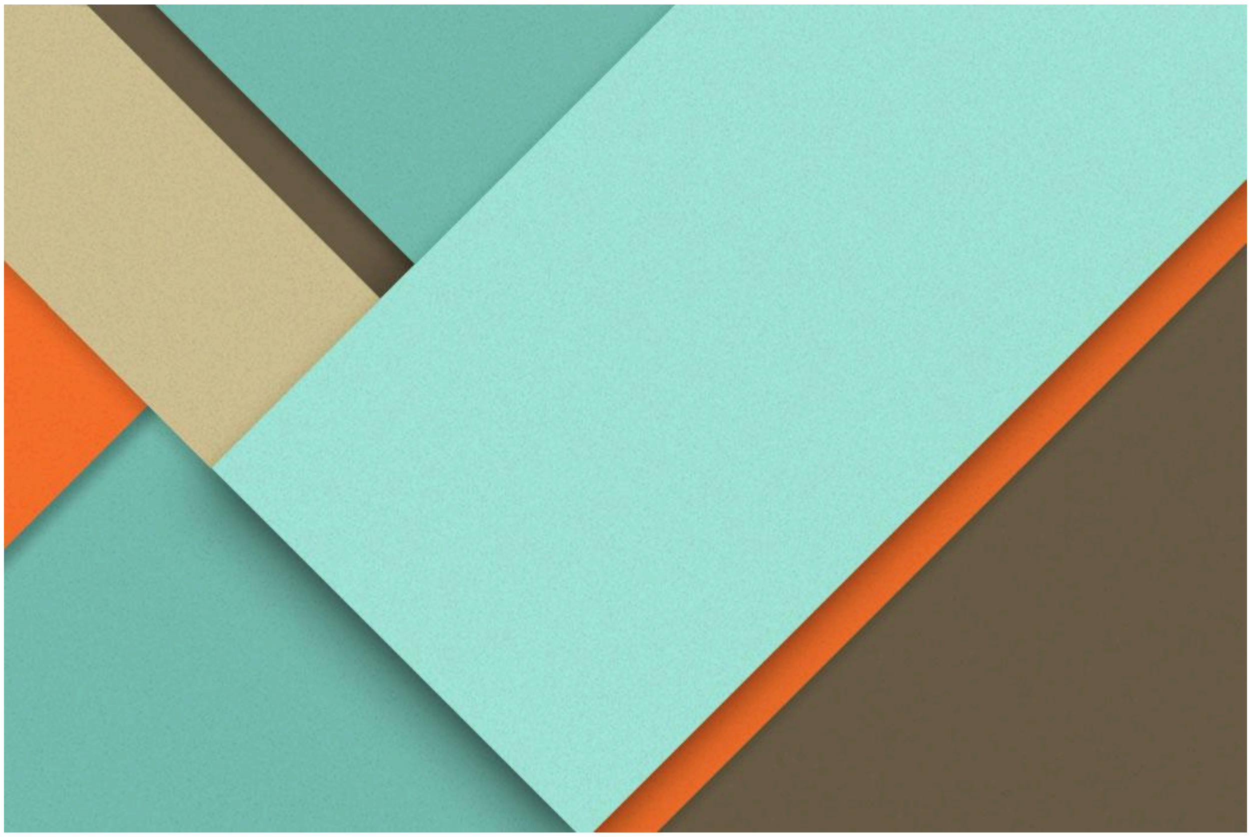


EDUTUTOR AI: PERSONALIZED LEARNING ASSISTANT

19.09.2025

EDUTUTOR AI: Personalized Learning Assistant

(Smart AI-Enhanced Learning System)

Date: 19.09.2025

---

1. INTRODUCTION

Project Title: EDUTUTOR AI: Personalized Learning Assistant

Team Leader: RAJ P

TEAM Member:VEERA MANI S

TEAM MEMBER:HARISH K

TEAM MEMBER:DHINAKAR RAJ S

TEAM MEMBER:DHAYALAN S

---

2. PROJECT OVERVIEW

Purpose:

The purpose of the EDUTUTOR AI project is to create a personalized, AI-powered learning assistant that adapts to individual student needs. By leveraging AI-driven recommendations, adaptive learning paths, and real-time feedback, EDUTUTOR AI helps learners improve efficiency, understanding, and retention.

It assists in:

Content personalization (tailored learning materials)

Adaptive quizzes & assessments

Doubt clarification via AI tutor chat

Learning progress prediction & analytics

Smart study recommendations

Ultimately, this AI-powered platform enhances engagement, reduces learning gaps, and ensures each learner progresses at their own pace.

Features

AI-Powered Content Personalization

Key Point: Customized learning experience

Functionality: Analyzes learner profiles and recommends study materials, videos, and exercises.

Adaptive Assessment & Feedback

Key Point: Continuous learning improvement

Functionality: Generates quizzes, evaluates answers, and provides instant AI-driven feedback.

Smart Doubt Solver (Chatbot Tutor)

Key Point: 24/7 learning support

Functionality: Uses LLMs to answer student queries, explain concepts, and provide examples.

Learning Path Forecasting

Key Point: Predict student performance

Functionality: AI predicts weak areas and suggests future topics to focus on.

Engagement Monitoring & Anomaly Detection

Key Point: Early intervention system

Functionality: Identifies unusual learning behavior (e.g., low activity, repeated mistakes) and alerts teachers/parents.

Study Material Summarization

Key Point: Faster knowledge absorption

Functionality: Converts lengthy textbooks or documents into concise summaries.

AI-Driven Feedback Loop

Key Point: Continuous platform improvement

Functionality: Collects student feedback, analyzes it, and refines recommendations.

---

3. ARCHITECTURE

Frontend (React / Streamlit / Gradio):

Interactive dashboards for students, teachers, and parents to view progress, insights, and recommendations.

Backend (FastAPI / Django):

Handles user authentication, learning analytics, quiz generation, and AI-driven content recommendations.

LLM Integration (OpenAI / IBM Watsonx):

Used for personalized tutoring, document summarization, and chatbot support.

Vector Database (Pinecone / FAISS):

Stores course content, student history, and knowledge embeddings for semantic search and recommendation.

ML Modules (Forecasting & Performance Prediction):

Implements models to predict student progress, weak areas, and risk of dropout.

---

4. SETUP INSTRUCTIONS

Prerequisites:

Python 3.9+

pip & virtual environment tools

API keys for AI services (Watsonx / OpenAI / Pinecone)

Internet access

Installation Process:

1. Clone the repository

2. Install dependencies from requirements.txt

3. Configure .env file with API keys

4. Run backend server with FastAPI

5. Launch frontend dashboard with Streamlit/React

6. Upload study materials and interact with AI modules

---

5. FOLDER STRUCTURE

app/ – Backend logic (APIs, models, services)

app/api/ – Routes for personalization, quizzes, analytics

ui/ – Frontend dashboards (React/Gradio/Streamlit)

ai_llm.py – AI tutor chatbot & summarization module

quiz_generator.py – Adaptive quiz creation

progress_predictor.py – Predicts student learning performance

doc_summarizer.py – Summarizes study materials

report_generator.py – Generates student progress reports

---

6. RUNNING THE APPLICATION

1. Start FastAPI backend server

2. Launch dashboard for web interface

3. Navigate via sidebar (Personalized learning, quizzes, reports)

4. Upload study content or query AI tutor

5. Receive AI-driven insights, summaries, and recommendations

---

7. API DOCUMENTATION

POST /content/personalize – Recommends study material

POST /quiz/generate – Creates adaptive quizzes

POST /chat/ask – AI tutor Q&A system

GET /progress/predict – Predicts learning outcomes

POST /feedback/submit – Collects learner feedback

---

8. AUTHENTICATION

Role-based access control (student, teacher, parent)

Secure API keys for endpoints

JWT authentication for user accounts

---

9. KNOWN ISSUES

AI accuracy may vary for domain-specific subjects

Summarization may miss context in lengthy books

Prediction models need large datasets for reliability

---

10. FUTURE ENHANCEMENTS

Integration with LMS platforms (Moodle, Google Classroom)

Support for multilingual learning

Real-time voice-based tutoring assistant

Gamification for increased student engagementEDUTUTOR AI: Personalized Learning Assistant

(Smart AI-Enhanced Learning System)

Date: 19.09.2025

---

1. INTRODUCTION

Project Title: EDUTUTOR AI: Personalized Learning Assistant

Team Leader: [Your Name]

Team Members: [Team details if needed]

---

2. PROJECT OVERVIEW

Purpose:

The purpose of the EDUTUTOR AI project is to create a personalized, AI-powered learning assistant that adapts to individual student needs. By leveraging AI-driven recommendations, adaptive learning paths, and real-time feedback, EDUTUTOR AI helps learners improve efficiency, understanding, and retention.

It assists in:

Content personalization (tailored learning materials)

Adaptive quizzes & assessments

Doubt clarification via AI tutor chat

Learning progress prediction & analytics

Smart study recommendations

Ultimately, this AI-powered platform enhances engagement, reduces learning gaps, and ensures each learner progresses at their own pace.

Features

AI-Powered Content Personalization

Key Point: Customized learning experience

Functionality: Analyzes learner profiles and recommends study materials, videos, and exercises.

Adaptive Assessment & Feedback

Key Point: Continuous learning improvement

Functionality: Generates quizzes, evaluates answers, and provides instant AI-driven feedback.

Smart Doubt Solver (Chatbot Tutor)

Key Point: 24/7 learning support

Functionality: Uses LLMs to answer student queries, explain concepts, and provide examples.

Learning Path Forecasting

Key Point: Predict student performance

Functionality: AI predicts weak areas and suggests future topics to focus on.

Engagement Monitoring & Anomaly Detection

Key Point: Early intervention system

Functionality: Identifies unusual learning behavior (e.g., low activity, repeated mistakes) and alerts teachers/parents.

Study Material Summarization

Key Point: Faster knowledge absorption

Functionality: Converts lengthy textbooks or documents into concise summaries.

AI-Driven Feedback Loop

Key Point: Continuous platform improvement

Functionality: Collects student feedback, analyzes it, and refines recommendations.

---

3. ARCHITECTURE

Frontend (React / Streamlit / Gradio):

Interactive dashboards for students, teachers, and parents to view progress, insights, and recommendations.

Backend (FastAPI / Django):

Handles user authentication, learning analytics, quiz generation, and AI-driven content recommendations.

LLM Integration (OpenAI / IBM Watsonx):

Used for personalized tutoring, document summarization, and chatbot support.

Vector Database (Pinecone / FAISS):

Stores course content, student history, and knowledge embeddings for semantic search and recommendation.

ML Modules (Forecasting & Performance Prediction):

Implements models to predict student progress, weak areas, and risk of dropout.

---

4. SETUP INSTRUCTIONS

Prerequisites:

Python 3.9+

pip & virtual environment tools

API keys for AI services (Watsonx / OpenAI / Pinecone)

Internet access

Installation Process:

1. Clone the repository

2. Install dependencies from requirements.txt

3. Configure .env file with API keys

4. Run backend server with FastAPI

5. Launch frontend dashboard with Streamlit/React

6. Upload study materials and interact with AI modules

---

5. FOLDER STRUCTURE

app/ – Backend logic (APIs, models, services)
app/api/ – Routes for personalization, quizzes, analytics
ui/ – Frontend dashboards (React/Gradio/Streamlit)
ai_llm.py – AI tutor chatbot & summarization module
quiz_generator.py – Adaptive quiz creation
progress_predictor.py – Predicts student learning performance
doc_summarizer.py – Summarizes study materials
report_generator.py – Generates student progress reports

---

6. RUNNING THE APPLICATION

1. Start FastAPI backend server

2. Launch dashboard for web interface

3. Navigate via sidebar (Personalized learning, quizzes, reports)

4. Upload study content or query AI tutor

5. Receive AI-driven insights, summaries, and recommendations

---

7. API DOCUMENTATION

POST /content/personalize – Recommends study material

POST /quiz/generate – Creates adaptive quizzes

POST /chat/ask – AI tutor Q&A system

GET /progress/predict – Predicts learning outcomes

POST /feedback/submit – Collects learner feedback

---

8. AUTHENTICATION

Role-based access control (student, teacher, parent)

Secure API keys for endpoints

JWT authentication for user accounts

---

9. KNOWN ISSUES

AI accuracy may vary for domain-specific subjects

Summarization may miss context in lengthy books

Prediction models need large datasets for reliability

---

10. FUTURE ENHANCEMENTS

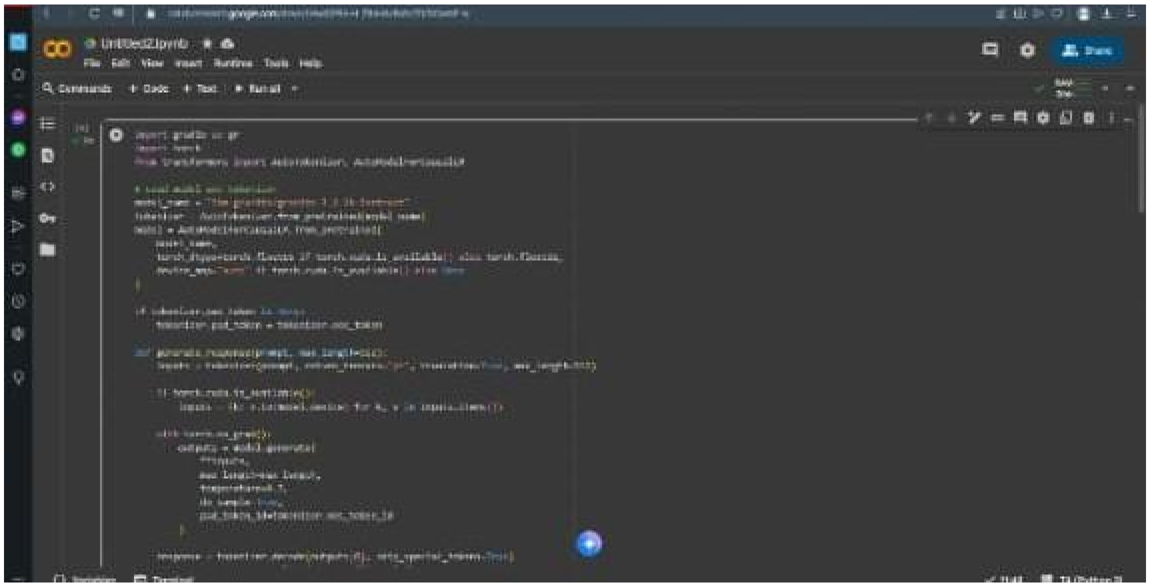
Integration with LMS platforms (Moodle, Google Classroom)

Support for multilingual learning

Real-time voice-based tutoring assistant

Gamification for increased student engagement

11.*SCREENSHOTS*


Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▸ Run all
Share

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model and tokenizer
model_name = "[illegible]"
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)
```