**Documentation Report: Multi-Modal AI Chatbot**

**1. Project Overview**

The **Multi-Modal AI Chatbot** is a Gradio-based application designed to leverage Google's Gemini AI for processing both **text and image inputs**. This application allows users to engage in natural language conversations, perform detailed image analysis, and execute combined multi-modal queries (text accompanying an image). The key benefit is its ability to switch between text-only and vision-capable models dynamically based on user input.

**2. Architecture and Core Components**

The application is structured around a **client-side Gradio interface** and **server-side Python functions** that interact with the Google Generative AI API.

**2.1. Technology Stack**

| Component | Library/Framework | Version/Use | Purpose |
| --- | --- | --- | --- |
| **Frontend/Interface** | gradio | 4.44.0 | Creates the web-based, interactive user interface. |
| **AI Backend** | google-generativeai | 0.3.2 | Handles API communication, model configuration, and content generation. |
| **Image Handling** | Pillow (PIL) | 10.1.0 | Used for opening and processing image files for vision tasks. |

**2.2. Key Functions and Logic**

| Function Name | Purpose | Models Targeted |
| --- | --- | --- |
| get\_available\_model(api\_key) | Selects the best available **text-only model** from a list of candidates (e.g., gemini-2.5-pro, gemini-2.5-flash, gemini-pro). | Text models |
| get\_available\_vision\_model(api\_key) | Selects the best available **vision-capable model** from a list of candidates (e.g., gemini-2.5-pro, gemini-1.5-flash, gemini-pro-vision). | Multi-modal models |
| process\_multimodal\_input(...) | **Core chat function**. Determines if the input is text-only or multi-modal (text + image) and calls the appropriate model. It also adds the last two turns of conversation to the prompt for **context awareness**. | Text and Vision models |
| generate\_image\_description(...) | Specifically dedicated to the **Image Analysis** tab, generating a detailed description based on a comprehensive prompt. | Vision models |
| test\_api\_key(api\_key) | Verifies the API key and lists up to 15 available models supported by the API key. | N/A (Uses genai.list\_models()) |

**3. User Interface and Workflow**

The interface is organized into three main tabs within a Gradio gr.Blocks structure.

**3.1. API Key Configuration**

* A dedicated gr.Textbox is provided for the user to input their **Google Gemini API Key**.
* The **"Test API Key"** button runs the test\_api\_key function to validate the key and list available models.

**3.2. "💬 Chat" Tab**

* **Inputs:** A primary message input box (gr.Textbox) and an optional image upload component (gr.Image).
* **Output:** A standard gr.Chatbot component for displaying the conversation history.
* **Workflow:**
  + If an **image is uploaded**, a vision model is used for the prompt (e.g., "Describe this image in detail" if no text is provided).
  + If **only text is provided**, a text model is used, incorporating the last few messages for conversation context.
  + A "Clear Chat" button is available to reset the conversation history.

**3.3. "🔍 Image Analysis" Tab**

* This tab focuses purely on visual analysis.
* Users upload an image and click **"Analyze Image"**.
* The generate\_image\_description function is called, using a detailed, fixed prompt to describe subjects, scene, colors, composition, and potential context.

**4. Key Operational Notes and Guides**

* **Model Selection:** The app attempts to prioritize higher-capability models like gemini-2.5-pro first, falling back to lighter models like gemini-2.5-flash or gemini-pro-vision if the preferred model is unavailable.
* **Context Management:** For text conversations, the last two turns of the chat history are prepended to the user's new message to maintain conversational flow.
* **Error Handling:** If the API key is missing or invalid, or if no appropriate model can be found, the chatbot returns an error message to the user.
* **API Key Security:** The application guide explicitly states that the API key is **not stored on servers**, conversations are **not saved permanently**, and all data is **session-based**.