WEEK 2 / DAY 10

Microsoft AutoGen

Conversational Programming &

Autonomous Coding Agents.

By,

Shikha Tyagi

Founder - AI JAMIC (AI Research and Consulting)

Education: IIT Delhi (M.Tech.)

Today's Agenda

Part 1: The AutoGen Philosophy

Conversation as Control Flow.

Part 2: Core Components

Assistant vs. UserProxy.

Part 3: Framework War

AutoGen vs. CrewAI.

Part 4: Hands-On Lab

Break:

15 min break around 10:30AM

Week 2 - Working with Frameworks

Day 6 & Day 7 ✓

LangChain

Day 8 ✓

LlamaIndex

Day 9 ✓

crewai

Day 10

AG

Which of the following BEST differentiates an *agent* from a traditional chatbot?

A. Agents always require GPU hardware, while chatbots do not
B. Agents can perform multi-step reasoning + take actions via tools, while chatbots only respond to text
C. Chatbots use APIs but agents cannot
D. Agents cannot store memory but chatbots can

The R.O.L.E.S. prompt framework primarily helps with:

A. Optimizing LLM cost per API call
B. Structuring instructions to improve model controllability and output consistency
C. Reducing the number of tokens in an API request
D. Increasing GPU speed in local inference

Which of the following is TRUE about embeddings in a RAG system?

A. They store the entire document in plain text form
B. They map text into high-dimensional vectors enabling similarity search
C. They eliminate the need for chunking and text splitting
D. They can only be computed using OpenAI models

When building a "Chat with Your PDF" app using LangChain, the **VectorStoreRetriever** is used to:

A. Upload PDFs to the cloud for storage
B. Re-rank the model outputs
C. Fetch the most relevant embedded chunks for a query
D. Preprocess PDF text into smaller chunks

In CrewAI, a **Task** differs from an **Agent** because:

A. Only tasks can access tools
B. Tasks define *what needs to be done*, while agents define *who performs the task*
C. Agents can run only one task at a time
D. Tasks cannot contain descriptions or expected outputs

The Next Evolution

1. Yesterday, we learned **CrewAI** (Process-Driven).

https://microsoft.github.io/autogen/stable//user-guide/agentchat-user-guide/migration-guide.html#assistant-agent

2. Today, we will work on **AutoGen** (Conversation-Driven).

"What if solving a problem was as simple as two agents chatting in a Slack channel?"

The "Conversational" Paradigm

Standard Agents

Input -> Thought -> Action -> Output.

Linear and rigid.

AutoGen

Agent A says something. Agent B responds.

Agent A corrects Agent B.

Dynamic and self-correcting loop.

Key Superpower: Code Execution

Most agents "pretend" to write code (they output text).

AutoGen agents **ACTUALLY run the code**.

- If an agent writes a Python script to plot a chart...
- The other agent executes it locally (or in Docker) and reports the error.
- The first agent fixes the bug based on the error message.

1. The AssistantAgent

The "Brain".

- Backed by an LLM (GPT-4).
- Writes code, plans tasks, solves problems.
- **Cannot execute code.** It only suggests it.

```
     assistant = AssistantAgent(
name="coder", llm_config=llm_config )
```

2. The UserProxyAgent

The "Hands" (and the Human Proxy).

- Can execute code (locally/Docker).
- Can ask the Human for input.
- **Default behavior:** If it sees a code block in the chat, it runs it and replies with the result.

```
    user_proxy = UserProxyAgent(
name="user", human_input_mode="NEVER",
code_execution_config={"work_dir": "coding"}
)
```

The Interaction Loop

Assistant: "Here is python code to print 'Hello World'."

UserProxy: (Runs Code) "Output: Hello World"

Assistant: "Great! Task complete. TERMINATE."

Which agent in AutoGen is responsible for EXECUTING the Python code?

A. AssistantAgent

B. UserProxyAgent

C. The LLM itself

D. The GroupManager

Framework Comparison

Feature	CrewAI (Day 9)	AutoGen (Day 10)
Style	Role-Playing Process	Conversational
Control	Sequential/Hierarchical	Chat Loop (Auto-reply)
Code Exec	Via Tools (PythonREPL)	Native / Core Feature

When to use what?

Use CrewAI when...

You have a defined process. (Research -> to -> Write -> Edit). You want reliable, production-ready workflows.

Use AutoGen when...

You need code generation and execution. The problem is open-ended (e.g., "Build a snake game"). You want agents to debug each other.

Advanced Components
Group Chats

Scaling from 2 agents to a room full of them.

The GroupChat Manager

What if you have 3+ agents? (Coder, Designer, Product Manager).

The **GroupChatManager** acts as a moderator.

- It sees the conversation history.
- It decides **"Who speaks next?"** based on the LLM's decision.

Speaker Selection Methods

Auto

The LLM decides who is best suited to speak next.

Round Robin

A -> $ B -> C -> A.

Random

Chaotic, but sometimes useful for brainstorming.

Human Input Modes

The `UserProxyAgent` can pause for you (User input).

```
      human_input_mode="ALWAYS" # Stops after every agent message.  human_input_mode="TERMINATE"
# Stops only when the agent thinks the task         human_input_mode="NEVER" # Fully autonomous.
is done.
```

Human Input Modes

`human_input_mode`	Meaning	When to Use	Example Behavior
"ALWAYS"	Always pauses for human input before continuing	Teaching, demos, safety-critical operations	*“Should I execute this code? Y/N?”*
"NEVER"	Fully autonomous, no human asked	Automated workflows, fast experiments	Executes code immediately, retries automatically
"TERMINATE"	Stops when human input would normally be needed	Production automation where NO human input is allowed	AutoGen halts gracefully

In a Group Chat, who determines the next speaker by default?

A. The UserProxyAgent.

B. The GroupChatManager (via LLM).

C. The agents fight for it.

D. It is always Round Robin.

Why does the AssistantAgent typically NOT execute code?

A. It doesn't have permission.

B. It is designed to be the "Brain" (LLM) while the UserProxy acts as the "Body" (System environment).

C. Python is too hard for LLMs.

D. It reduces latency.

Advanced: Tools in AutoGen

You can register tools (functions) just like in CrewAI/LangChain.

```
from autogen import register_function # Define simple calculator  def calculator(a: int, b: int) →
int: return a + b # Register it to the agents  register_function( calculator, caller=assistant,
executor=user_proxy, description="Calculator tool" )
```

Hands-On Lab

What is async and wait?

Python normally runs code **one step at a time**, line by line.

But some things take time:

- calling an LLM API
- waiting for a response
- streaming output
- running multiple agents “in parallel”

If we wait the traditional (blocking) way, Python **stops everything** until the slow task finishes.

`async` & `await` let Python **continue doing other work while waiting**.

Why AutoGen uses async?

AutoGen 0.4 is built for **event-driven, multi-agent conversations**, where:

- Agents talk back and forth
- Some steps take long (LLM calls, code execution)
- Streaming responses arrive slowly (token-by-token)

Async allows AutoGen to:

- handle multiple agent messages concurrently
- cancel long-running tasks
- stream partial output
- keep the UI responsive (e.g., Colab or Streamlit)

Day 10 Summary

- **AutoGen** treats coding as a conversation.
- **UserProxy** allows for safe(r) code execution and human intervention.
- It excels at **iterative debugging** and coding tasks compared to other frameworks.

Use in Capstone?

Use AutoGen if...

Your team is building a "Coding Assistant" or "Data Analyst" that needs to generate charts dynamically.

Stick to CrewAI if...

Your workflow is content generation, research, or standard RAG where code execution isn't the main focus.