AI - Powered Emergency Response Assistant

In [1]:

```
import os
import sys
import time
import panel as pn
import google.generativeai as genai

# ------------------------------
# 1. Panel Setup
# ------------------------------
pn.extension()

# Add utils path if needed
sys.path.append('../..')

# Set your Gemini API key
genai.configure(api_key="AIzaSyAeyMCu97NaE4vJL5-StrB68ZoJT9qoRLE")

# Gemini model to be used
GEMINI_MODEL = "gemini-1.5-flash"

# ------------------------------
# 2. Helper Function - Gemini Call
# ------------------------------
def get_completion_from_messages(messages, model=GEMINI_MODEL, temperature=0, max_tokens
    """
    Converts role-based messages into a single prompt for Gemini and gets a response.
    """
    conversation = "\n".join([f"{msg['role'].capitalize()}: {msg['content']}" for msg in

    gemini_model = genai.GenerativeModel(model)
    response = gemini_model.generate_content(
        conversation,
        generation_config=genai.types.GenerationConfig(
            temperature=temperature,
            max_output_tokens=max_tokens,
        )
    )
    return response.text

# ------------------------------
# 3. Moderation Check
# ------------------------------
def moderate_content(input_text):
    """
    Checks for unsafe or false emergency reports using a custom moderation prompt.
    Returns True if unsafe content is found.
    """
    moderation_prompt = f"""
    You are a strict safety filter for an emergency assistant.
    Analyze the following report and decide if it contains unsafe, harmful, or fake cont

    Text: ```{input_text}```
```

```
    Respond ONLY with:
    SAFE - If the text is safe and valid.
    UNSAFE - If the text contains dangerous, fake, or harmful content.
    """

    moderation_model = genai.GenerativeModel(GEMINI_MODEL)
    response = moderation_model.generate_content(moderation_prompt)
    return "UNSAFE" in response.text.upper()

# ----------------------------
# 4. Process User Message
# ----------------------------
def process_user_message(user_input, all_messages, debug=True):
    """
    Handles user input, runs moderation checks, and generates AI response.
    """
    delimiter = "```"

    # Step 1: Moderation Check for user input
    if moderate_content(user_input):
        if debug:
            print("Moderation: Input flagged as unsafe or invalid.")
        return " Your message was flagged as unsafe or invalid. Please provide accurate

    if debug:
        print("Moderation: Input passed moderation check.")

    # Step 2: Generate Emergency Guidance
    system_message = """
    You are an AI emergency response assistant.
    - Understand the emergency described by the user.
    - Provide clear, step-by-step actions for safety.
    - If the situation is life-threatening, immediately advise calling local emergency n
    - Keep the response concise and calm.
    """

    messages = [
        {'role': 'system', 'content': system_message},
        {'role': 'user', 'content': f"{delimiter}{user_input}{delimiter}"}
    ]

    final_response = get_completion_from_messages(all_messages + messages)

    if debug:
        print("Generated Response:", final_response)

    # Step 3: Final Moderation Check on AI Output
    if moderate_content(final_response):
        if debug:
            print("Final Response flagged by moderation.")
        return " Unable to display response due to unsafe content.", all_messages

    # Update conversation history
    all_messages = all_messages + messages[1:]
    return final_response, all_messages

# ----------------------------
# 5. Collect Messages for Chat
# ----------------------------
```

```python
def collect_messages(debug=False):
    """
    Collects messages, updates chat history, and displays them on the dashboard.
    """
    user_input = inp.value_input
    if debug:
        print(f"User Input = {user_input}")

    if user_input == "":
        return

    # Clear input box
    inp.value = ''

    global context
    response, context = process_user_message(user_input, context, debug=True)

    # Append messages to chat history
    panels.append(
        pn.Row(
            ' **User Report:**',
            pn.pane.Markdown(user_input, width=600, styles={'color': 'red'})
        )
    )
    panels.append(
        pn.Row(
            ' **AI Guidance:**',
            pn.pane.Markdown(response, width=600, styles={'background-color': '#F0F8FF'}
        )
    )

    return pn.Column(*panels)

# ------------------------------
# 6. GUI Setup
# ------------------------------
panels = []  # Chat history
context = [{'role': 'system', 'content': "You are Service Assistant"}]

# Input widget
inp = pn.widgets.TextInput(placeholder='Enter emergency details here...')

# Button widget
button_conversation = pn.widgets.Button(name="Service Assistant")

# Bind function to button click
interactive_conversation = pn.bind(collect_messages, button_conversation)

# Create dashboard layout
dashboard = pn.Column(
    "#  Emergency Response Assistant",
    "Provide details about the emergency situation, and get step-by-step guidance.",
    pn.Spacer(height=10),
    inp,
    pn.Row(button_conversation),
    pn.panel(interactive_conversation, loading_indicator=True, height=400),
)

dashboard.servable()
```

```
# ----------------------------
# 7. Run Instructions
# ----------------------------
# Run this app using:
# panel serve emergency_system.py --show --port 5006
```

Out[1]:

[1]: