, and body to make its decision.

***Counterfactual Explanations*:** These techniques help identify what changes in the input would lead to a different output. For instance, if the model predicts "cat" based on an image, counterfactual explanations might show a modified image – e.g., slightly changing the color of the cat's fur – to demonstrate how the model's decision is influenced by specific visual features.

Quiz Generator:

Educational AI Assistant

[Concept Explanation](#)[Quiz Generator](#)

Quiz Questions

- Multiple Choice:** What is the primary function of Generative Artificial Intelligence (Gen AI)?
A) Data analysis
B) Content creation
C) Decision-making
D) Coding
- True or False:** Gen AI models can learn and improve without human intervention, a concept known as self-supervised learning.
- Short Answer:** Describe a real-world application of Gen AI in generating text, such as a news article.
- Multiple Choice:** Which of the following is NOT a type of Generative Adversarial Network (GAN)?
A) DCGAN (Deep Convolutional GAN)
B) WGAN (Wasserstein GAN)
C) Flow-based GAN
D) Radial basis function GAN
- True or False:** As Gen AI continues to evolve, there are growing concerns about its potential misuse.

ANSWERS:

Output:

```

/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)
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, 695kB/s]
vocab.json: 777k/? [00:00<00:00, 30.9MB/s]
merges.txt: 442k/? [00:00<00:00, 23.4MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 84.3MB/s]
added_tokens.json: 100% [00:00<00:00, 87.0/87.0 [00:00<00:00, 8.14kB/s]
special_tokens_map.json: 100% [00:00<00:00, 701/701 [00:00<00:00, 50.9kB/s]
config.json: 100% [00:00<00:00, 786/786 [00:00<00:00, 48.8kB/s]
model.safetensors.index.json: 29.8k/? [00:00<00:00, 2.54MB/s]
Fetching 2 files: 100% [00:21<00:00, 141.84s/it]
model-00001-of-00002.safetensors: 100% [00:21<00:00, 5.00G/5.00G [00:21<00:00, 50.7MB/s]
model-00002-of-00002.safetensors: 100% [00:02<00:00, 67.1M/67.1M [00:02<00:00, 37.0MB/s]
Loading checkpoint shards: 100% [00:25<00:00, 10.58s/it]
generation_config.json: 100% [00:00<00:00, 137/137 [00:00<00:00, 10.5kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://92320020f660b93f05.gradio.live

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

```