Educational AI Assistant

Personalized Learning with Generative AI

PROJECT DOCUMENTATION

Team Members:

1. ABEEB MOHAMMED KASIM.M – Concept Explanation Module
2. GUGAN.D – Quiz Generator and Attender Module
3. GURU BHAVESH.H – Testing & Compiling
4. Viswanathan.M– Uploading In Github

Project Overview

Education is transforming through Artificial Intelligence integration. The Educational AI Assistant harnesses Generative AI to provide in-depth topic explanations, dynamic quizzes, immediate feedback, and performance scoring. Utilizing the Granite-3.2 model for content generation, PyTorch and Transformers for inference, and Gradio for the interface, this platform enables personalized learning for students across academic levels, fostering engagement and knowledge retention.

Objectives

1. To create a personalized learning platform powered by Generative AI.
2. To generate detailed explanations for educational concepts or topics.
3. To provide dynamic multiple-choice quizzes with real-time feedback and scoring.
4. To develop an interactive web interface for seamless user engagement.
5. To ensure clean, structured output from AI-generated content.
6. To deploy the application easily via a web browser or cloud environment.

System Architecture

The architecture consists of four main modules:

- Concept Explanation Module – Users input topics to receive detailed explanations.
- Quiz Generator and Attender Module – Generates quizzes, accepts answers, and provides feedback with scoring.
- Backend AI & Interface – Employs Granite-3.2 for generation, integrated with Gradio UI.
- Integration & Deployment – Manages model loading via Hugging Face, PyTorch inference, and Gradio deployment.

Workflow:

User starts the app → Chooses a tab (Explanation or Quiz) → Enters a topic → AI produces content → User engages (views or answers) → Instant feedback/scores shown.

Module Description

1. Concept Explanation Module
    - User inputs educational topic
    - AI delivers in-depth explanations with examples
    - Output displayed readably
2. Quiz Generator and Attender Module
    - User enters quiz topic
    - Generates 5 MCQs with A, B, C, D options
    - Submits answers for instant feedback, scoring, and details
3. Backend AI & Interface
    - Links to Granite-3.2 for responses
    - Manages prompts for explanations/quizzes
    - Handles Gradio state for answers/interactions
4. Integration & Deployment
    - Incorporates PyTorch/Transformers for inference
    - Deploys via Gradio

- Ensures module integration

Tools & Technologies Used

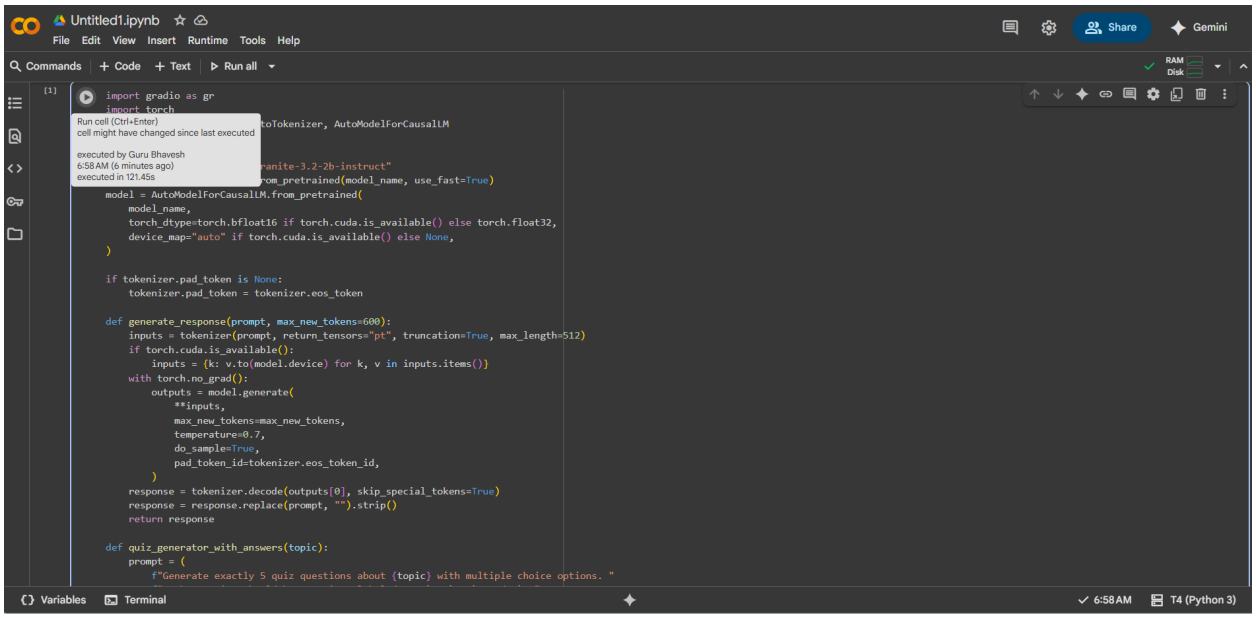
- Python – Programming Language
- Gradio – Web Application Framework
- Granite-3.2 Model – Generative AI for explanations and quizzes
- PyTorch – Model inference and transformations
- Transformers (Hugging Face) – Model loading/interaction
- GitHub – Version Control

Implementation Steps

1. Set up environment with Python and libraries.
2. Connect Granite-3.2 via Hugging Face.
3. Build explanation interface for input/display.
4. Develop quiz generator with answer handling.
5. Add feedback/scoring for quizzes.
6. Implement Gradio UI with tabs.

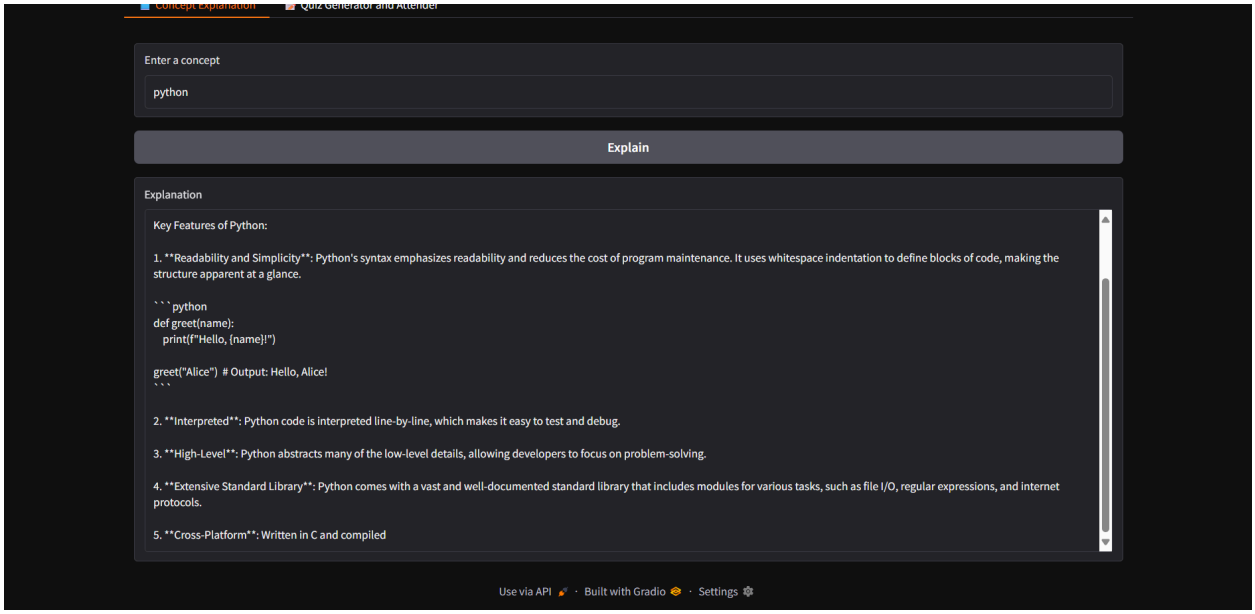
7. Test with samples, refine.
8. Deploy via Gradio.
9. Document and present.

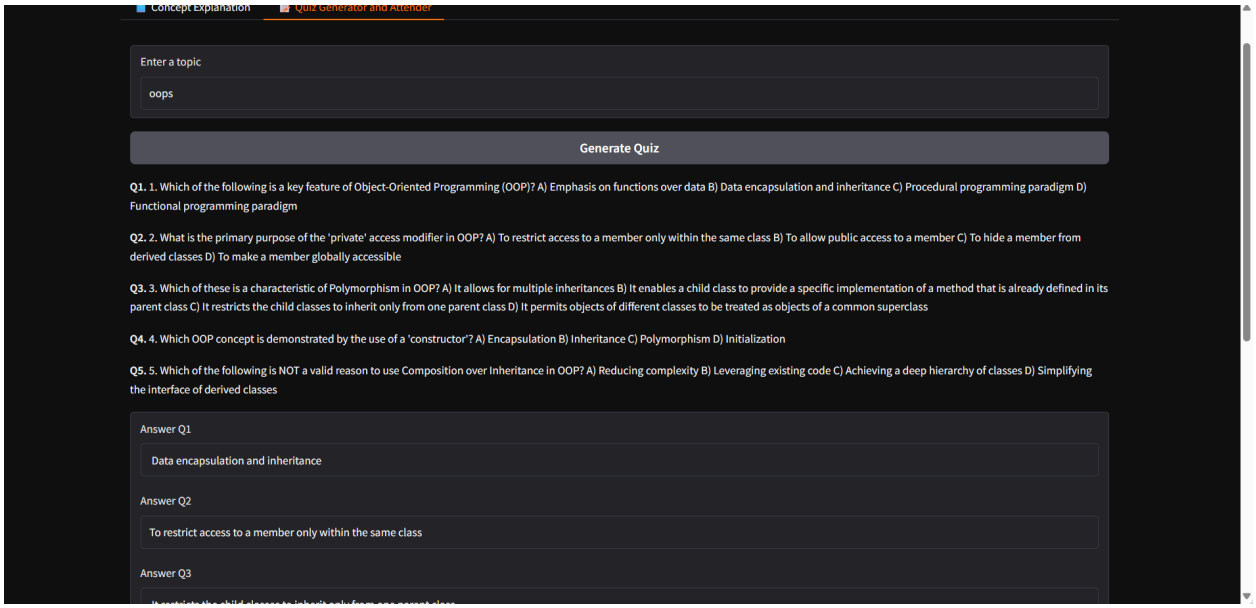
Results & Screenshots:

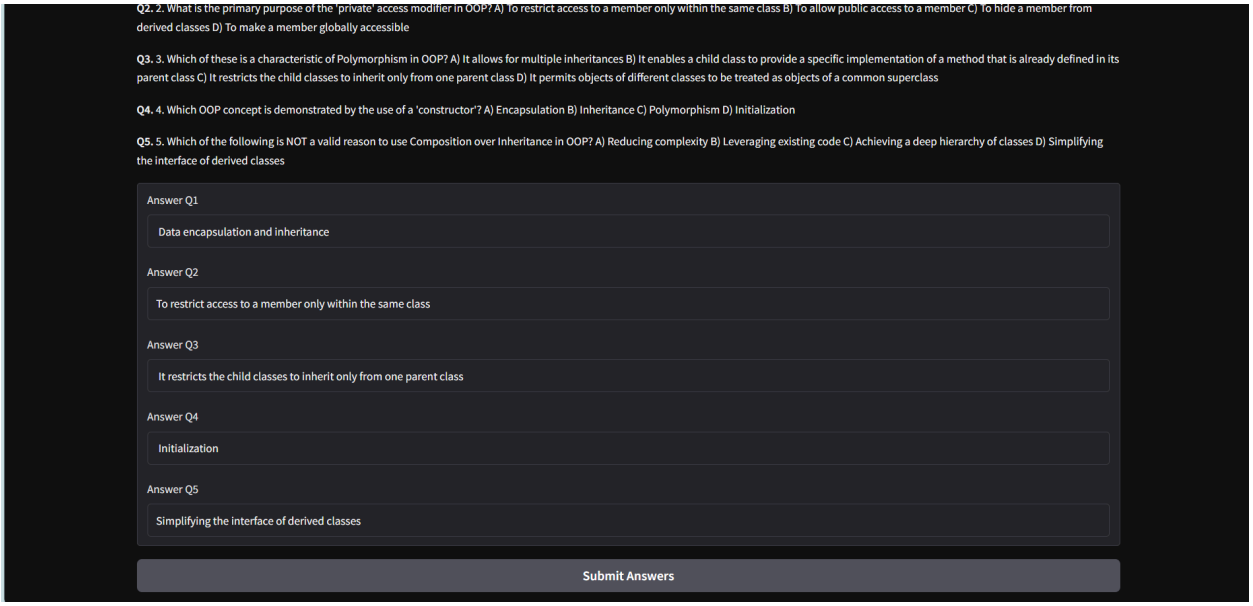



Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
[1]
f"Generate exactly 5 quiz questions about {topic} with multiple choice options.
f"Each question should have options labeled as A), B), C), and D). "
f"Format like:\n"
f"1. Question text?\n"
f"A) option1\n"
f"B) option2\n"
f"C) option3\n"
f"D) option4\n\n"
f"At the end, provide all correct answers in a separate ANSWERS section."
)
full_text = generate_response(prompt, max_new_tokens=600)
if "ANSWERS" in full_text:
questions_part, answers_part = full_text.split("ANSWERS", 1)
else:
questions_part, answers_part = full_text, ""
questions = [q.strip() for q in questions_part.strip().split("\n") if q.strip()]
answers = [a.strip() for a in answers_part.strip().split("\n") if a.strip()]
return questions, answers
def clean_questions(questions):
cleaned = []
skip = 0
for i, line in enumerate(questions):
if skip > 0:
skip -= 1
continue
# Assuming question lines start with numbers, options with letters
if line[0].isdigit() and line[1] == ".":
# Append question line + next 4 lines as options
block = line
opt_lines = []
for j in range(1,5):
# Defensive check to not go out of range
if i+j < len(questions):
Variables
Terminal
6:58AM
T4 (Python 3)


Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
[1]
/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/settings/tokens), set it as secret in your Google Colab and restart your session.
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, 880kB/s]
vocab.json: 777k/? [00:00<00:00, 18.1MB/s]
merges.txt: 442k/? [00:00<00:00, 29.9MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 85.2MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 8.56kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 63.5kB/s]
config.json: 100% 786/786 [00:00<00:00, 95.1kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 2.99MB/s]
Fetching 2 files: 100% 2/2 [01:04<00:00, 64.35s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:04<00:00, 70.6MB/s]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:06<00:00, 9.23MB/s]
Loading checkpoint shards: 100% 2/2 [00:19<00:00, 7.89s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 13.3kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://cf63c58bbacd52a458.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co
Variables
Terminal
6:58AM
T4 (Python 3)


Concept Explanation
Quiz Generator and Attender
Enter a concept
python
Explain
Explanation
Key Features of Python:
1. **Readability and Simplicity**: Python's syntax emphasizes readability and reduces the cost of program maintenance. It uses whitespace indentation to define blocks of code, making the structure apparent at a glance.
```python
def greet(name):
print(f"Hello, {name}!")
greet("Alice") # Output: Hello, Alice!
```
2. **Interpreted**: Python code is interpreted line-by-line, which makes it easy to test and debug.
3. **High-Level**: Python abstracts many of the low-level details, allowing developers to focus on problem-solving.
4. **Extensive Standard Library**: Python comes with a vast and well-documented standard library that includes modules for various tasks, such as file I/O, regular expressions, and internet protocols.
5. **Cross-Platform**: Written in C and compiled
Use via API · Built with Gradio · Settings


Concept Explanation
Quiz Generator and Attender
Enter a topic
oops
Generate Quiz
Q1. 1. Which of the following is a key feature of Object-Oriented Programming (OOP)? A) Emphasis on functions over data B) Data encapsulation and inheritance C) Procedural programming paradigm D) Functional programming paradigm
Q2. 2. What is the primary purpose of the 'private' access modifier in OOP? A) To restrict access to a member only within the same class B) To allow public access to a member C) To hide a member from derived classes D) To make a member globally accessible
Q3. 3. Which of these is a characteristic of Polymorphism in OOP? A) It allows for multiple inheritances B) It enables a child class to provide a specific implementation of a method that is already defined in its parent class C) It restricts the child classes to inherit only from one parent class D) It permits objects of different classes to be treated as objects of a common superclass
Q4. 4. Which OOP concept is demonstrated by the use of a 'constructor'? A) Encapsulation B) Inheritance C) Polymorphism D) Initialization
Q5. 5. Which of the following is NOT a valid reason to use Composition over Inheritance in OOP? A) Reducing complexity B) Leveraging existing code C) Achieving a deep hierarchy of classes D) Simplifying the interface of derived classes
Answer Q1
Data encapsulation and inheritance
Answer Q2
To restrict access to a member only within the same class
Answer Q3



The application successfully:

- Generates explanations automatically.
- Creates formatted quizzes with evaluations.

Known Issues

During development and testing of the Educational AI Assistant, several potential issues were identified based on the technologies used. These are derived from common reports in the Hugging Face community, Stack Overflow discussions, and official documentation for Granite-3.2 and Gradio integrations. While the project functions as intended in controlled environments like Google Colab with T4 GPU, users may encounter the following:

1. **Device Mismatch Errors in Gradio with Transformers/PyTorch**: When running the app, especially on mixed CPU/GPU setups, a common error occurs where tensors are placed on incompatible devices (e.g., model on GPU but inputs on CPU). This leads to runtime errors like "RuntimeError: Expected all tensors to be on the same device." Solution: Explicitly move inputs and model to the same device (e.g., using to(device) in PyTorch) before inference. This issue is frequently reported in Gradio demos integrating Transformers models.
2. **Model Loading and Memory Constraints**: The Granite-3.2-2B-Instruct model (2 billion parameters) requires significant RAM/GPU memory (at least 4-6 GB VRAM recommended). On low-resource machines or without GPU acceleration, loading via AutoModelForCausalLM.from_pretrained() may fail or cause out-of-memory errors. In Google Colab, ensure T4 GPU is selected; for local runs, use quantization (e.g., via torch_dtype=torch.bfloat16). Inherited from earlier Granite versions, the model may produce inaccurate or biased responses in edge cases, requiring safety testing.
3. **Pipeline Compatibility with Gradio**: For certain tasks like object detection or custom pipelines, gr.Interface.from_pipeline() may raise "Unsupported pipeline type" errors if the Transformers pipeline isn't directly compatible (e.g., for non-standard tasks). In this project, since we use custom functions for quiz generation

and explanation, this is avoided by defining manual predict functions. However, if extending to other pipelines, verify compatibility.

4. **Generation Quality and Prompt Sensitivity**: As an instruct-tuned model, Granite-3.2 performs well on reasoning tasks but may generate inconsistent or hallucinated content for complex educational topics, especially without refined prompts. Users report occasional degraded performance in multilingual or highly specialized domains. Mitigation: Use structured prompts with <think></think> tags for better reasoning, as recommended in IBM documentation.
5. **Deployment and Sharing Limitations**: When sharing via Gradio's public links (e.g., in Colab), sessions may timeout after inactivity, and high-traffic use can exceed free tier limits on Hugging Face Spaces. Additionally, without user authentication, progress isn't saved across sessions. For production, deploy on Hugging Face Spaces or IBM Cloud, but monitor for API rate limits if integrating external services.

These issues do not prevent core functionality but highlight areas for optimization. The model inherits ethical limitations from prior versions, such as potential biases in outputs, and should be used with safeguards like Granite Guardian for risk detection. Future enhancements can address these through fine-tuning or additional safety layers.

Conclusion & Future Scope

Educational AI Assistant showcases Generative AI's role in education by offering personalized explanations, quizzes, and feedback. It meets objectives through integrated generation and interactive UI.

Future Enhancements:

- Multi-language support (Tamil, Hindi, etc.)
- Voice-enabled features for accessibility
- Predictive analytics for performance
- LMS integrations
- Additional question types (true/false, short answer)
- User authentication for progress tracking
- Text-to-speech delivery

Conclusion

The **Educational AI Assistant** project demonstrates the powerful potential of combining advanced AI language models with interactive web interfaces to enhance learning experiences. By leveraging the Granite-3.2 model in a Gradio application, the assistant effectively supports users with detailed concept explanations and dynamically generated quizzes tailored to chosen topics. This fosters active learning and provides immediate feedback, essential for student engagement and mastery.

This project highlights the practicality of AI in education — offering scalable, personalized educational assistance without the constraints of human resource limitations. It shows how carefully designed prompts and interface logic can create clear, structured educational content from AI-generated text.

Future developments can further improve interactivity and accessibility, expanding to diverse question formats, adaptive learning paths, and multimodal content delivery. Overall, the project affirms that AI-powered educational tools can augment traditional methods, helping learners achieve their goals more efficiently and confidently.