Edu tutor AI-personalization learning

Project documentation

1.Introduction

- Project Title: Edu Tutor AI-Personalization Learning
- Team member: SRIJA.M
- Team member: SUJATHA.C
- Team member: SUJI.K
- Team member: SUSMITHA.R

2.Project Overview:

SmartSDLC – Edu Tutor AI-Personalization Learning is an educational platform designed to teach the Software Development Life Cycle (SDLC) using artificial intelligence to tailor instruction to each learner's needs. The project blends core software engineering practices with adaptive learning technology to create a dynamic, student-centered experience.

At its core, SmartSDLC functions as an intelligent tutor. It analyzes a learner's progress, identifies knowledge gaps, and adjusts the curriculum in real time. This means that students who quickly grasp requirement analysis, for example, can move on to design and implementation topics sooner, while those needing more support receive targeted explanations, exercises, and feedback.

3.Conversational Interface:

The conversational interface is the central interaction layer of the SmartSDLC – Edu Tutor AI-Personalization Learning platform. Instead of relying solely on static menus or traditional course navigation, this interface enables learners to communicate naturally with the system—much like chatting with a knowledgeable mentor.

4.Resource Forecasting:

Resource forecasting in SmartSDLC – Edu Tutor AI-Personalization Learning focuses on predicting and managing the people, time, and technology required to deliver both the educational platform itself and the learning projects created within it. Accurate forecasting ensures that learners, instructors, and administrators can plan effectively and avoid bottlenecks during each phase of the Software Development Life Cycle (SDLC).

5.Eco-Tip Generator:

The Eco-Tip Generator within SmartSDLC – Edu Tutor AI-Personalization Learning is designed to encourage sustainable practices while learners explore the Software Development Life Cycle (SDLC). This feature highlights the environmental impact of technology projects and offers actionable advice to reduce energy consumption, cut e-waste, and promote greener development habits.

6.Citizen Feedback Loop:

The Citizen Feedback Loop in SmartSDLC – Edu Tutor AI-Personalization Learning is designed to capture insights and suggestions from the very people who use or are affected by the platform—students, educators, and the broader community. By treating these users as “citizens” of the learning ecosystem, the feature ensures that improvements are guided by real-world experiences and evolving educational needs.

KPI Forecasting:

The KPI (Key Performance Indicator) Forecasting module in SmartSDLC – Edu Tutor AI-Personalization Learning predicts how well the platform and its learners are likely to perform over time. By combining AI-driven analytics with SDLC principles, it provides data-backed insights for strategic decision-making and continuous improvement.

Multimodal Input Support:

•The Multimodal Input Support feature of SmartSDLC – Edu Tutor AI-Personalization Learning allows learners to interact with the platform using a variety of communication methods, creating an inclusive and flexible learning environment. By enabling text, voice, image, and other input types, the system adapts to different learning styles and accessibility needs while enriching the overall experience of studying the Software Development Life Cycle (SDLC).

•The SmartSDLC – Edu Tutor AI-Personalization Learning platform can leverage modern Python-based web frameworks—Streamlit or Gradio—to create an interactive, data-driven user interface. Both options allow rapid development of AI-powered applications, but each offers unique advantages that align with SmartSDLC's goals of personalization, real-time feedback, and rich multimedia content.

7.Architecture

•Frontend (Streamlit):

The frontend of SmartSDLC – Edu Tutor AI-Personalization Learning is built with Streamlit, a Python-based framework that allows rapid development of interactive web applications without extensive JavaScript or front-end coding. Streamlit provides a clean, responsive interface that integrates seamlessly with the platform's AI-driven backend and supports real-time personalization.

Backend (FastAPI):

The backend of SmartSDLC – Edu Tutor AI-Personalization Learning is powered by FastAPI, a modern, high-performance Python web framework ideal for building RESTful and real-time APIs. FastAPI enables the platform to handle multiple AI services, data pipelines, and personalization logic efficiently while maintaining scalability and security.

LLM Integration (IBM Watsonx Granite):

•The Large Language Model (LLM) Integration layer connects SmartSDLC – Edu Tutor AI-Personalization Learning with IBM Watsonx Granite, IBM's family of enterprise-grade foundation models. This integration powers the platform's natural-language understanding, adaptive tutoring, and advanced analytics while meeting the scalability and governance requirements of educational institutions.

Vector Search (Pinecone):

The Vector Search component of SmartSDLC – Edu Tutor AI-Personalization Learning uses Pinecone, a managed vector database, to provide lightning-fast semantic search and content retrieval across all learning materials and user-generated data. By storing text, images, and other resources as high-dimensional vector embeddings, the platform can surface the most relevant information—even when a learner's query doesn't match exact keywords.

ML Modules

Forecasting and Anomaly Detection

The Machine Learning (ML) Modules in SmartSDLC – Edu Tutor AI-Personalization Learning provide advanced analytics that predict future trends and identify unusual patterns across learner activity, system performance, and educational outcomes. These modules enhance decision-making for administrators, instructors, and students by delivering actionable insights in real time.

8.Setup Instructions

•Prerequisites

Before deploying SmartSDLC – Edu Tutor AI-Personalization Learning, ensure that the following hardware, software, and service requirements are met. These prerequisites prepare your environment for a smooth installation and integration of all platform components.

1. System Requirements

•Operating System: Linux (Ubuntu 20.04+ recommended) or macOS; Windows 10/11 supported for development.

•Processor: Quad-core CPU (Intel i5/Ryzen 5 or better).

•Memory: Minimum 8 GB RAM (16 GB+ recommended for production).

•Storage: At least 20 GB free disk space for application code, and temporary data.

•GPU (Optional): NVIDIA GPU with CUDA support for accelerated ML training or inference.

9.Folder Structure :

frontend/ – Houses the Streamlit UI, including pages, reusable

components, and static assets for the interactive learning interface.

backend/ – Contains the FastAPI application with API routes, database models, services for LLM integration, Pinecone vector search, and ML modules for forecasting and anomaly detection.

Configs/ – Centralized configuration files and sample .env templates to manage environment variables securely.

tests/ – Unit and integration tests for both frontend and backend to ensure code reliability and maintainability.

docker/ – Dockerfiles and docker-compose.yml for easy containerized deployment across different environments.

scripts/ – Utility scripts for database migrations, data seeding, and maintenance tasks.

10.Running the Application :

•Activate Virtual Environment: Ensure Python dependencies are isolated using venv or conda.

•Start Backend (FastAPI): Launch the API server to handle AI models, database operations, and user requests. Access API docs at http://localhost:8000/docs.

•Launch Frontend (Streamlit): Run the Streamlit interface for interactive SDLC learning at http://localhost:8501.

•Verify Integrations: Test IBM Watsonx Granite LLM and Pinecone vector search to confirm connectivity and functionality.

•Access Application: Open a browser, log in, and start exploring AI-personalized modules, forecasting insights, and interactive exercises.

Frontend (Streamlit):

The frontend of SmartSDLC – Edu Tutor AI-Personalization Learning is built using Streamlit, providing a responsive, interactive, and user-friendly interface for learners to access AI-personalized SDLC modules. Streamlit enables rapid development of web applications with minimal code while supporting rich multimedia content and real-time data visualization.

Backend (FastAPI):

The backend of SmartSDLC – Edu Tutor AI-Personalization Learning is built using FastAPI, a high-performance Python framework designed for building RESTful APIs and WebSocket services. It powers all the platform's core functionalities, including AI model integration, data management, and real-time personalization.

11.API Documentation :

Interactive Swagger UI: Access at http://localhost:8000/docs to explore and test all API endpoints directly from the browser.

Redoc Documentation: Provides a detailed, searchable reference at http://localhost:8000/redoc.

Endpoint Categories:

User Management (registration, login, profiles)

SDLC Modules (content access, quizzes, progress tracking)

AI Services (conversational AI, eco-tip generation, personalized recommendations)

Vector Search (semantic retrieval via Pinecone)

Analytics & Forecasting (KPI predictions, resource planning, anomaly detection)

Citizen Feedback (submission and retrieval of feedback)

12.Authentication

The Authentication system in SmartSDLC – Edu Tutor AI-Personalization Learning ensures secure access to the platform while enabling role-based personalization for learners, instructors, and administrators. It protects sensitive data, maintains user privacy, and supports seamless interaction with AI-powered services.

Key Features

User Registration & Login:

Learners and instructors can create accounts with email verification.

Login supports secure password hashing and optional multi-factor authentication (MFA).

JWT (JSON Web Tokens):

Tokens are issued upon successful login to authenticate subsequent API requests.

Tokens have configurable expiration times to balance security and user convenience.

Role-Based Access Control (RBAC):

Defines different permission levels for learners, mentors, and admins.

Ensures that users can only access resources and functionalities appropriate for their role.

User Interface

The User Interface (UI) of SmartSDLC – Edu Tutor AI-Personalization Learning is designed to provide an intuitive, interactive, and engaging learning environment. Built with Streamlit, it focuses on ease of navigation, real-time feedback, and seamless integration with AI-powered features, enabling learners to explore the Software Development Life Cycle (SDLC) effectively.

Key Features

Dashboard Overview:

Displays personalized learning progress, upcoming tasks, KPI insights, and eco-tip highlights.

Quick visual summaries of completed modules and predicted performance trends.

Sidebar Navigation:

Provides easy access to SDLC phases, AI tools (Conversational Interface, Eco-Tip Generator), analytics dashboards, and feedback sections.

Interactive Learning Components:

Quizzes, coding exercises, and flowchart builders to reinforce SDLC concepts.

Supports multimodal input, including text, voice, and image uploads.

Testing

The Testing phase in SmartSDLC – Edu Tutor AI-Personalization Learning ensures that all components of the platform—frontend, backend, AI modules, and integrations—function correctly, reliably, and securely. A robust testing strategy helps identify and resolve bugs early, ensures accurate AI-driven personalization, and maintains a smooth user experience.

Unit Testing:

•Tests individual functions and modules in isolation, including backend API endpoints, ML model functions, and frontend components.

•Frameworks: pytest for Python, with coverage reports to track test completeness.

Integration Testing:

•Ensures that frontend, backend, AI models (IBM Watsonx Granite), and vector search (Pinecone) work together seamlessly.

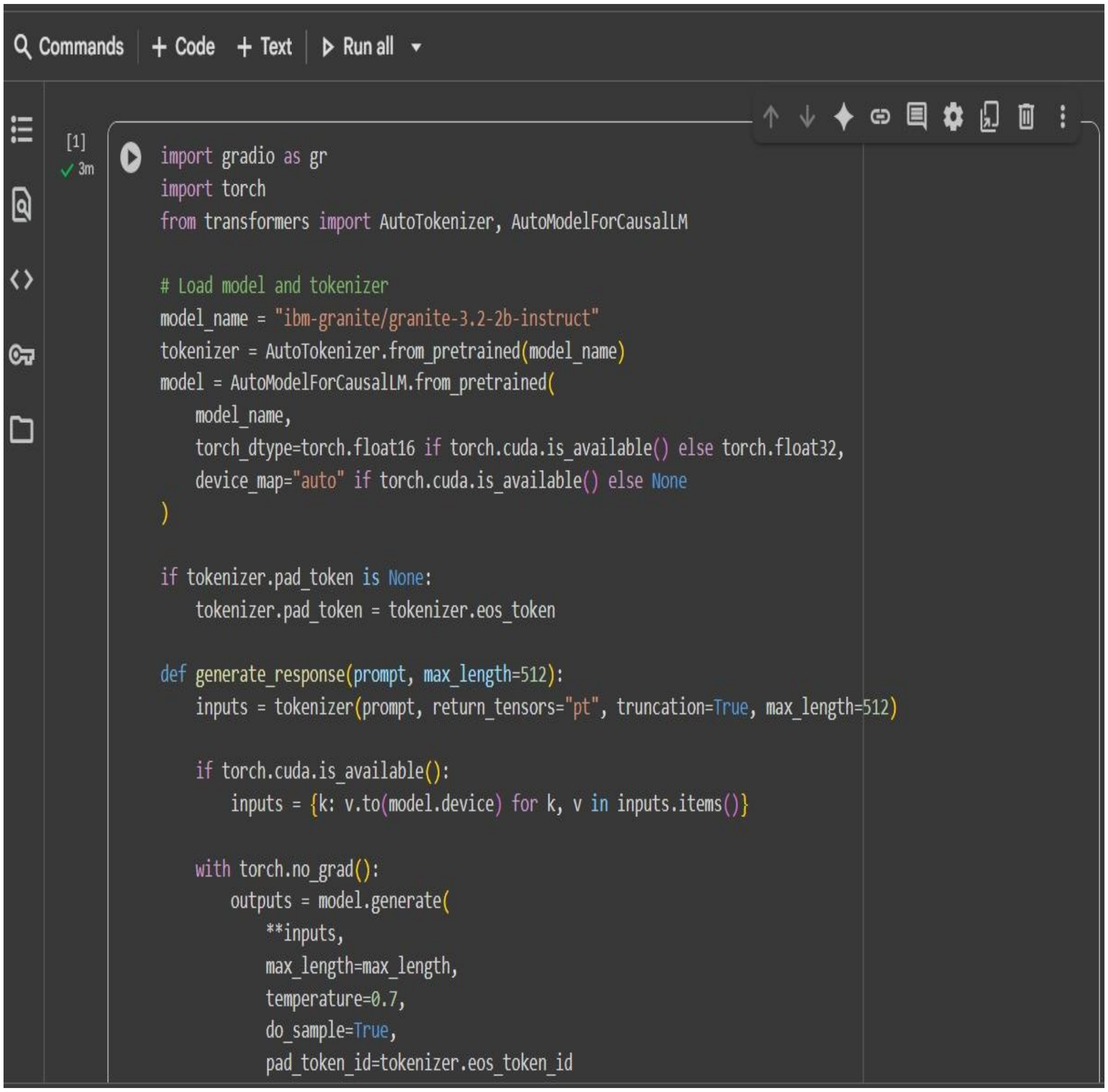
•Validates end-to-end workflows, including user interactions, data flow, and response accuracy.

Functional Testing:

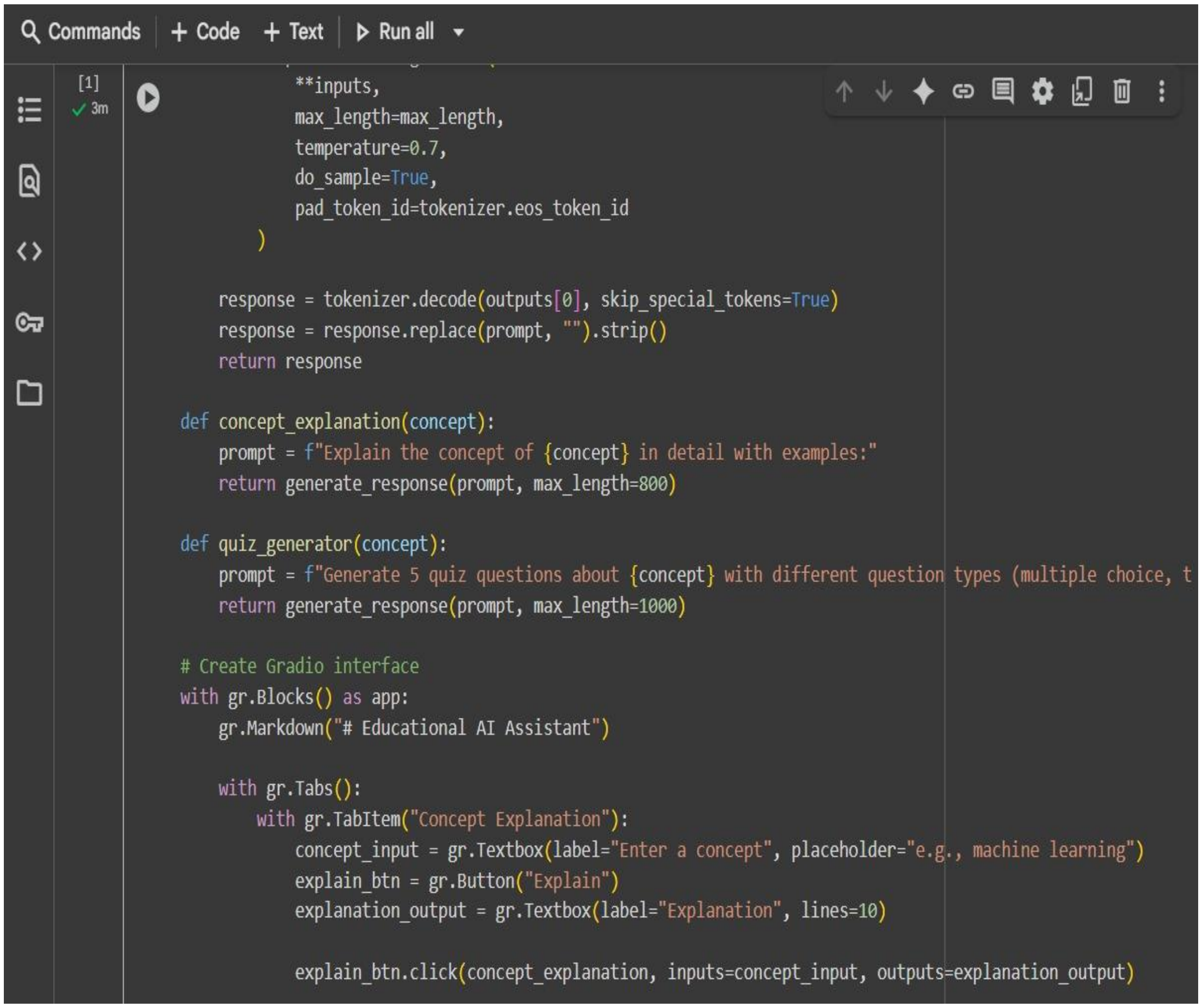
•Verifies that all features, such as SDLC module access, quizzes, forecasting, anomaly detection, and conversational AI, behave as expected.

•Includes testing of authentication, role-based access, and session

13.screenshot

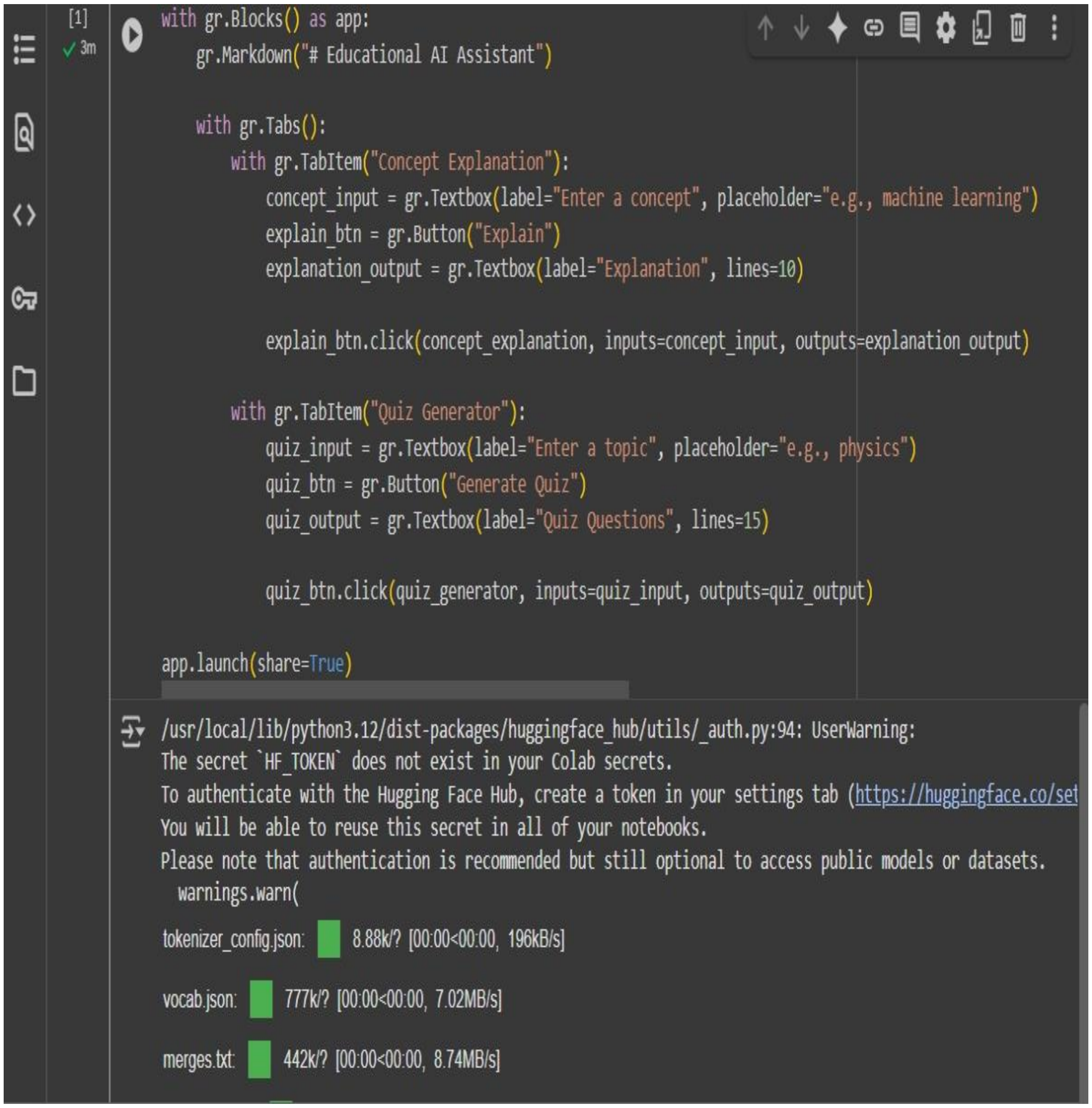


```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 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
```



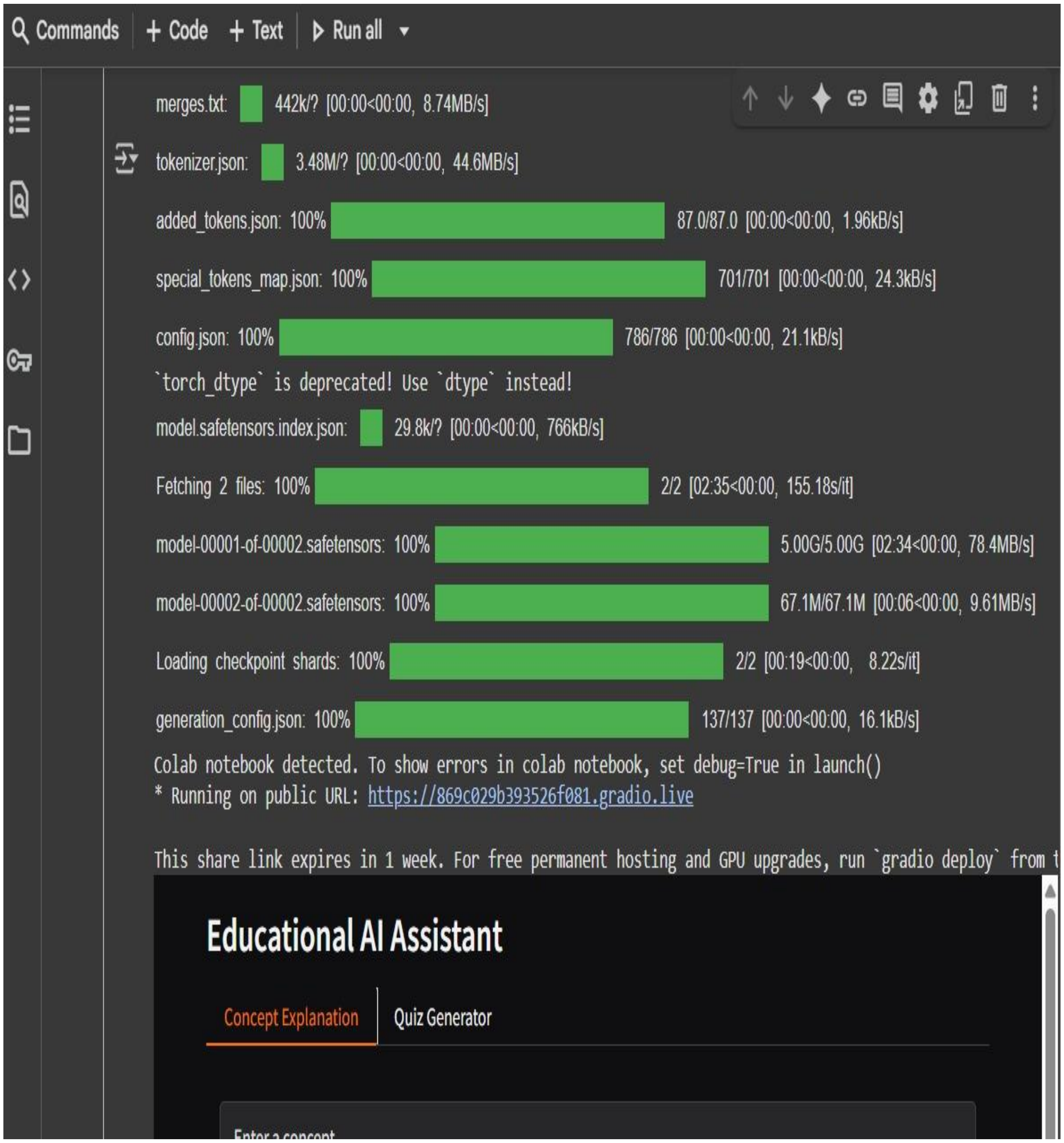
```
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    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, t
    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)
```

L



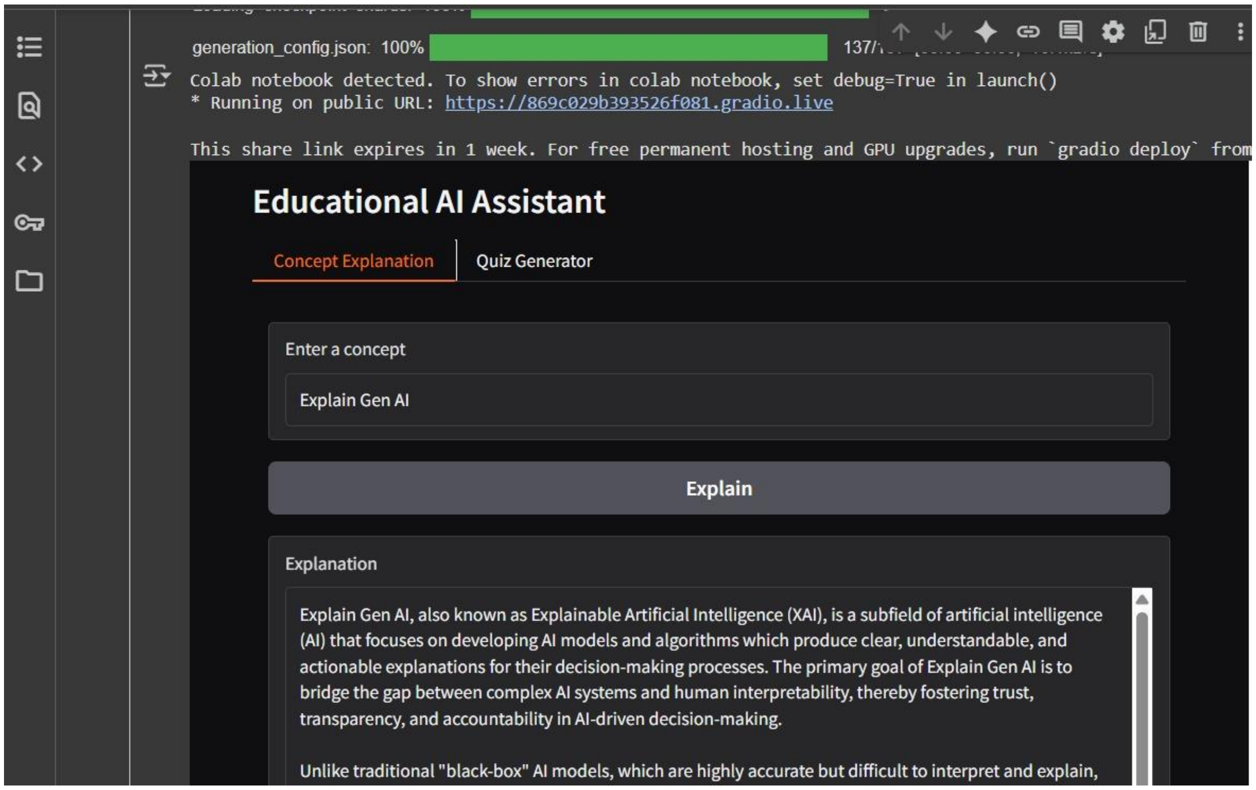
```
[1]
✓ 3m
with gr.Blocks() as app:
    gr.Markdown("# Educational AI Assistant")

    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)
```

```
/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/set
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, 196kB/s]
vocab.json: 777k/? [00:00<00:00, 7.02MB/s]
merges.txt: 442k/? [00:00<00:00, 8.74MB/s]
```



```
merges.txt:  442k/? [00:00<00:00, 8.74MB/s]
tokenizer.json:  3.48M/? [00:00<00:00, 44.6MB/s]
added_tokens.json: 100%  87.0/87.0 [00:00<00:00, 1.96kB/s]
special_tokens_map.json: 100%  701/701 [00:00<00:00, 24.3kB/s]
config.json: 100%  786/786 [00:00<00:00, 21.1kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json:  29.8k/? [00:00<00:00, 766kB/s]
Fetching 2 files: 100%  2/2 [02:35<00:00, 155.18s/it]
model-00001-of-00002.safetensors: 100%  5.00G/5.00G [02:34<00:00, 78.4MB/s]
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:06<00:00, 9.61MB/s]
Loading checkpoint shards: 100%  2/2 [00:19<00:00, 8.22s/it]
generation_config.json: 100%  137/137 [00:00<00:00, 16.1kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://869c029b393526f081.gradio.live

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

Educational AI Assistant



```
generation_config.json: 100% 137/1
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://869c029b393526f081.gradio.live

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

Educational AI Assistant

Explain Gen AI

Explain Gen AI, also known as Explainable Artificial Intelligence (XAI), is a subfield of artificial intelligence (AI) that focuses on developing AI models and algorithms which produce clear, understandable, and actionable explanations for their decision-making processes. The primary goal of Explain Gen AI is to bridge the gap between complex AI systems and human interpretability, thereby fostering trust, transparency, and accountability in AI-driven decision-making.

Unlike traditional "black-box" AI models, which are highly accurate but difficult to interpret and explain,

14. Known Issues

Latency for long quiz generation.

Limited to text-based input (no speech yet).

English-only support.

15. Future Enhancements

Voice-enabled tutoring.

Multilingual quiz generation.

Real-time collaborative classrooms.

Offline LLM inference for low-resource environments.