Building AI Agents with Autogen

Demo 1

Demo: Research AI Agents with Autogen

Overview

The Autogen-powered Research Agent leverages advanced AI models to assist users in researching a topic by generating detailed AI-driven summaries for each subtopic. This interactive agent allows users to input a broad topic, which is then broken down into subtopics. For each subtopic, the agent generates a short summary, allowing users to review, revise, and approve the content. This tool combines **Generative AI**, **Human-in-the-Loop** feedback, and a structured multi-step workflow for efficient research assistance.

Scenario

A researcher needs to quickly gather insights on a specific topic. The researcher may have to break down a complex subject into smaller, manageable subtopics and gather concise summaries for each. However, manually conducting this research is time-consuming and labor-intensive. By leveraging the Autogen-powered Research Agent, the researcher can save time, get organized summaries, and focus on deeper analysis and decision-making.

Problem Statement

The problem is the manual, time-consuming task of breaking down complex research topics into subtopics and generating summaries. The research agent automates these tasks, allowing users to focus on evaluating and using the results instead of performing repetitive research tasks.

Approach for the Solution

The solution is an interactive AI agent powered by Autogen's AssistantAgent and UserProxyAgent, orchestrated in a Streamlit app. The workflow is:

1. User inputs a research topic or question.
2. The assistant agent generates an answer using Gemini LLM.
3. The user can request a summary of the last response.
4. The app stores conversation history in Streamlit's session state, enabling multi-turn interaction and HITL control by requiring the user to trigger new responses, generate subtopics or summarize.

This approach balances automation with user-driven control, ensuring a flexible research experience.

Salient Features

- **Generative AI Model:** Uses Google Gemini LLM (configured via model_config.json) to generate detailed answers and summaries.
- **Human-in-the-Loop:** User explicitly inputs topics/questions and controls when to ask for summaries or generate subtopics.
- **Interactive Interface:** Streamlit provides a user-friendly UI for input, displaying AI responses, and summary requests.
- **Session Management:** Uses st.session_state to keep track of conversation messages for continuity.
- **Code Execution:** Supports limited code execution configuration for extensibility, though disabled Docker usage.

Code:

```
if st.button("Ask"):
    if not topic.strip():
        st.warning("Please enter a question or topic.")
    else:
        st.session_state.assistant_messages = []
        user_proxy.initiate_chat(assistant, message=topic)
```

Roles and Conversations

Roles:

- **User:** Provides the research topic or question and controls the conversation flow.
- **Assistant:** Generates AI-driven responses and summaries based on user input.

Code:

```
assistant = AssistantAgent(name="assistant")
user_proxy = UserProxyAgent(name="user_proxy")

# Start conversation
user_proxy.initiate_chat(assistant, message="Hello, assistant!")
```

Conversation Pattern:

1. User submits a question/topic.
2. Assistant generates and returns an answer.
3. User optionally requests a summary of the last answer.
4. Conversation continues or terminates based on user input.

Chat Terminations

The conversation can be terminated anytime by sending a message containing "TERMINATE". The system recognizes this through the termination_msg function in the UserProxyAgent, ending the chat session gracefully.

Code:

```
user_proxy = UserProxyAgent(
    name="user_proxy",
    is_termination_msg=lambda msg: "TERMINATE" in msg.get("content", "")
)
```

Human-in-the-Loop

Human-in-the-Loop is implemented by requiring user input for each new query or summary request. The assistant does not auto-respond indefinitely but waits for explicit user triggers. This approach ensures user oversight, letting them control when the agent responds or when to end the interaction.

Code:

```
user_proxy = UserProxyAgent(
    name="user_proxy",
    human_input_mode="ALWAYS"  # Requires user input every turn
)
```

Code Execution

The backend Python code orchestrates:

- Initializing the Autogen agents (AssistantAgent and UserProxyAgent) with Gemini config.
- Managing message reception and storing messages in st.session_state.
- Handling user inputs and button clicks to trigger assistant responses or summarization prompts.
- Detecting termination commands within the user messages.
- Running inside Streamlit to provide a live, interactive web app.

Code:

```
user_proxy = UserProxyAgent(
    name="user_proxy",
    code_execution_config={
        "work_dir": "coding",
        "use_docker": False
    }
)
```

Tool Use

- **Autogen:** Provides agent classes (AssistantAgent, UserProxyAgent) and conversation management.
- **Google Gemini (via config):** The generative LLM model used for responses and summaries.
- **Streamlit:** Web framework for UI components (text input, buttons, markdown display).
- **Python:** Core logic for agent orchestration, session state, and custom message handling.

Code:

```
from autogen import Tool

def summarise_text(text: str) -> str:
    # simple summarization logic or call an API/LLM
    return text[:200] + "..." if len(text) > 200 else text

summarise_tool = Tool(
    name="summarise_text",
    description="Summarizes input text concisely.",
    func_or_tool=summarise_text
)

assistant.add_tool(summarise_tool)
```

Conversation Patterns

1. User inputs research topic →
2. Assistant replies with AI-generated answer →
3. User optionally clicks “Generate Subtopics” →
4. User optionally clicks “Summarise” →
5. Assistant replies with concise summary →

Code:

```
assistant = AssistantAgent(
    name="assistant",
    max_consecutive_auto_reply=3  # Limits auto replies to prevent endless loops
)
```

Developing Autogen-powered Agents

To build this agent:

- Select the generative AI model (Gemini) via config.

- Define assistant and user proxy agents with appropriate configurations.
- Implement message reception to update UI state.
- Build Streamlit UI for user interaction and control.
- Enable termination detection and controlled multi-turn conversation.

Code:

```
# Create assistant agent using Gemini or other LLM
assistant = AssistantAgent(
    name="assistant",
    llm_config={"config_list": config_list, "seed": 42},
    max_consecutive_auto_reply=3,
    system_message="You are a helpful assistant."
)
```

Deployment and Monitoring

The app is deployed as a Streamlit web app, accessible via browser. Monitoring includes:

- Tracking user inputs and AI responses in session state.
- Observing termination rates and usage patterns.
- Collecting user feedback on response quality (future extension).

Code:

```
# Streamlit UI acting as deployment layer
st.set_page_config(page_title="Research Agent", layout="centered")
st.title("📚 Gemini Research Agent")
```

Structure of the Solution

1. **User Interface (Streamlit)**
    - Text input for research topic/question.
    - Buttons for “Ask”, “Generate Subtopics” and “Summarise”.
    - Display area for assistant responses and summaries.
    - Buttons for “Download in Pdf”, “Download in Word” and “Download in .csv”.
2. **Autogen Agents**
    - AssistantAgent: Uses Gemini LLM to generate responses.
    - UserProxyAgent: Manages user inputs, detects termination messages, and handles message reception.
3. **Conversation Management**
    - Message storage and UI updates via st.session_state.
    - Controlled message flow triggered by button clicks.
4. **Termination Handling**
    - Detects "TERMINATE" to end conversation.
5. **Custom Message Handling**
    - Overwrites receive method in UserProxyAgent to update UI messages live.

Stepwise Solution

Step 1: Setup Environment

- Install Python, Streamlit, Autogen, and dependencies.
- Prepare model_config.json for Gemini LLM.

Code:

```
from autogen import AssistantAgent, UserProxyAgent
```

Step 2: Initialize Agents

- Create AssistantAgent with Gemini config.
- Create UserProxyAgent with termination detection and custom receive function.

```
user_proxy = UserProxyAgent(
    name="user_proxy",
    code_execution_config={
        "work_dir": "coding",
        "use_docker": False
    }
)
```

Step 3: Build Streamlit UI

- Add text input for topics/questions.
- Add buttons to trigger assistant response and summarisation.
- Display conversation history from st.session_state.

```
# Create assistant agent using Gemini or other LLM
assistant = AssistantAgent(
    name="assistant",
    llm_config={"config_list": config_list, "seed": 42},
    max_consecutive_auto_reply=3,
    system_message="You are a helpful assistant."
)
```

Step 4: Handle Conversation Logic

- On “Ask”, clear previous messages, send user input to assistant.
- On “Generate Subtopics”, send a prompt to generate detailed subtopics.

- On “Summarise”, send a summarisation prompt with last assistant response.
- Append AI responses to session state and update UI.

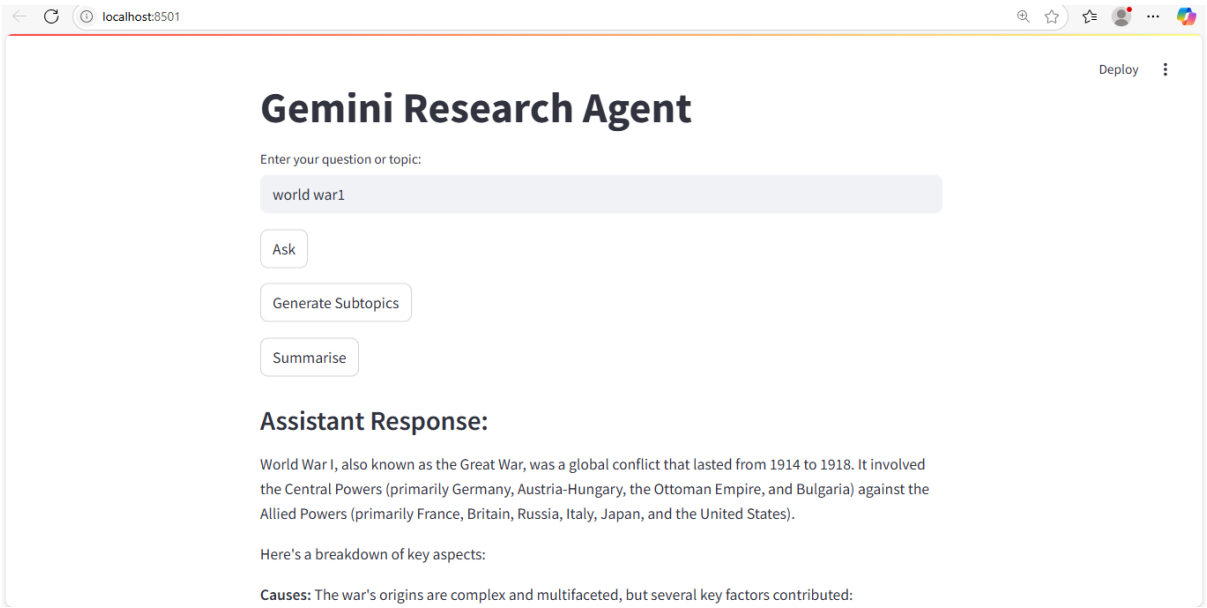
```
if st.button("Ask"):
    if not topic.strip():
        st.warning("Please enter a question or topic.")
    else:
        st.session_state.assistant_messages = []
        user_proxy.initiate_chat(assistant, message=topic)
```

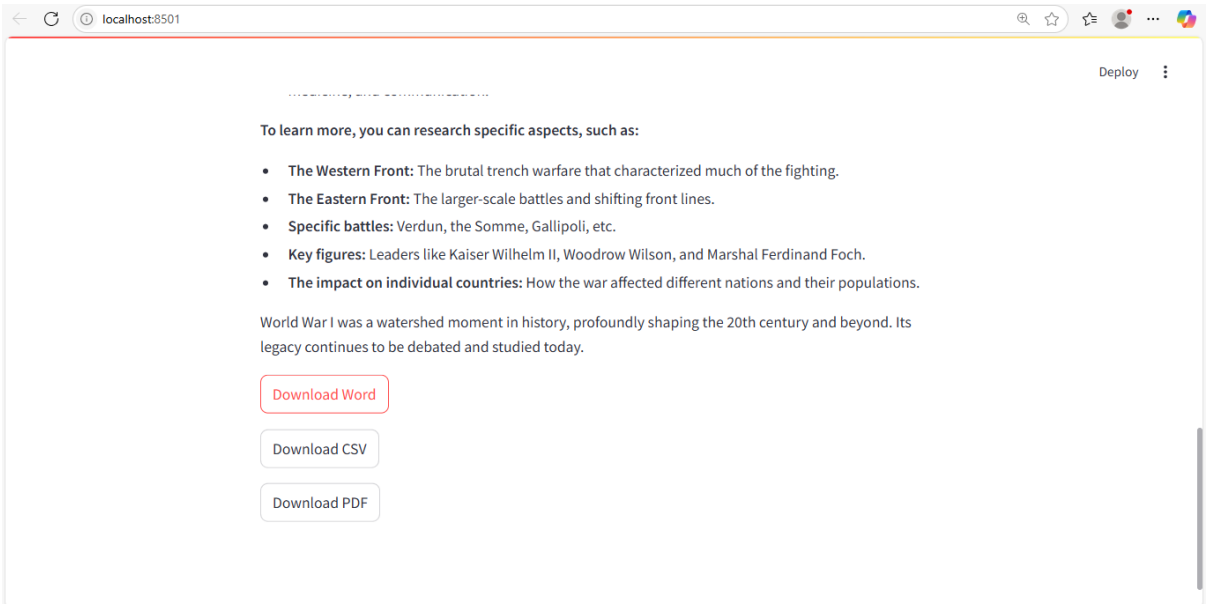
Step 5: Termination and Human Control

- Detect termination messages to stop conversation.
- Require user-triggered actions to continue chat.

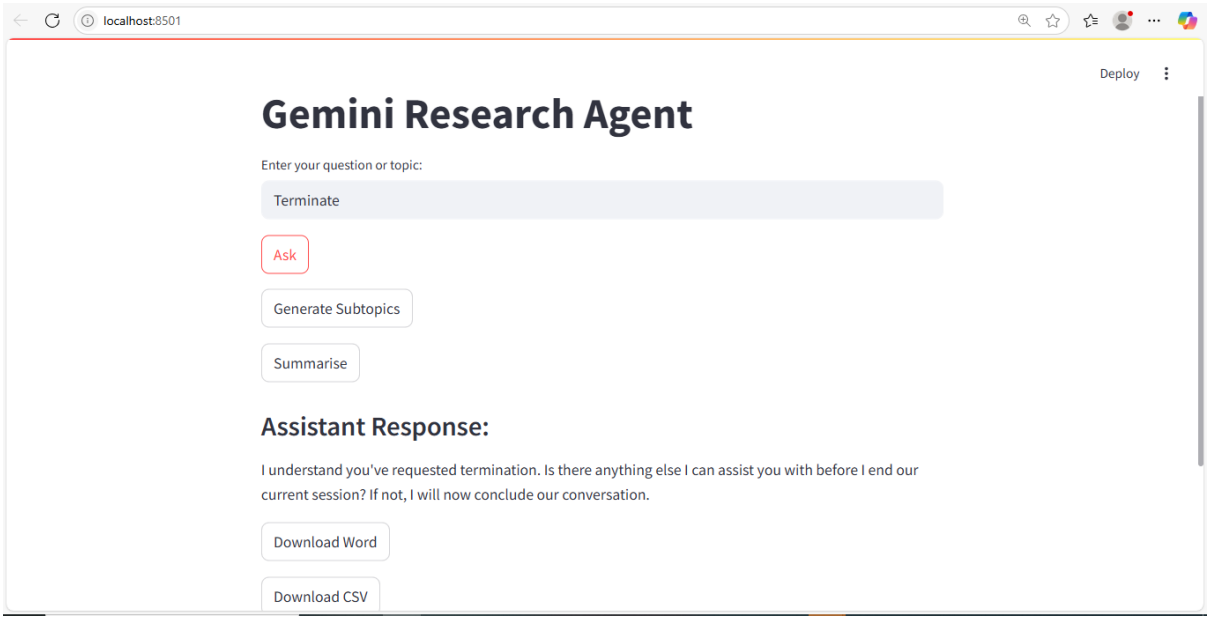
```
user_proxy = UserProxyAgent(
    name="user_proxy",
    is_termination_msg=lambda msg: "TERMINATE" in msg.get("content", "")
)
```

Output:





Terminate:



Conclusion

This project demonstrates a streamlined Research Agent using Autogen and Google Gemini, deployed on Streamlit for easy interaction. The agent supports user-driven research via natural language questions and generates informative answers with optional summarisation. The Human-in-the-Loop design ensures that users retain control over the conversation flow and content quality. This architecture exemplifies how multi-turn AI conversations can be managed efficiently with Autogen's agent framework, providing a foundation for more complex research assistant applications.