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

Project Documentation

- **Introduction**
  - Project title : Edu Tutor AI : Personalized Learning with Genarative 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 Al - Personalized Learning with Generative Al and LMS Integration

Objective: Create an Al-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 Al
- Instant doubt-solving via Al-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 Al and LMS.

3. Al Module (Generative AI/NLP)

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

4. Database / LMS

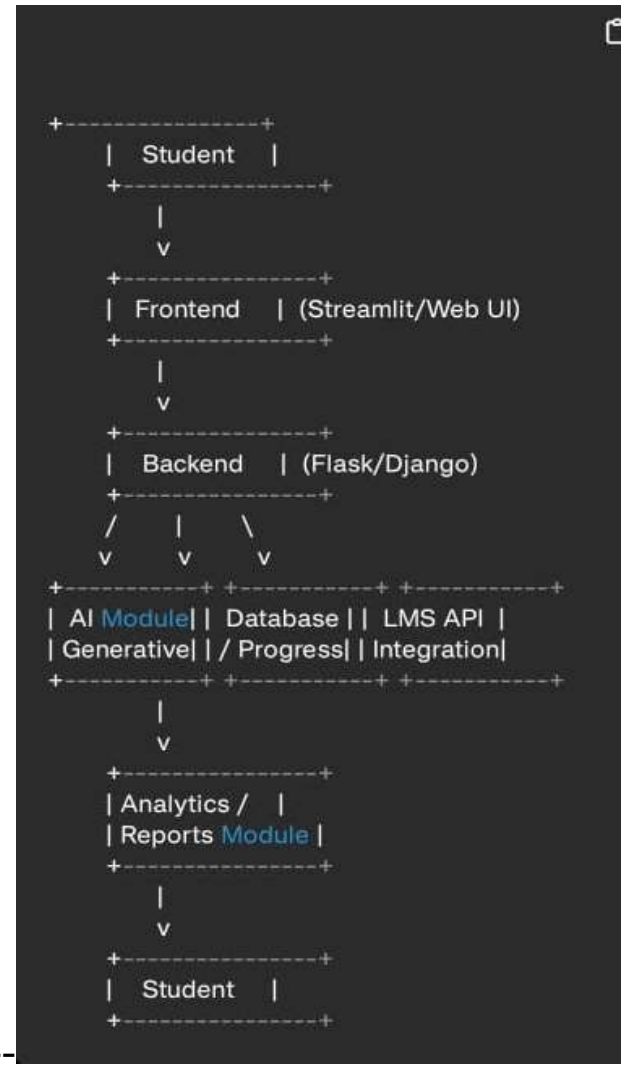
Stores student profiles, course data, quiz results, and learning progress.

5. Analytics Module

Monitors performance and generates reports for students and teachers.

Flow:

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



Explanation:

- Students interact with the frontend.
- Requests go to the backend, which communicates with the Al module, database, and LMS.
- Al 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 Al requests.
3. Al Processing
   - Generative Al 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 Al-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 → Al 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 Al
    - 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 Al 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 Al 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 Al (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, Al module, and LMS integration can run smoothly without configuration issues.

Milestone 3: Al Integration with IBM Watsonx

Objective: Integrate IBM Watsonx generative Al 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 Al Module to Backend:
   - Use IBM Watsonx APIs to send student queries from the backend.
   - Receive Al-generated responses (lessons, explanations, quizzes).
3. Process Al Responses:
   - Parse responses and format them for frontend display.
   - Ensure personalized content based on student's learning history.
4. Testing & Validation:

   Test Al responses for accuracy and relevance.

   quality.

Outcome:

A fully functional Al 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 Al 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 Al quizzes and assignments with Google Classroom.

3. Update Student Progress:
   - After Al-generated quizzes or lessons, automatically update grades and progress in google classroom.

4.Testing & Validation:

- Ensure all data syncs correctly between Edu TutorAl 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.rogress in Google Classroom.

Milestone 5: Pinecone Vector DB Integration

Objective: Integrate a Pinecone vector database to store and retrieve embeddings for personalized learning and Al 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 Al Module:

- Use Pinecone results to enhance Al-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 Al 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 Al 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 Al module..
   - Display Al 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 Al system to ensure all components work correctly and meet project requirements.

Steps:

1. Verify Frontend:
    - Check Streamlit Ul for smooth navigation, chat, quizzes, and dashboards.
2. Verify Backend & Al Module:
    - Ensure student queries are correctly processed by the Al.
    - 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 Al responses, LMS updates, and analytics dashboards work seamlessly.

Outcome:

Edu Tutor Al is fully functional and reliable, providing:

Personalized learning

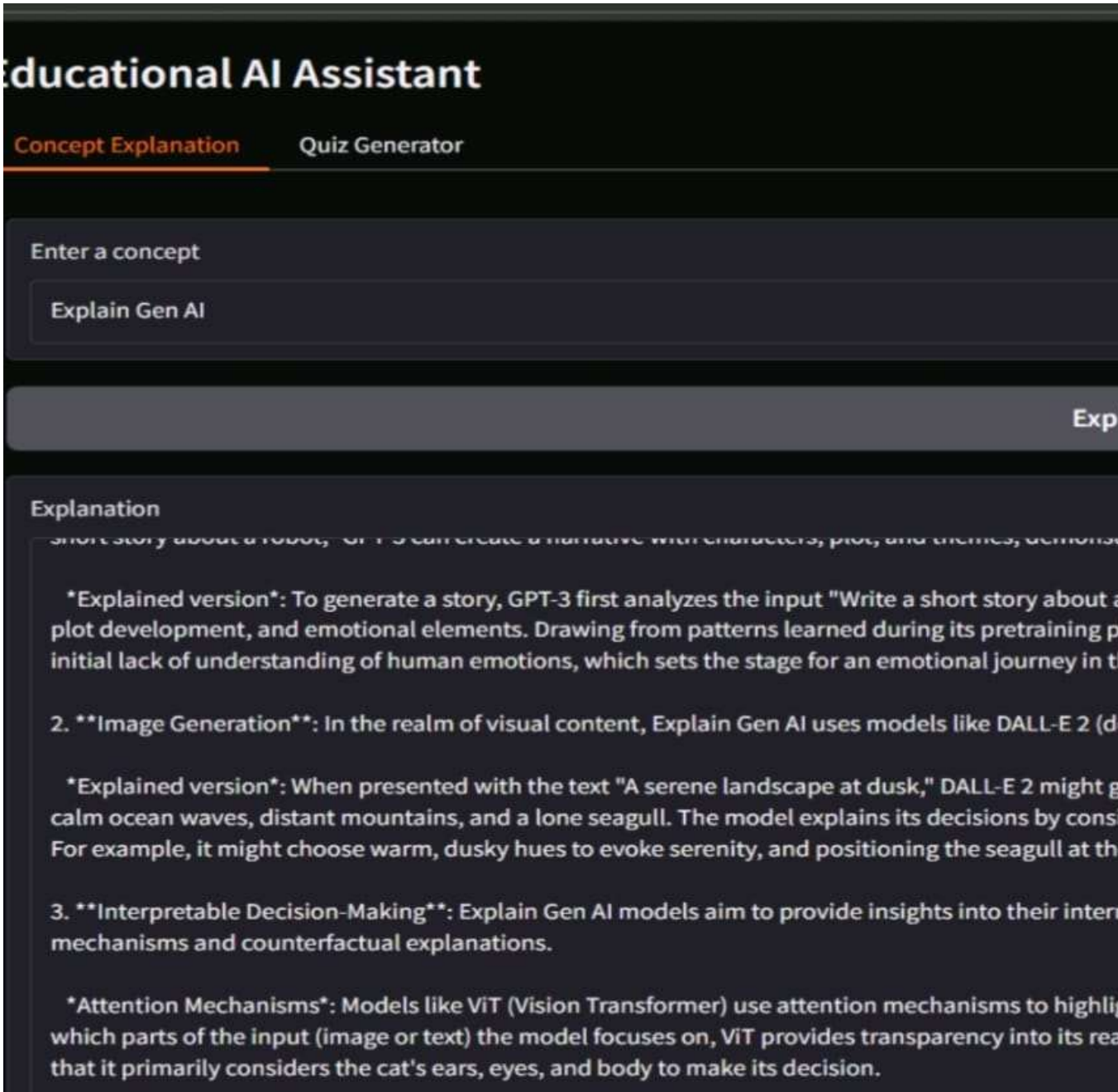
- Real-time doubt-solving
- Adaptive quizzes
- LMS progress tracking
- Accurate analytics

ScreenShots:

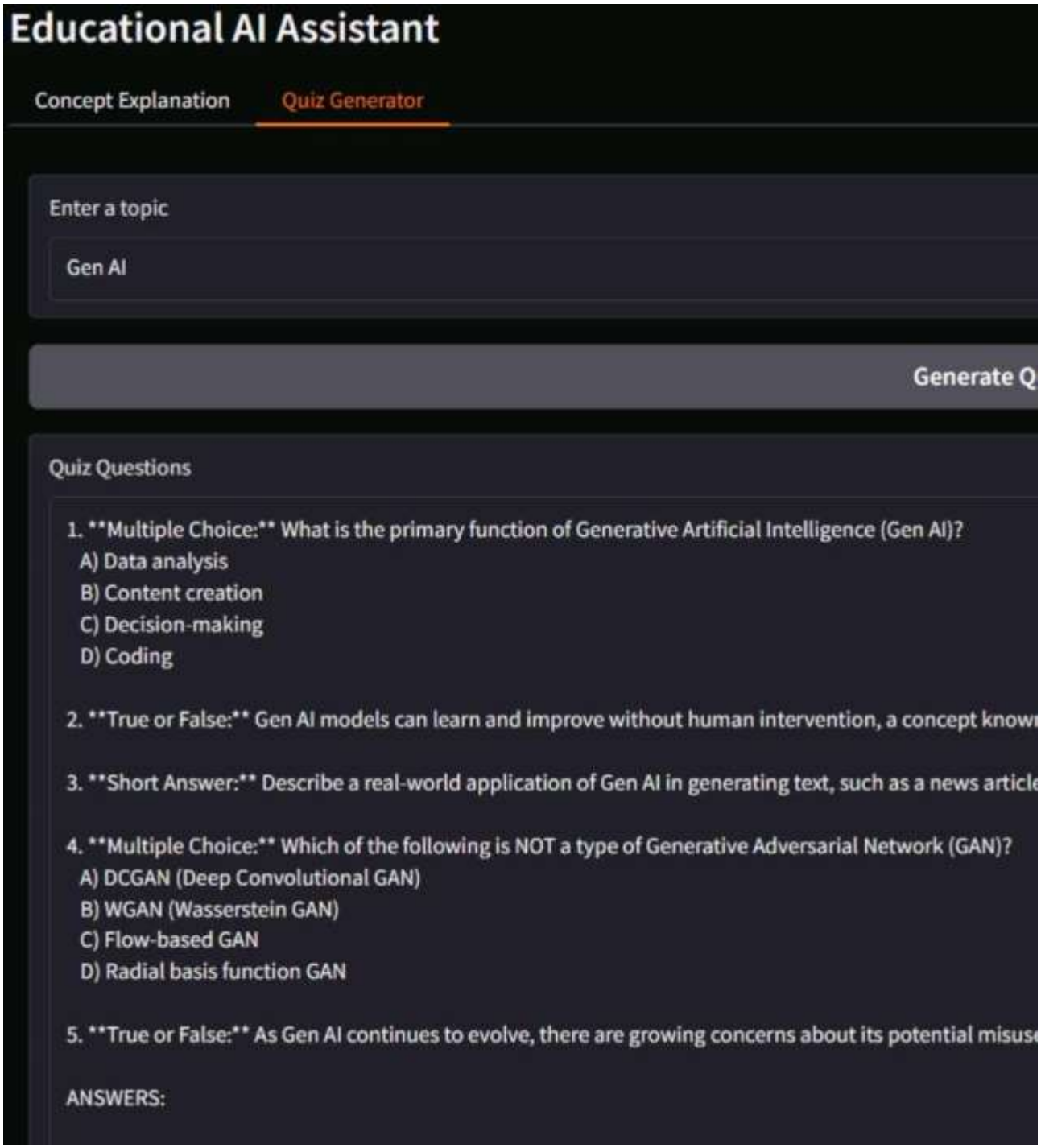
Program:

```
mport gradio as gr
mport torch
rom transformers import AutoTokenizer, AutoModelForCausalLM

 Load model and tokenizer
odel_name = "ibm-granite/granite-3.2-2b-instruct"
okenizer = AutoTokenizer.from_pretrained(model_name)
odel = 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


f tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

ef 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)
    response = response.replace(prompt, "").strip()
    return response

ef concept_explanation(concept):
    prompt = f"Explain the concept of {concept} in detail with examples:"
    return generate_response(prompt, max_length=800)

ef quiz_generator(concept):
    prompt = f"Generate 5 quiz questions about {concept} with different question types (multiple choice, true
    return generate_response(prompt, max_length=1000)

 Create Gradio interface
ith 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)

pp.launch(share=True)
```

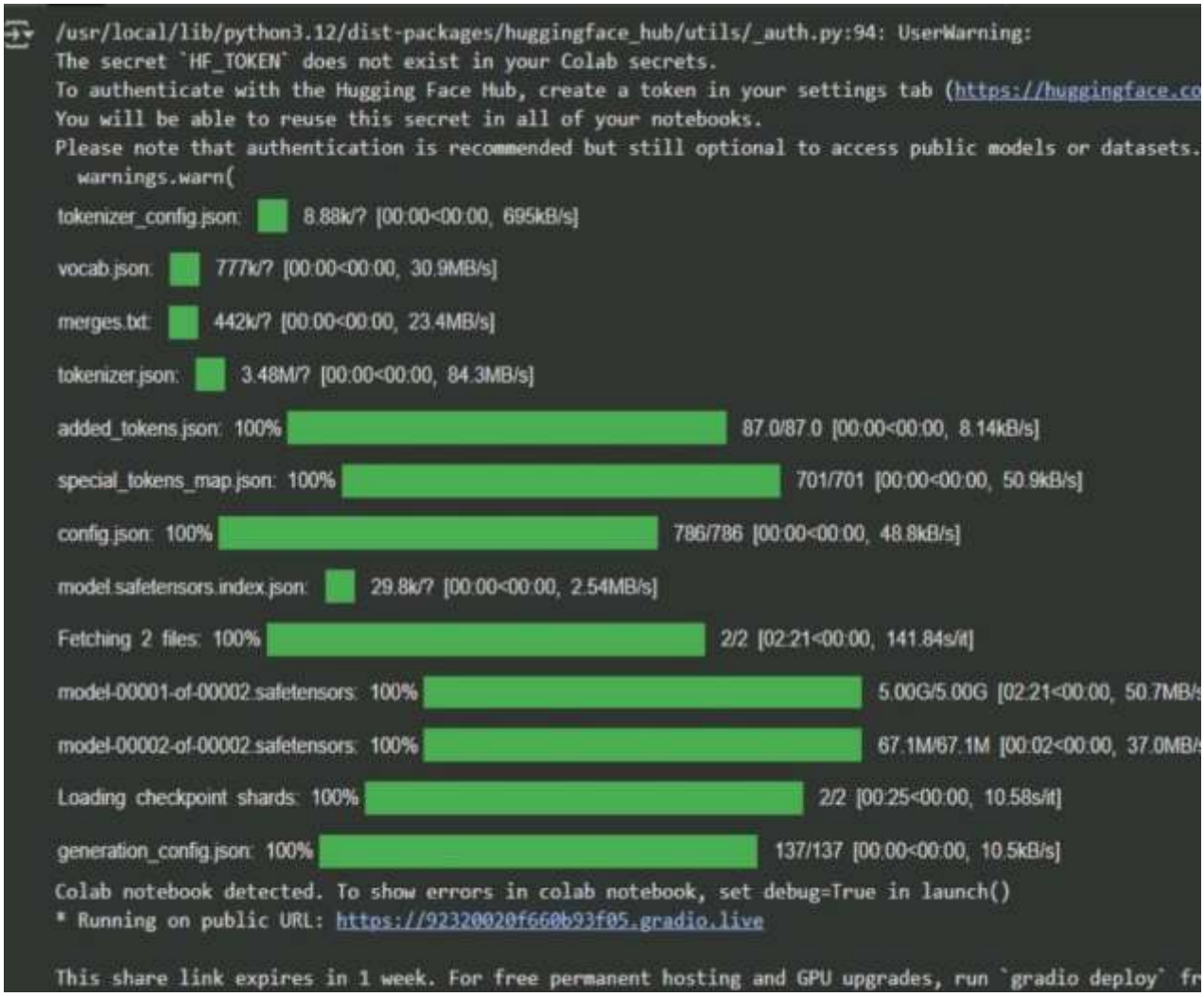
Concept Explanation:



Quiz Generator:



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
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% 87.0/87.0 [00:00<00:00, 8.14kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 50.9kB/s]
config.json: 100% 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% 2/2 [02:21<00:00, 141.84s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [02:21<00:00, 50.7MB/
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:02<00:00, 37.0MB/
Loading checkpoint shards: 100% 2/2 [00:25<00:00, 10.58s/it]
generation_config.json: 100% 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