**Project Tree Structure**

Project Tree Structure

MultiModal\_GenProject/

│

├── app/

│ ├── \_\_init\_\_.py # Package initialization file.

│ ├── aclient.py # Streamlit client for audio generation.

│ ├── client.py # Streamlit client for text generation.

│ ├── main.py # FastAPI server handling text, audio, and image generation endpoints.

│ ├── models.py # Model loading and generation logic for text, audio, and images.

│ ├── schemas.py # Schema definitions for input validation (e.g., audio presets).

│ ├── utils.py # Utility functions for handling audio and image processing.

│ ├── vclient.py # Streamlit client for image generation.

│

├── .env # Environment variables (e.g., Hugging Face token).

├── .gitignore # Files and directories to be ignored by Git.

├── requirements.txt # Python dependencies for the project.

Technical Components

1. FastAPI (Backend)

Purpose:

Hosts RESTful endpoints for text, audio, and image generation.

Facilitates interaction between the models and client applications.

Key Endpoints:

/generate/text: Generates a text response based on a user prompt using a text generation model.

/generate/audio: Generates audio (speech synthesis) using an audio generation model and supports customizable voice presets.

/generate/image: Produces an image based on a text prompt using a diffusion model.

/: Redirects to OpenAPI documentation.

Key Features:

Implements efficient caching for reusable model instances.

Uses StreamingResponse for streaming audio files.

Encodes images and audio into transferable formats (PNG, WAV).

2. Streamlit Clients (Frontend)

Purpose:

Provide user-friendly interfaces for interacting with the FastAPI backend.

Clients:

client.py: Facilitates text generation and chatbot-style interactions.

aclient.py: Handles audio generation and playback of synthesized speech.

vclient.py: Enables users to generate and view AI-generated images.

Key Features:

Chat Interface: Maintains chat history using st.session\_state.

Dynamic Content Rendering:

Text: Displays text responses.

Audio: Streams audio files for playback.

Images: Displays generated images directly in the UI.

Seamless Integration: Communicates with FastAPI endpoints via REST API calls.

3. Models and Generation Logic (models.py)

Text Generation:

Uses a Hugging Face text generation pipeline (e.g., TinyLlama/TinyLlama-1.1B-Chat-v1.0).

Processes user input with a system prompt to maintain conversational context.

Audio Generation:

Leverages Suno’s bark-small model for speech synthesis.

Customizable voice presets enhance personalization.

Image Generation:

Integrates segmind/tiny-sd (a Stable Diffusion variant) for text-to-image synthesis.

Supports high-quality output with controlled inference steps.

4. Utilities (utils.py)

Audio Processing:

Converts NumPy audio arrays to WAV format using soundfile.

Ensures compatibility for streaming.

Image Conversion:

Converts PIL.Image objects to byte streams for HTTP responses.

5. Schemas (schemas.py)

Purpose:

Provides validation for input parameters.

Defines VoicePresets as a Literal type for allowed audio presets.

Integration Flow

Frontend (Streamlit):

Users interact with the chatbot interface in Streamlit.

Inputs (text/image/audio prompts) are sent to the FastAPI backend.

Responses (text/audio/image) are displayed in the UI.

Backend (FastAPI):

Receives requests from Streamlit clients.

Routes the requests to appropriate model inference functions.

Processes the responses (e.g., encodes audio, converts images).

Sends the results back to the clients.

Models:

Loaded on-demand (or globally for optimization).

Perform inference to generate text, audio, or images.

Business Impact

Enhanced User Experience:

Enables interactive and multimodal AI content generation for applications such as virtual assistants, creative tools, and educational platforms.

Personalized Content Generation:

Offers customizable voice and image generation, catering to diverse user preferences.

Prototype for Product Development:

Serves as a foundation for building AI-powered chatbots, creative studios, or assistive technologies.

Streamlined Deployment:

Combines the power of FastAPI (scalable backend) with Streamlit (intuitive frontend) for rapid prototyping and deployment.

This project is a showcase of how generative AI models can be effectively integrated into real-world applications, leveraging a scalable architecture with impactful multimodal capabilities.

PROJECTS Large Language Model Training Pipeline (PyTorch, Distributed Computing) GitHub ● Implemented distributed training pipeline for transformer models across multi-GPU clusters, achieving 70% reduction in training time through optimized parallelization strategies ● Developed custom model architecture for enterprise use-cases, demonstrating scalability across 8-node GPU cluster while maintaining training stability GenAI Inference Optimization Framework (TensorFlow, JAX) GitHub ● Built production-grade inference system handling 1000+ concurrent requests, optimizing GPU utilization and reducing latency by 40% through efficient batching ● Designed automated deployment pipeline for ML models, enabling seamless scaling across distributed infrastructure while ensuring consistent performance Enterprise MLOps Platform (AWS, Docker) GitHub ● Architected containerized ML training environment supporting distributed workloads across multiple AWS regions ● Implemented monitoring and observability solutions for large-scale model training, reducing debugging time by 60% through automated metric collection