EduTutor AI: Personalized Learning with Generative AI and LMS Integration

Leader: SIVANESAN.M

Team Member: SIVARANJAN.S

Team Member: SRIRAM.P

Team Member: SRIHARISH.K

1. Introduction

EduTutor AI is an AI-powered personalized education platform that revolutionizes the way students
learn and educators assess progress. It provides dynamic quiz generation, student evaluation,
Google Classroom integration, and real-time feedback—all powered by IBM Watsonx and Granite
foundation models.

2. Project Overview

Purpose: The purpose of EduTutor AI is to create a personalized and adaptive learning environment that helps students improve their skills through tailored quizzes, smart feedback, and
progress tracking. It empowers educators with AI-driven insights to evaluate performance and
recommend learning paths.

Features:
- Dynamic Quiz Generation
- Student Performance Evaluation
- Google Classroom Integration
- AI-Powered Feedback
- Real-time Progress Tracking
- Multi-modal Input Support
- User-Friendly Dashboard Interface

3. Architecture

Frontend: Built with React/Streamlit for an interactive and user-friendly UI.
Backend: Implemented with FastAPI to handle quiz generation, evaluation, and feedback APIs.
LLM Integration: IBM Watsonx Granite models for text generation, evaluation, and recommendations.
Database: MySQL for storing user details, results, and learning data.
Integration: Google Classroom API for seamless learning management system connectivity.

4. Setup Instructions

Prerequisites:
- Python 3.9 or later
- FastAPI and required libraries
- IBM Watsonx API key
- MySQL server setup

Installation Steps:
1. Clone the repository
2. Install dependencies from requirements.txt
3. Configure database in .env file
4. Run FastAPI backend server
5. Launch frontend dashboard
6. Start interacting with EduTutor AI modules

5. Folder Structure

app/ – Backend APIs and business logic
ui/ – Frontend code and dashboards
models/ – Machine learning and LLM integration
database/ – MySQL configurations
docs/ – Documentation and reports
main.py – Entry point to run the backend server

6. Running the Application

➢Start FastAPI server
➢Launch React/Streamlit dashboard
➢Navigate through the quiz, performance, and feedback pages
➢Upload student data, generate quizzes, and review AI feedback

7. API Documentation

Available APIs:
- POST /quiz/generate – Generates quizzes dynamically
- POST /evaluate – Evaluates student answers
- GET /progress – Retrieves student progress reports
- POST /feedback – Provides AI-generated learning feedback
- GET /classroom/sync – Integrates with Google Classroom

8. Authentication

EduTutor AI supports:
- JWT-based authentication
- Role-based access control (Admin, Educator, Student)
- Secure API key for third-party integrations

9. User Interface

The UI is minimal and interactive, providing:
- Sidebar navigation for modules
- Quiz interface
- Student progress charts
- AI-generated feedback panel
- Report download option

10. Testing

Testing Phases:
- Unit Testing (Quiz generation, evaluation logic)
- API Testing (via Swagger, Postman)
- Manual Testing (student flows, quiz attempts)
- Edge Cases (incorrect inputs, invalid logins)
- Integration Testing (Google Classroom sync)

12. Known Issues

- Limited offline support
- Dependency on Watsonx cloud availability
- Basic UI theme (can be enhanced)

13. Future Enhancements

- Add voice-based quiz interaction
- Mobile app version

- Multi-language support
- Advanced analytics dashboards
- Offline learning mode

14. Screenshots & Outputs

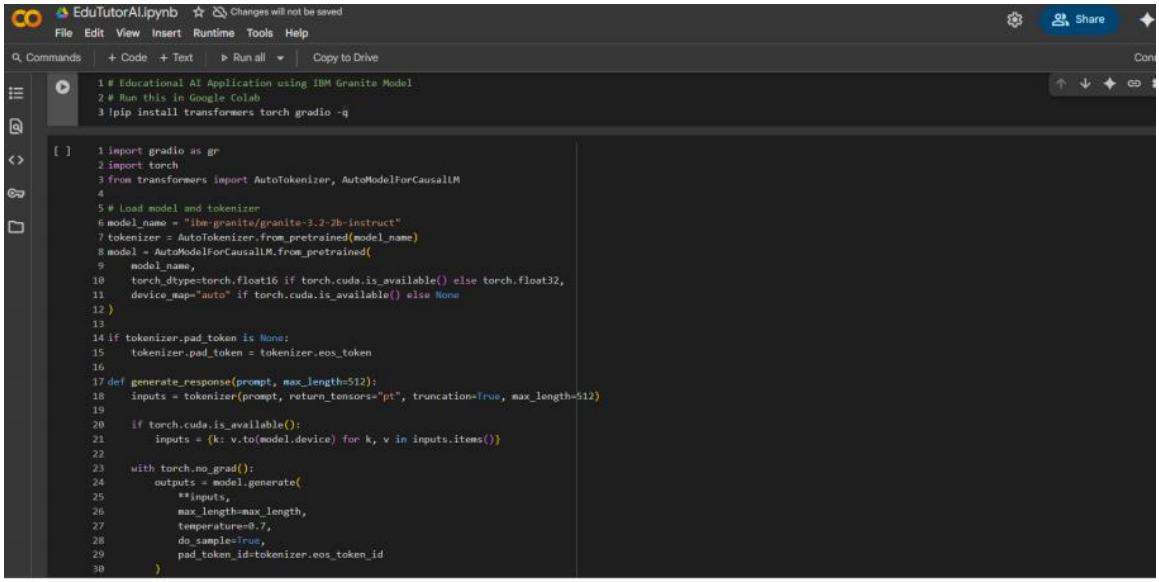


Figure 1: Google Colab - Model Import and Setup

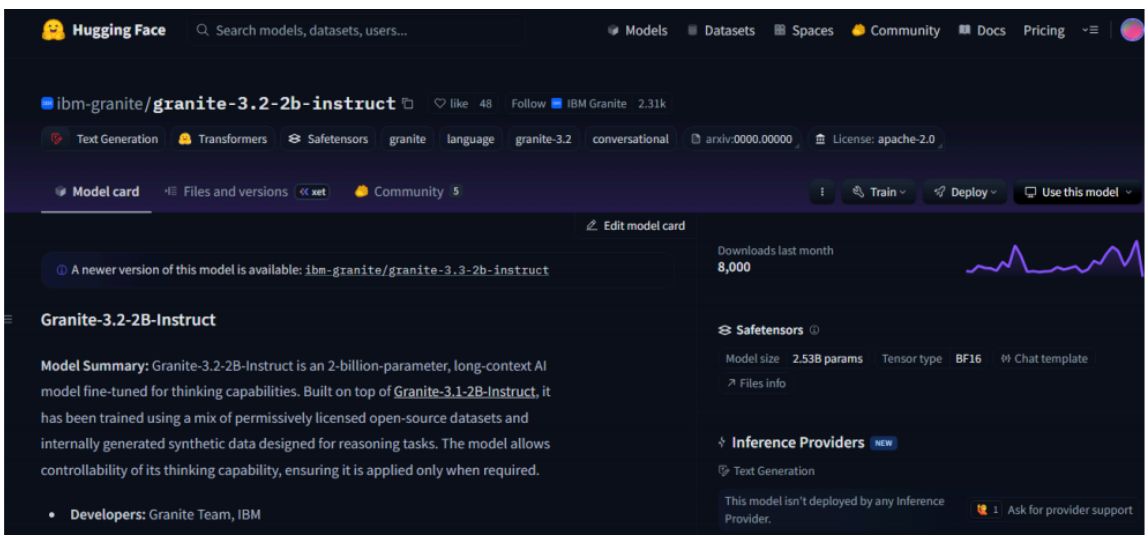
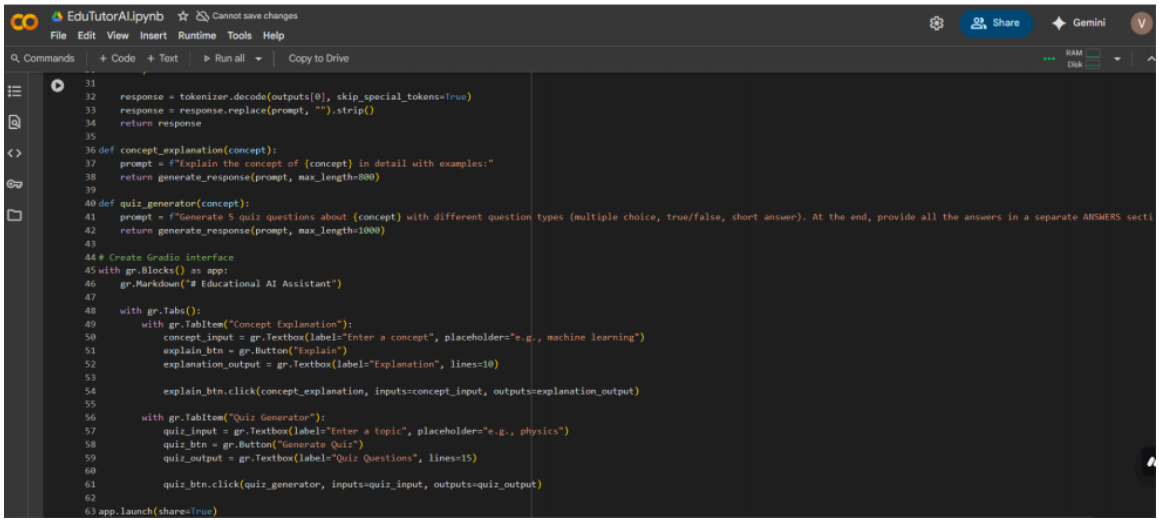


Figure 2: Hugging Face - IBM Granite Model Page



```python

    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). At the end, provide all the answers in a separate ANSWERS secti
    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)
```

Figure 3: Google Colab - Application Code with Gradio UI