PROJECT REPORT ON EDUTUTOR AI:PERSONILIZED LEARNING

HuggingFaceH4/zephyr-7b-beta

TEAM MEMBERS :

S.NO	NAME	REG.NO
01	**B.Ragul**	23053161802111081
02	Saran	23053161802111086
03	Saravanan	23053161802111087
03	Santhosh rajan	23053161802111088

ANNAI VAILANKANNI ARTS AND SCIENCE COLLEGE

OWNED AND MANAGED BY"DIOCESE OF TANJORE SOCIETY"(REGD.NO.S.8 OF 1958)

(AFFILIATED TO BHARATHIDASAN UNIVERSITY,TIRUCHIRAPPALLI-24)

(AFFILIATED TO BHARATHIDASAN UNIVERSITY,TIRUCHIRAPPALLI-24)

BISHOP SUNDARAM CAMPUS,PUDUKOTTAI ROAD,THANJAVUR–613007.

Index

EduTutorAI – Project Documentation

1. Introduction

Project Title:

EduTutorAI – AI-Powered Personalized Learning

Objective:

To transform the traditional learning process using AI that adapts to each learner's unique needs.

Motivation:

Modern education is one-size-fits-all, whichoften leaves learners behind. EduTutorAI bridges this gap by making learning adaptive, interactive,and personalized

2. Project Overview

2. Project Overview

• Purpose

The purpose of **EduTutor AI: Personalized Learning** is to transform the way students learn by providing adaptive, interactive, and personalized education through AI. By leveraging large language models and learner data, the assistant delivers tailored lessons, real-time feedback, and adaptive practice exercises. It bridges the gap between traditional classroom teaching and self-paced digital learning by offering students individualized support, while also empowering educators with insights into learner progress. Ultimately, EduTutor AI fosters an inclusive learning environment that adapts to different skill levels, learning styles, and academic goals.

• Features

Conversational Learning Assistant

- **Key Point:** Interactive tutoring
- **Functionality:** Enables students to ask questions, receive explanations, and practice problems through natural language interaction.

Personalized Lesson Generator

- **Key Point:** Adaptive learning content
- **Functionality:** Creates customized lessons and examples based on the learner's knowledge level and progress.

Step-by-Step Problem Solver

- **Key Point:** Guided learning approach
- **Functionality:** Provides detailed, stepwise solutions to academic problems, with optional hints to encourage independent thinking.

Quiz & Assessment Creator

- **Key Point:** Knowledge testing

- **Functionality:** Generates multiple-choice, short-answer, or practice tests with instant feedback to reinforce learning.

Progress Tracking Dashboard

- **Key Point:** Performance insights
- **Functionality:** Monitors student progress, tracks strengths and weaknesses, and recommends next topics for study.

Adaptive Difficulty & Recommendations

- **Key Point:** Tailored learning journey
- **Functionality:** Adjusts the complexity of questions and lessons according to learner performance and engagement.

Policy Summarization for Education (Optional Future Feature)

- **Key Point:** Simplified curriculum mapping
- **Functionality:** Summarizes syllabus or policy documents into clear, student-friendly learning goals.

Multimodal Input Support

- **Key Point:** Flexible learning resources
- **Functionality:** Accepts text, documents, and problem statements (PDF/CSV) for analysis and tutoring.

Streamlit or Gradio UI

- **Key Point:** User-friendly interface
- **Functionality:** Provides an intuitive dashboard for learners and educators to interact with the assistant.

3. Architecture

Frontend (Streamlit or Gradio):

The frontend is built with **Streamlit/Gradio**, providing an interactive web-based UI where learners can access lessons, quizzes, and tutoring sessions. It includes multiple pages such as:

- **Chat interface:** for conversational tutoring.
- **Lesson viewer:** to display personalized study material.
- **Quiz & practice panel:** for assessments with instant feedback.
- **Progress dashboard:** to track learning achievements.
- **Feedback forms:** to gather student input for improving recommendations. Navigation is modular, ensuring scalability and smooth user experience.

Backend (FastAPI / Hugging Face Inference API):

The backend is powered by **FastAPI**, exposing REST endpoints for tutoring queries, quiz generation, progress tracking, and lesson retrieval. In Hugging Face Spaces, the backend integrates directly with the **Zephyr 7B model** through the **Transformers library**. The architecture supports asynchronous requests and offers easy integration with external APIs.

LLM Integration (HuggingFaceH4 Zephyr-7B-Beta)

The core of the system is the **Zephyr-7B-Beta** model from Hugging Face. It powers:

- Conversational tutoring.
- Step-by-step explanations.
- Quiz/assessment generation.

- Personalized study recommendations.
  Prompt engineering ensures responses are tailored to student age, subject level, and difficulty.

Vector Search (FAISS or Pinecone - optional future feature):

Student notes, past interactions, or uploaded documents (e.g., homework, textbooks) can be embedded using **Sentence Transformers** and stored in **FAISS/Pinecone**. This enables semantic search, allowing learners to retrieve explanations or related examples with

4. Setup Instructions

Prerequisites

- Python **3.9 or later**
- pip and virtual environment tools (venv or conda)
- Hugging Face account with access to HuggingFaceH4/zephyr-7b-beta
- (Optional) Hugging Face **API key** for hosted inference
- GPU-enabled environment (recommended for Colab or local training)
- Internet access to load models and dependencies

Installation Process

1. **Clone the repository**
   Gitclonehttps://github.com/23ucs578prog/EduTutorAI_507.gitcdEduTutorAI_507
2. **Install dependencies**

   pip install -r requirements.txt
3. **Configure environment variables**
   - Create a .env file in the project root.
   - Add your Hugging Face API key (if using hosted inference):

   HF_API_KEY=your_huggingface_api_key
4. **Run the backend (FastAPI or Colab)**

   uvicorn app:app --reload

(or run the provided Google Colab notebook for a cloud-based demo).

5. **Launch the frontend (Streamlit/Gradio)**

   streamlit run app.py

(or python app.py if Gradio is used).

6. **Interact with the modules**

- Open the local/hosted web app.
- Upload learning material (text, problems, or documents).
- Start chatting with the tutor, take quizzes, and view progress dashboards.

5. Folder Structure

```
EduTutorAI_507/
│
├── app/                # Backend logic (FastAPI / Hugging Face integration)
│   ├── api/            # Modular API routes: chat, quiz, progress, feedback
│   ├── models/         # Data models and schemas for requests/responses
│   ├── prompts/        # Prompt templates for tutoring, quizzes, explanations
│   └── utils/          # Helper functions (formatting, validation, logging)
│
├── ui/                 # Frontend components (Streamlit/Gradio)
│   ├── pages/          # Modular Streamlit pages (dashboard, lessons, quizzes)
│   ├── components/     # Reusable UI elements (cards, charts, forms)
│   └── styles/         # Custom CSS or UI configurations
│
├── tutor_dashboard.py      # Entry script for launching the main Streamlit dashboa
rd
├── zephyr_llm.py           # Handles all communication with HuggingFace Zephyr-7
B model
├── quiz_generator.py       # Creates quizzes and assessments based on user input
├── lesson_recommender.py   # Suggests lessons and practice topics using learner
profile
├── progress_tracker.py     # Tracks student performance and generates analytics
├── document_parser.py      # Parses uploaded documents/notes for tutoring integ
ration
```

```
├── requirements.txt        # Python dependencies
├── .env.example            # Example environment variables file
└── README.md               # Project documentation
```

6. Running the Application

To start the project:
➢ **Launch the FastAPI server** to expose backend endpoints for tutoring, quiz generation, and progress tracking.
➢ **Run the Streamlit (or Gradio) dashboard** to access the interactive learning interface.
➢ **Navigate through pages via the sidebar**, including chat, lessons, quizzes, and progress dashboard.
➢ **Upload documents or problem sets** (optional), interact with the AI tutor, and view outputs such as personalized lessons, step-by-step solutions, and quizzes.
➢ All interactions are handled in real-time, with the backend dynamically updating the frontend through API calls.

Frontend (Streamlit / Gradio)

The frontend is built with **Streamlit/Gradio**, offering an intuitive web interface for students and educators. Key features include:

- **Chat interface** for interactive Q&A.
- **Lesson viewer** to display AI-generated content.
- **Quiz panel** for assessments with instant feedback.
- **Progress dashboard** for visual learning analytics.
  Navigation is managed through a **sidebar menu**, with each page modularized for scalability and easy extension.

Backend (FastAPI / Hugging Face Integration)

The backend is powered by **FastAPI**, serving as the REST framework for:

- Chat interactions with the Zephyr-7B model.
- Lesson and quiz generation.
- Progress tracking and recommendation engine.
- Document parsing and embedding (future enhancement).
  It is optimized for asynchronous requests and can integrate with **Swagger UI** for easy testing and API exploration.

7. API Documentation

7. API Documentation

Backend APIs available include:

POST /chat/ask

- **Description:** Accepts a learner's query (question, problem, or topic request) and responds with an AI-generated explanation or solution.
- **Use case:** Interactive tutoring conversations.

POST /generate-quiz

- **Description:** Generates quizzes or practice questions based on a selected subject or uploaded material.
- **Use case:** Self-assessment and practice exercises.

GET /progress-report

- **Description:** Returns a student's learning progress, including topics covered, scores, and recommended next steps.
- **Use case:** Progress tracking and personalized recommendations.

POST /upload-material

- **Description:** Uploads documents, notes, or problem sets to be parsed and embedded for tutoring support.
- **Use case:** Custom study material integration.

POST /submit-feedback

- **Description:** Stores student or teacher feedback about the tutor's responses for improvement.
- **Use case:** Continuous refinement of the tutoring system.

Each endpoint is tested and documented in **Swagger UI** (or FastAPI's built-in interactive docs) for quick inspection and trial during development.

8. Authentication

Each endpoint is tested and documented in **Swagger UI** (via FastAPI) for quick inspection and trial during development.

This version of the project runs in an **open environment** for demonstration purposes. However, secure deployments can integrate:

- **Token-based authentication (JWT or API keys)**
  Ensures only authorized users or apps can access tutoring endpoints.
- **OAuth2 with Hugging Face credentials**
  Allows secure access to hosted inference APIs and model endpoints.
- **Role-based access (student, teacher, admin)**
  Teachers may upload materials and view class-level reports, while students can only access personal learning dashboards.
- **Planned enhancements**
  - Persistent **user sessions** with login/logout support.
  - **Learning history tracking** tied to user accounts for personalized progress continuity.

9. User Interface

The interface is designed to be **minimalist, intuitive, and learner-friendly**, ensuring accessibility for both students and educators. It includes:

Sidebar navigation
For quick access to chat, lessons, quizzes, progress dashboard, and feedback forms.

Progress visualizations with summary cards
Displays learner scores, completed topics, and upcoming recommendations in a clean, card-based layout.

Tabbed layouts for chat, lessons, and quizzes
Separates core functions into easy-to-navigate sections.

Real-time input handling
Student queries, answers, and uploads are processed instantly with dynamic updates.

Downloadable progress reports
Allows exporting personalized study summaries or quiz results in PDF format.

The design prioritizes **clarity, accessibility, and guidance**, with help texts, tooltips, and adaptive hints to ensure smooth learning flows.

10. Testing

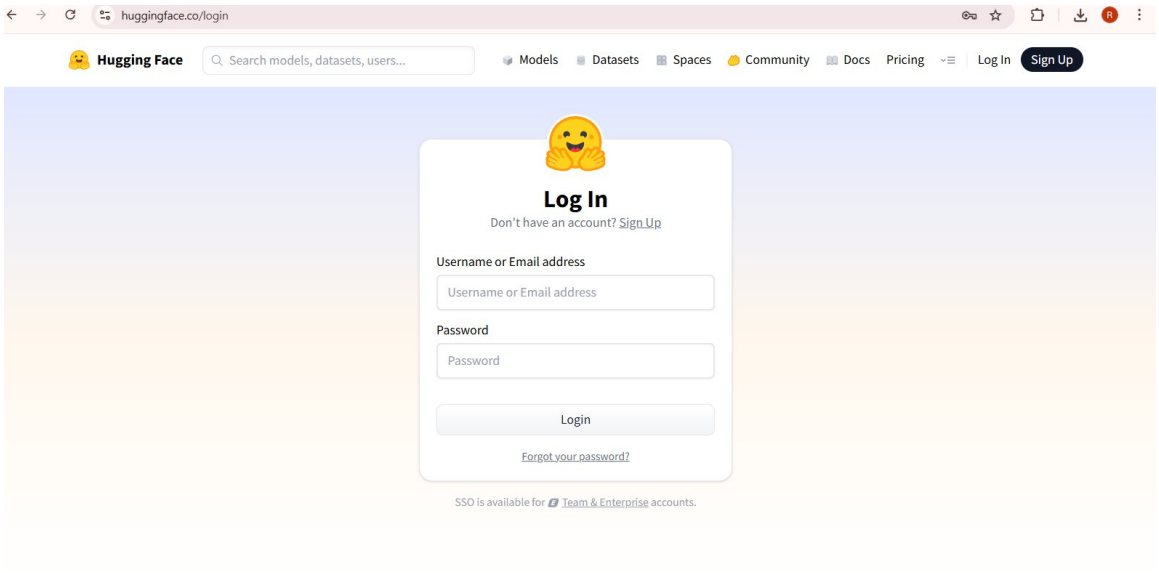
Testing was conducted in multiple phases to ensure system robustness and learning accuracy:

- **Unit Testing**
  Focused on prompt handling, quiz generation logic, and utility functions such as data preprocessing and embedding.
- **API Testing**
  Performed via **Swagger UI**, **Postman**, and automated test scripts to validate endpoints like chat, document upload, and progress tracking.
- **Manual Testing**
  Covered student–AI chat interactions, personalized content delivery, file uploads, and report downloads for consistency.
- **Edge Case Handling**
  Included malformed queries, large file uploads, invalid credentials, and ambiguous learning requests to ensure graceful fallback behavior.

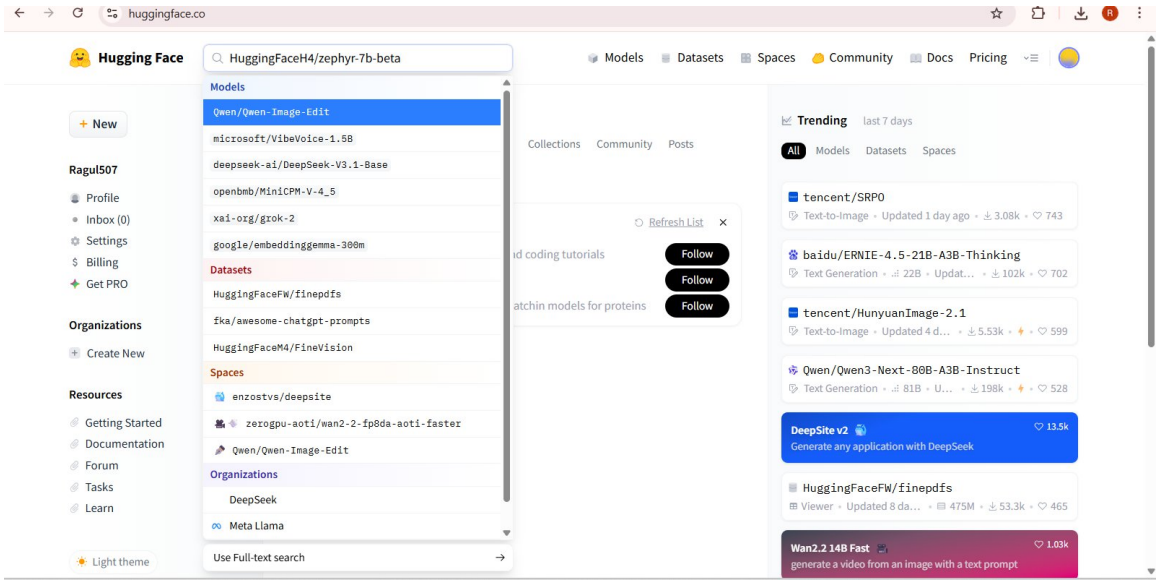
Each function and module was validated to confirm **reliability, accuracy, and performance** in both **standalone (offline)** and **cloud-connected modes**.

11. Screenshots

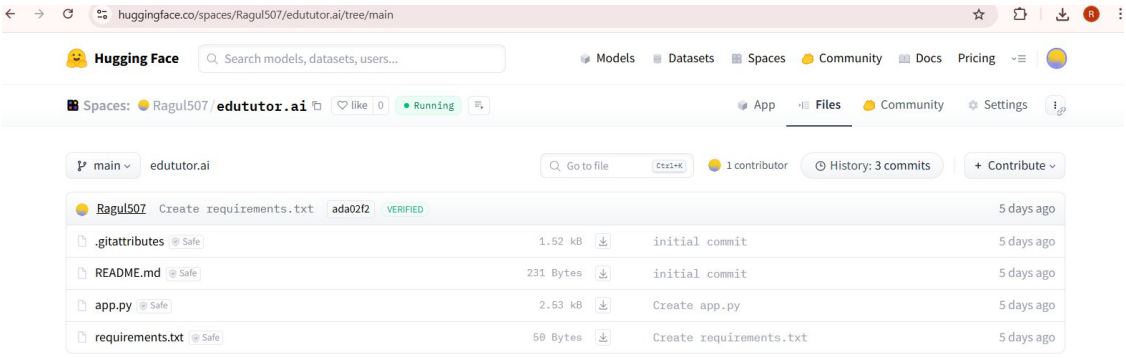
Creating account on hugging face:



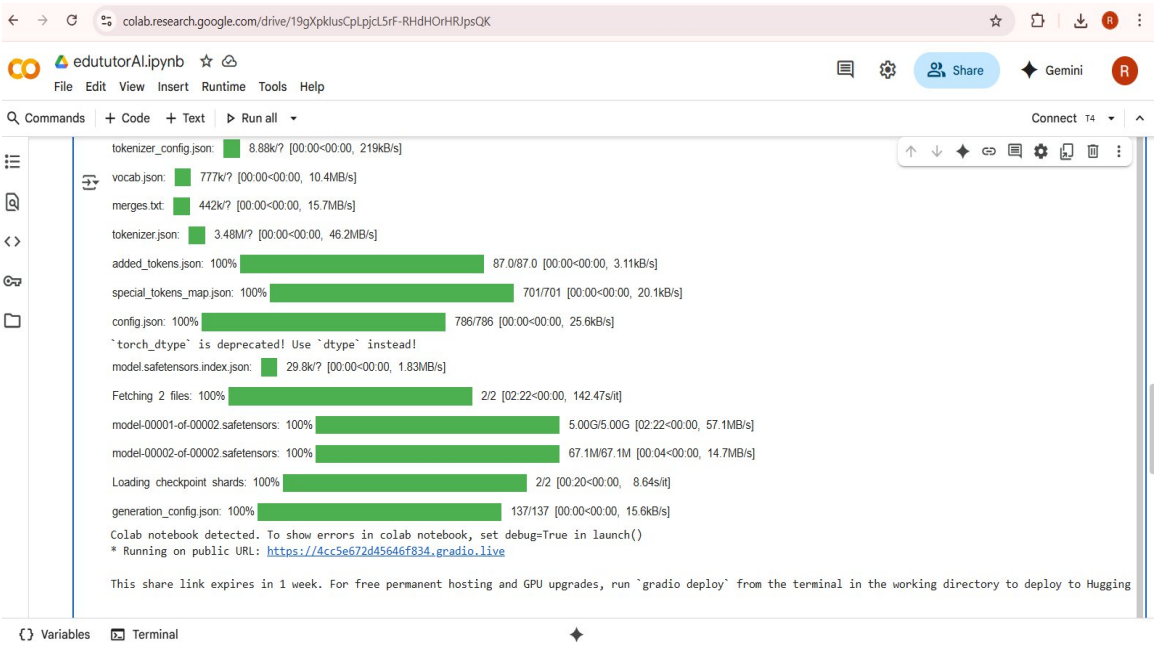
Selecting model:



Uploading code for app.py ,recquirement.txt file to new space:

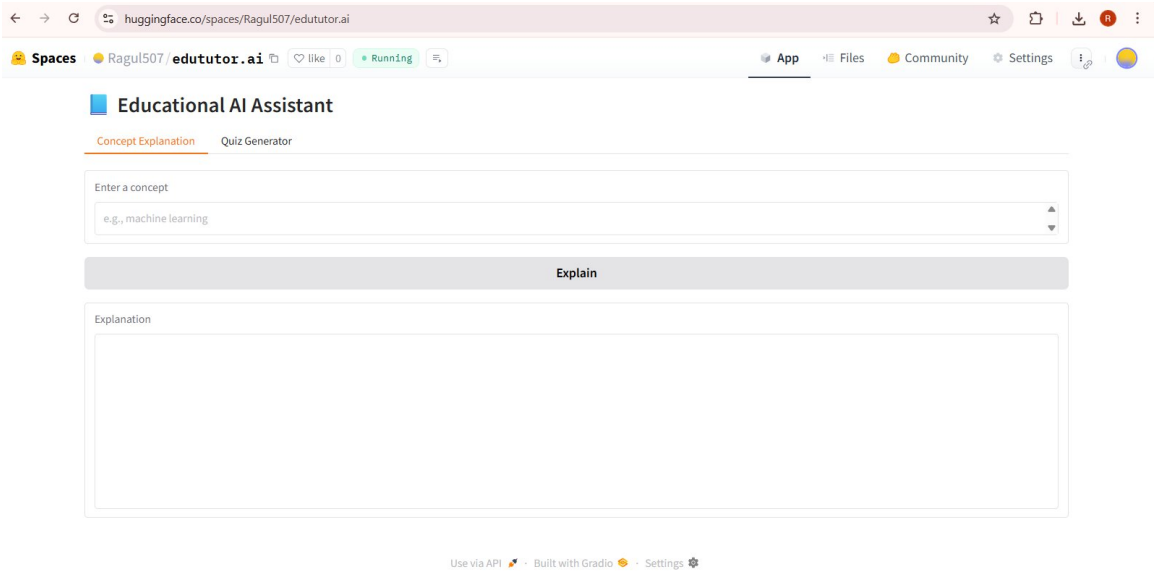


runapp.pycode in google collab and downloading healthai.ipnyb file:



:

Testing code in hugging face and the result will be displayed:



12. Known Issues

- Performance depends on API latency.
- Personalization is basic without historical learning data.
- Limited offline mode.

13. Future Enhancements

- Adaptive curriculum building.
- Multi-language tutoring.
- Voice-based interactive tutor.
- Mobile application version.
- Gamified learning experience (points, badges, levels).