AutoGen, AG2, and Semantic Kernel: Complete Guide

🚀 Deep Dive into AutoGen (0.2 & 0.4), AG2, and Semantic Kernel — Exploring Agent Architecture, Chats using all Frameworks, and More 🤖

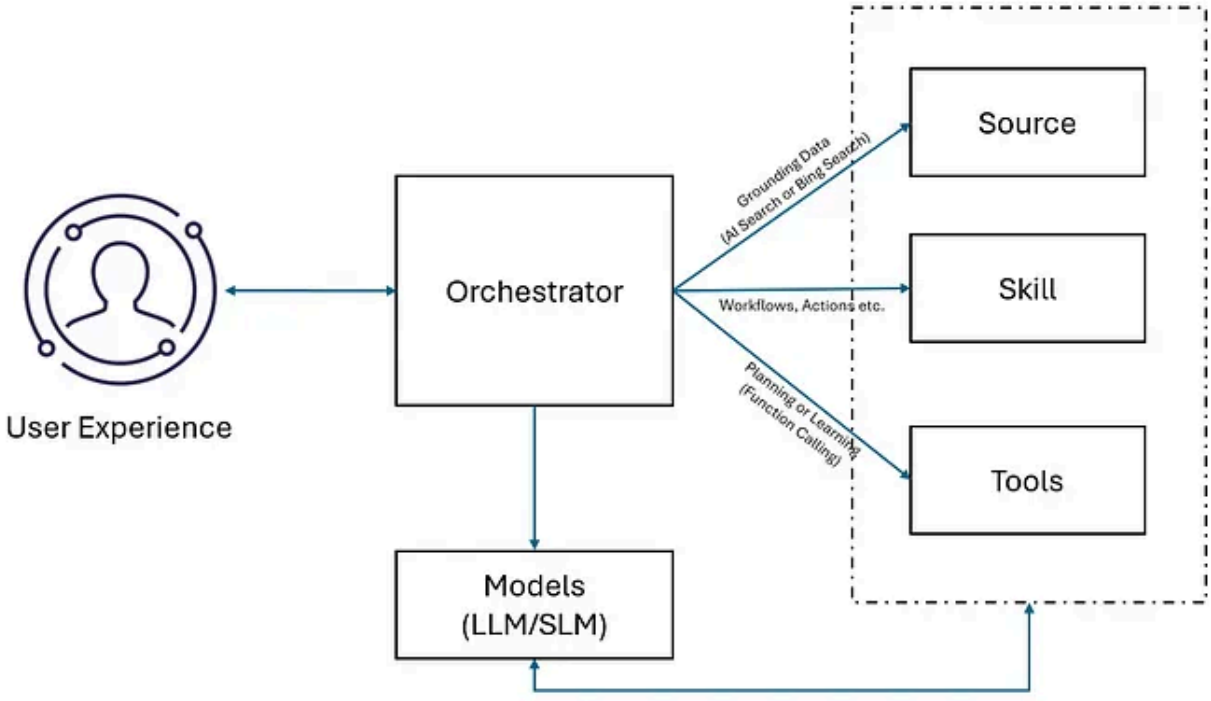
Naveen Krishnan
Published in Towards AI
17 min read · Jan 24, 2025

Image generated using Microsoft Designer

1. Introduction

As AI (Artificial Intelligence) is evolving 🌍 rapidly, developers and organization have started leveraging AI agents to build autonomous systems 🤖. AI agents are **independent, goal-oriented** which can perform tedious tasks autonomously. It can

sense, reason, act, and adapt to the environment, which helps us in automating tasks. Agent architecture 🏗️ comprises of **Orchestrator, Models, Source, Skill and Tools.**



Standard high-level Agent Architecture. Image Source—Author

2. About the Frameworks

There are different frameworks available, and each has its own characteristics. In this blog we will mainly focus on AutoGen (which I actually first used to explore agents), AG2 and Semantic Kernel which is offering unique features and capabilities. We also explore different chat types these frameworks support along with examples and some guidance on selecting the most suitable framework for your specific scenarios.

⚠️ **Note:** This blog is **not** about comparing which framework is the "best" 😉 — instead, it's about understanding their strengths and use cases.

Let's see about these frameworks at high level: 🔎

2.1 AutoGen:

The initial version of AutoGen framework v0.2 was very popular in agentic technologies, but there were few struggles in terms of architectural constraints, limited debugging and invention capabilities. It needed stronger observability and control, more flexible multi-agent collaboration patterns, and reusable components.

2.2 AutoGen v0.4:

This version was out few days back (Jan 14th) which got robust and extensible features from feedback provided by both external and internal Microsoft resources. Following are the few important new features:

✅ **Asynchronous Messaging:** Agents can now communicate through asynchronous messages, with both event-driven and request/response interaction patterns.

✅ **Modular & Etensible:** We can easily customize systems with pluggable components, including custom agents, tools, memory, and models. Also, we can now build proactive and long-running agents using event-driven patterns.

✅ **Observability and debugging:** This is one of the most needed features, built-in metric tracking, message tracing, and debugging tools provide monitoring and control over agent interactions and workflows.

Each layer in the framework has clearly divided responsibilities and build on top of layers below.

- **Core API:** It implements message passing, event-driven agents, and local and distributed runtime for flexibility and power.
- **AgentChat API:** This is built on top of Core API, and it is similar to v0.2, a multi-agent patterns such as two-agent chat or group chats 👥 .
- **Extensions API:** This enables first- and third-party extensions continuously expanding framework capabilities. It supports specific implementation of LLM clients (e.g., OpenAI, AzureOpenAI), and capabilities such as code execution.

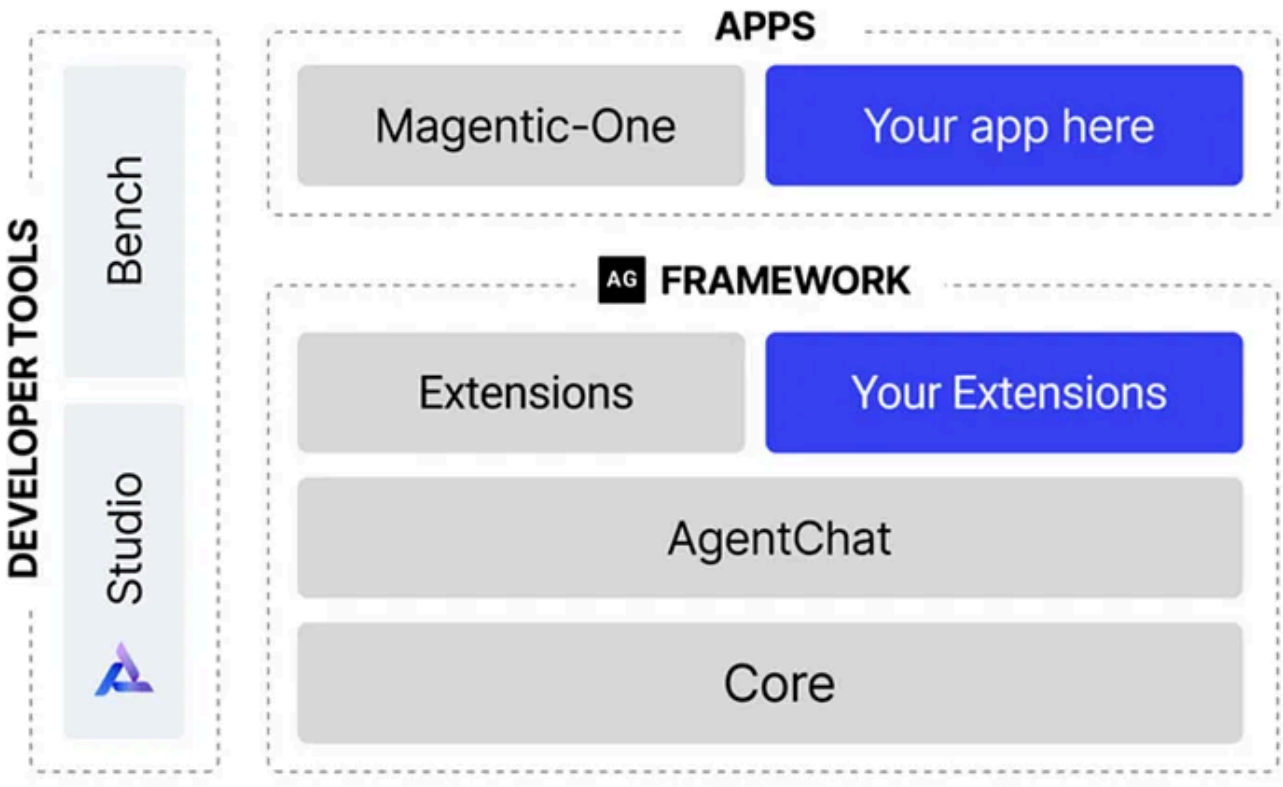


Image Soure: microsoft.com/blogs

In addition to the framework, AutoGen 0.4 also includes upgraded programming tools and applications.

✅ **AutoGen Bench:** Using this we can how benchmark the agents by measuring and comparing performance across tasks and environments.

✅ **AutoGen Studio:** This is completely rebuilt on v0.4 and now it enables rapid prototyping of AI agents along with several new capabilities and here are few important ones: **real-time agent updates** with asynchronous action streams, **mid-execution control** to pause, redirect, and adjust teams, and **message flow visualization** for clearer communication insights. Its **drag-and-drop** interface simplifies agent team design and configuration.

✅ **Magentic-One:** A new general multi-agent application to solve web and file-based tasks across various domains. This tool marks a significant step toward creating agents capable of completing tasks commonly encountered in both work and personal contexts.

2.3 Semantic Kernel:

It is more of a lightweight and open-source framework which allows us to build AI agents and integrate with latest AI models. It now supports 3 programming languages — C#, Python, or Java. It's an efficient middleware for enterprise-grade

production solutions. The Semantic Kernel Agent Framework provides a platform within the Semantic Kernel ecosystem for creating AI agents and incorporating agentic patterns into applications. Its architecture is built upon several key components:

✅ **Agent:** The abstract Agent class is the foundational structure for all agent types and the subclass KernelAgent associates directly with a Kernal object through which we can create the agents like the ChatCompletionAgent and OpenAI AssistantAgent. These agents can operate independently or collaborate within an Agen Chat.

✅ **Agent Chat:** This allows various agents to converse. It manages interactions within a chat environment, while the AgentGroupChat class extends this functionality by enabling multiple agents to collaborate across numerous interactions within the same conversation.

✅ **Agent Channel:** This class facilitates the participation of different agent types in an AgentChat, also it is essential for creating custom agents.

This Framework's design aligns with the core principles of the Semantic Kernel, ensuring consistency and ease of integration. By leveraging the Kernel, which serves as the central object driving AI operations, the framework supports the creation of intelligent, adaptable agents which are capable of managing multiple concurrent conversations and collaborating effectively within diverse scenarios.

2.4 AG2:

AG2 is **not a new framework** — it's a **rebranded** version of **AutoGen 0.2.34**, now continuing under the name **AG2**, with the latest version at **0.3.2**.
Community-driven with an **Open RFC process** ⏳. Maintains the **familiar agentic architecture** from early AutoGen versions. **Microsoft is taking AutoGen in a different direction**, while **AG2 stays community-led**.

AG2's Roadmap & Features: 🌟 AG2 also has good futuristic view of AG2 studio, AG2 Marketplace and Scaling Tools. It has got 20k active builders, daily technical discussions, Open RFC process etc. It simplifies the orchestration, optimization, and automation of large language model (LLM) workflows, offering customizable and conversable agents that leverage advanced LLMs like GPT-4.

3. Implementation

In this section, we'll see **how to implement different types of chats** using:
✅ **AutoGen**

✅ AG2
✅ Semantic Kernel

Stay tuned as we explore **practical examples** and guide you through **choosing the right framework** for your AI applications! 🚀

3.1 AutoGen:

Here we will focus only on v0.4, if you are on v0.2 look for ways to migrate then you can follow the instructions here — migration guide.

SetUp:

```
pip install -U "autogen-agentchat" "autogen-ext[openai, azure]"
```

3.1.1 Simple Chat:

```
import asyncio
from autogen_agentchat.agents import AssistantAgent
from autogen_ext.models.openai import AzureOpenAIChatCompletionClient

az_model_client = AzureOpenAIChatCompletionClient(
    azure_deployment="gpt-4o",
    model="gpt-4o",
    api_version="2024-08-01-preview",
    azure_endpoint="https://<<your-azure-openai-endpoint-name>>.openai.azure.cc
    api_key="<<your-azure-openai-api-key>>", # For key-based authentication. `A
)

async def main() -> None:
    agent = AssistantAgent("assistant", az_model_client)
    print(await agent.run(task="How are you doing today?"))

asyncio.run(main())
```

Result:

```
(.venv) PS D:\Dev Work\Autogen-0.4> python .\00-simple-chat.py
D:\Dev Work\Autogen-0.4\.venv\Lib\site-packages\autogen_agentchat\agents\_assistant_agent.py:330: UserWarning: Resolv
ed model mismatch: gpt-4o-2024-08-06 != gpt-4o-2024-11-20. Model mapping may be incorrect.
  result = await self._model_client.create(
TaskResult(messages=[TextMessage(source='user', models_usage=None, content='How are you doing today?', type='TextMess
age'), TextMessage(source='assistant', models_usage=RequestUsage(prompt_tokens=42, completion_tokens=27), content="I'
m just a program, so I don't have feelings, but I'm here and ready to assist you! How can I help you today?", type='T
extMessage')], stop_reason=None)
```

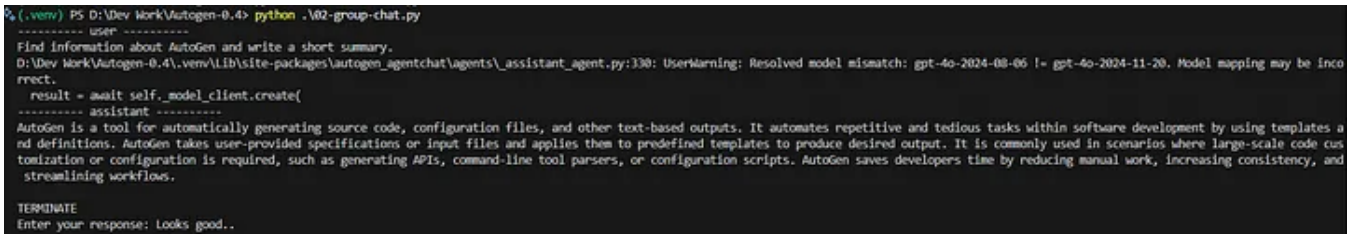
Execution result screenshot taken by Author

> *There are examples for OpenAI models, so in these examples I will use AzureOpenAI models and few other models from model catalogs.*

3.1.2 Group Chat:

```
import asyncio
from autogen_agentchat.agents import AssistantAgent, UserProxyAgent
from autogen_agentchat.conditions import TextMentionTermination
from autogen_agentchat.teams import RoundRobinGroupChat
from autogen_agentchat.ui import Console
from autogen_ext.models.openai import AzureOpenAIChatCompletionClient

async def main() -> None:
    az_model_client = AzureOpenAIChatCompletionClient(
        azure_deployment="gpt-4o",
        model="gpt-4o",
        api_version="2024-08-01-preview",
        azure_endpoint="https://<<your-azure-openai-endpoint-name>>.openai.azur
        api_key="<<your-azure-openai-api-key>>", # For key-based authentication
    )

    assistant = AssistantAgent("assistant",
        az_model_client,
        description="Agent that can help find the information",
        system_message="You are a helpful assistant that can help find the infc
    )
    user_proxy = UserProxyAgent("user_proxy")
    termination = TextMentionTermination("exit") # Type 'exit' to end the conve
    team = RoundRobinGroupChat([assistant, user_proxy], termination_condition=t
    await Console(team.run_stream(task="Find information about AutoGen and writ

asyncio.run(main())
```

Result:



Execution result screenshot taken by Author r

3.1.3 Tools:

```
from autogen_agentchat.agents import AssistantAgent
from autogen_agentchat.conditions import TextMentionTermination
from autogen_agentchat.teams import RoundRobinGroupChat
from autogen_agentchat.ui import Console
from autogen_core.tools import FunctionTool
from autogen_ext.models.openai import AzureOpenAIChatCompletionClient
import os
import time
import requests
from bs4 import BeautifulSoup
import asyncio

def bing_search(query: str, num_results: int = 2, max_chars: int = 500) -> list
    api_key = "<<your-bing-search-api-key>>"
    endpoint = "https://api.bing.microsoft.com/v7.0/search"

    if not api_key:
        raise ValueError("API key not found in environment variables")

    headers = {"Ocp-Apim-Subscription-Key": api_key}
    params = {"q": query, "count": num_results}

    response = requests.get(endpoint, headers=headers, params=params)

    if response.status_code != 200:
        print(response.json())
        raise Exception(f"Error in API request: {response.status_code}")

    results = response.json().get("webPages", {}).get("value", [])

    def get_page_content(url: str) -> str:
        try:
            response = requests.get(url, timeout=10)
            soup = BeautifulSoup(response.content, "html.parser")
            text = soup.get_text(separator=" ", strip=True)
            words = text.split()
            content = ""
            for word in words:
                if len(content) + len(word) + 1 > max_chars:
                    break
```

```
                content += " " + word
            return content.strip()
        except Exception as e:
            print(f"Error fetching {url}: {str(e)}")
            return ""

    enriched_results = []
    for item in results:
        body = get_page_content(item["url"])
        enriched_results.append(
            {"title": item["name"], "link": item["url"], "snippet": item["snipp
        )
        time.sleep(1)  # Be respectful to the servers

    return enriched_results


bing_search_tool = FunctionTool(
    bing_search, description="Search bing for information, returns results with
)


az_model_client = AzureOpenAIChatCompletionClient(
    azure_deployment="gpt-4o",
    model="gpt-4o",
    api_version="2024-08-01-preview",
    azure_endpoint="https://<<your-azure-openai-endpoint-name>>.openai.azure.cc
    api_key="<<your-azure-openai-api-key>>", # For key-based authentication. `A
)

bing_search_agent = AssistantAgent(
    name="bing_Search_Agent",
    model_client=az_model_client,
    tools=[bing_search_tool],
    description="Search Bing for information, returns top 2 results with a snip
    system_message="You are a helpful AI assistant. Solve tasks using your tool
)

report_agent = AssistantAgent(
    name="Report_Agent",
    model_client=az_model_client,
    description="Generate a report based on the search and results of stock ana
    system_message="You are a helpful assistant that can generate a comprehensi
)

team = RoundRobinGroupChat([bing_search_agent, report_agent], max_turns=3)

async def main() -> None:
    await Console(team.run_stream(task="Write a financial report on American ai

asyncio.run(main())
```

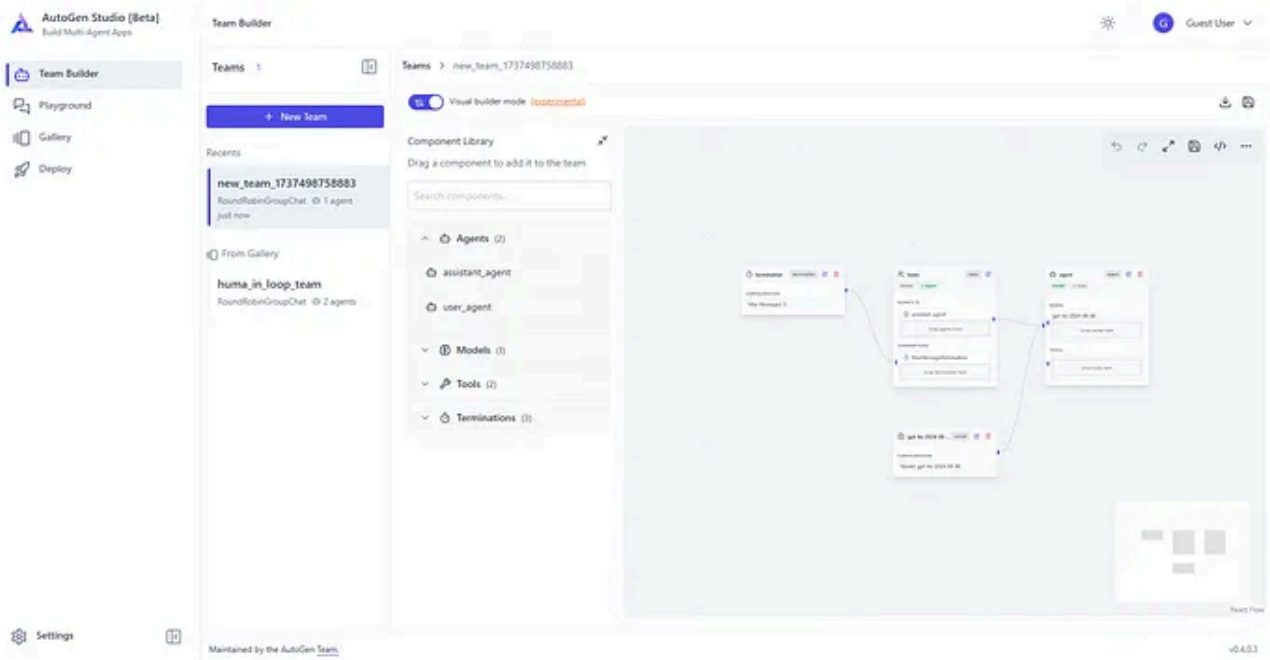
Result:

```
---------- user ----------
Write a financial report on American airlines
D:\Dev Work\Autogen-0.4\.venv\Lib\site-packages\autogen_agentchat\agents\_assis
  result = await self._model_client.create(
---------- bing_Search_Agent ----------
[FunctionCall(id='call_mqmQBz3ZFH6y8q8aKHbJi6Zv', arguments='{"query":"Americar
Some characters could not be decoded, and were replaced with REPLACEMENT CHARAC
---------- bing_Search_Agent ----------
[FunctionExecutionResult(content="[{'title': '2023 Annual Report on Form 10-K -
---------- bing_Search_Agent ----------
[{'title': '2023 Annual Report on Form 10-K - American Airlines', 'link': 'http
---------- Report_Agent ----------
**Financial Report: American Airlines**

**Overview:**
American Airlines Group Inc. (“American Airlines”), one of the largest airline

---

**Key Financial Highlights (2023):**
1. **Market Value and Stock Status:**
   - As of June 30, 2023, the aggregate market value of voting stock held by no
   - As of February 16, 2024, the company had a total of **654,756,816 shares o

2. **Revenue and Profitability:**
   - American Airlines' revenue benefited from strong recovery in travel demanc
   - Despite high demand, ongoing cost pressures, including rising fuel prices,

3. **Balance Sheet:**
   - The company has been managing a significant debt load, a legacy of the ste
   - Liquidity measures are improving, with American Airlines focusing on effic

4. **Operating Costs and Fuel Prices:**
   - Fuel prices remain a critical operational cost factor for American Airline

---

**Revenue Streams Analysis:**
1. **Domestic Market:**
   - The domestic market remains the key revenue driver for American Airlines,

2. **International Operations:**
   - Regions like Latin America and Europe have shown impressive growth traject

---

**Key Investments and Strategic Initiatives:**
1. **Fleet Modernization:**
```

```
    - American Airlines has invested significantly in modernizing its fleet, inc

2. **Sustainability Commitments:**
    - The aviation industry faces growing pressure to reduce carbon emissions. A

3. **Enhancing Customer Experience:**
    - A renewed focus on customer loyalty programs and premium cabin offerings h

---

**Challenges:**
1. **Debt Burden:**
    - The pandemic-era debt still looms large, with significant focus being dire

2. **Macroeconomic Factors:**
    - Volatile oil prices and inflation present risks to operating costs. Additi

3. **Competitive Environment:**
    - American Airlines operates in a fiercely competitive market and continues

---

**Stock Market Performance:**
- Over the last year, American Airlines' stock saw fluctuating trends, influenc
- Investors are closely monitoring the company’s ability to manage its debt lev

---

**Outlook for 2024:**
- **Tailwinds:**
  - Favorable consumer travel demand, especially for long-haul and leisure segm
  - Operational efficiencies and cost-saving programs are expected to improve p

- **Headwinds:**
  - Persistent macroeconomic uncertainties, especially in Europe and Asia, alor

---

**Conclusion:**
American Airlines is navigating a transitional period as it seeks to balance gr

TERMINATE
---------- bing_Search_Agent ----------
This report is based on data available up to today. For detailed information ar
```

3.1.4 AutoGen Studio:

```
pip install -U "autogenstudio"

autogenstudio ui --port 8080 --appdir ./my-app
```



Latest AutoGen Studio — screenshot taken by Author

3.2 Semantic Kernel:

All samples here use Azure OpenAI gpt-4o model.

Kernel Configuration is needed to successfully run these samples. I kept them in .env file at project root. For more details refer SemanticKernel Configuration.

```
AZURE_OPENAI_CHAT_DEPLOYMENT_NAME="gpt-4o"
AZURE_OPENAI_ENDPOINT="https://<<your-azure-openai-deployment-name>>.openai.azu
AZURE_OPENAI_API_KEY="<<your-azure-openai-key>>"
AZURE_OPENAI_API_VERSION="2024-08-01-preview"
```

3.2.1 Assistant: It allows function calling, file search and a code interpreter. Assistant Threads are used to manage the conversation state, similar to a Semantic Kernel Chat History.

```python
# Copyright (c) Microsoft. All rights reserved.
import asyncio
from typing import Annotated

from semantic_kernel.agents.open_ai import AzureAssistantAgent, OpenAIAssistant
from semantic_kernel.contents.chat_message_content import ChatMessageContent
from semantic_kernel.contents.utils.author_role import AuthorRole
from semantic_kernel.functions.kernel_function_decorator import kernel_functior
from semantic_kernel.kernel import Kernel

HOST_NAME = "Host"
HOST_INSTRUCTIONS = "Answer questions about the menu."


# Define a sample plugin for the sample
class MenuPlugin:
    """A sample Menu Plugin used for the concept sample."""

    @kernel_function(description="Provides a list of specials from the menu.")
    def get_specials(self) -> Annotated[str, "Returns the specials from the mer
        return """
        Special Soup: Clam Chowder
        Special Salad: Cobb Salad
        Special Drink: Chai Tea
        """

    @kernel_function(description="Provides the price of the requested menu item
    def get_item_price(
        self, menu_item: Annotated[str, "The name of the menu item."]
    ) -> Annotated[str, "Returns the price of the menu item."]:
        return "$9.99"


# A helper method to invoke the agent with the user input
async def invoke_agent(agent: OpenAIAssistantAgent, thread_id: str, input: str)
    """Invoke the agent with the user input."""
    await agent.add_chat_message(thread_id=thread_id, message=ChatMessageConter

    print(f"# {AuthorRole.USER}: '{input}'")

    async for content in agent.invoke(thread_id=thread_id):
        if content.role != AuthorRole.TOOL:
            print(f"# {content.role}: {content.content}")


async def main():
    # Create the instance of the Kernel
    kernel = Kernel()

    # Add the sample plugin to the kernel
    kernel.add_plugin(plugin=MenuPlugin(), plugin_name="menu")
```

```
    # Create the OpenAI Assistant Agent
    service_id = "agent"

    agent = await AzureAssistantAgent.create(
        kernel=kernel, service_id=service_id, name=HOST_NAME, instructions=HOST
    )

    thread_id = await agent.create_thread()

    try:
        await invoke_agent(agent, thread_id=thread_id, input="Hello")
        await invoke_agent(agent, thread_id=thread_id, input="What is the speci
        await invoke_agent(agent, thread_id=thread_id, input="What is the speci
        await invoke_agent(agent, thread_id=thread_id, input="Thank you")
    finally:
        await agent.delete_thread(thread_id)
        await agent.delete()


if __name__ == "__main__":
    asyncio.run(main())
```

Result:

```
(.venv) PS D:\Dev Work\Autogen-0.4> python .\07-semantic-assistant.py
# AuthorRole.USER: 'Hello'
# AuthorRole.ASSISTANT: Hi there! How can I assist you today?
# AuthorRole.USER: 'What is the special soup?'
# AuthorRole.ASSISTANT: The special soup today is Clam Chowder.
# AuthorRole.USER: 'What is the special drink?'
# AuthorRole.ASSISTANT: The special drink today is Chai Tea.
# AuthorRole.USER: 'Thank you'
# AuthorRole.ASSISTANT: You're welcome! If you have any other questions, feel free to ask. Enjoy your day!
```

Execution result screenshot taken by Author

3.2.2 Code Interpreter:

```
import asyncio

from semantic_kernel.agents.open_ai.azure_assistant_agent import AzureAssistant
from semantic_kernel.agents.open_ai.open_ai_assistant_agent import OpenAIAssist
from semantic_kernel.contents.chat_message_content import ChatMessageContent
from semantic_kernel.contents.utils.author_role import AuthorRole
from semantic_kernel.kernel import Kernel


AGENT_NAME = "CodeRunner"
```

```
AGENT_INSTRUCTIONS = "Run the provided code file and return the result."


# A helper method to invoke the agent with the user input
async def invoke_agent(agent: OpenAIAssistantAgent, thread_id: str, input: str)
    """Invoke the agent with the user input."""
    await agent.add_chat_message(thread_id=thread_id, message=ChatMessageConter

    print(f"# {AuthorRole.USER}: '{input}'")

    async for content in agent.invoke(thread_id=thread_id):
        if content.role != AuthorRole.TOOL:
            print(f"# {content.role}: {content.content}")


async def main():
    # Create the instance of the Kernel
    kernel = Kernel()

    # Define a service_id for the sample
    service_id = "agent"

    # Create the agent
    agent = await AzureAssistantAgent.create(
            kernel=kernel,
            service_id=service_id,
            name=AGENT_NAME,
            instructions=AGENT_INSTRUCTIONS,
            enable_code_interpreter=True,
    )

    thread_id = await agent.create_thread()

    try:
        await invoke_agent(
            agent,
            thread_id=thread_id,
            input="Use code to determine the values in the Fibonacci sequence t
        )
    finally:
        await agent.delete_thread(thread_id)
        await agent.delete()


if __name__ == "__main__":
    asyncio.run(main())
```

Result:

```
(.venv) PS D:\Dev Work\Autogen-0.4> python .\06-Semantic-assistant-tool-code-interpreter.py
# AuthorRole.USER: 'Use code to determine the values in the Fibonacci sequence that that are less then the value of 101?'
# AuthorRole.ASSISTANT: # Define a function to generate Fibonacci sequence below a given limit
def fibonacci_below(limit):
    sequence = []
    a, b = 0, 1
    while a < limit:
        sequence.append(a)
        a, b = b, a + b
    return sequence

# Generate Fibonacci sequence below 101
fib_sequence = fibonacci_below(101)
fib_sequence
# AuthorRole.ASSISTANT: The values in the Fibonacci sequence that are less than 101 are:
\[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89\]
```

Execution result screenshot taken by Author

3.2.3 File Search: This sample demonstrates how to create an OpenAI assistant using Azure OpenAI leverage the assistant's file search functionality.

```python
import asyncio
import os

from semantic_kernel.agents.open_ai.azure_assistant_agent import AzureAssistant
from semantic_kernel.agents.open_ai.open_ai_assistant_agent import OpenAIAssist
from semantic_kernel.contents.chat_message_content import ChatMessageContent
from semantic_kernel.contents.utils.author_role import AuthorRole
from semantic_kernel.kernel import Kernel


AGENT_NAME = "FileSearch"
AGENT_INSTRUCTIONS = "Find answers to the user's questions in the provided file


# A helper method to invoke the agent with the user input
async def invoke_agent(agent: OpenAIAssistantAgent, thread_id: str, input: str)
    """Invoke the agent with the user input."""
    await agent.add_chat_message(thread_id=thread_id, message=ChatMessageConter

    print(f"# {AuthorRole.USER}: '{input}'")

    async for content in agent.invoke(thread_id=thread_id):
        if content.role != AuthorRole.TOOL:
            print(f"# {content.role}: {content.content}")


async def main():
    # Create the instance of the Kernel
    kernel = Kernel()

    # Define a service_id for the sample
    service_id = "agent"

    # Get the path to the travelinfo.txt file
    pdf_file_path = os.path.join(os.path.dirname(os.path.realpath(__file__)), "
```

```
    # Create the agent configuration
    agent = await AzureAssistantAgent.create(
            kernel=kernel,
            service_id=service_id,
            name=AGENT_NAME,
            instructions=AGENT_INSTRUCTIONS,
            enable_file_search=True,
            vector_store_filenames=[pdf_file_path],
    )

    # Define a thread and invoke the agent with the user input
    thread_id = await agent.create_thread()

    try:
        await invoke_agent(agent, thread_id=thread_id, input="Who is the younge
        await invoke_agent(agent, thread_id=thread_id, input="Who works in sale
        await invoke_agent(agent, thread_id=thread_id, input="I have a customer
    finally:
        [await agent.delete_file(file_id) for file_id in agent.file_search_file
        await agent.delete_thread(thread_id)
        await agent.delete()


if __name__ == "__main__":
    asyncio.run(main())
```

Result:

```
(.venv) PS D:\Dev Work\Autogen-0.4> python .\08-semantic-kernel-file-search.py
# AuthorRole.USER: 'Who is the youngest employee?'
# AuthorRole.ASSISTANT: The youngest employee is Teodor Britton, born on January 9, 1997【4:0†employees.pdf】.
# AuthorRole.USER: 'Who works in sales?'
# AuthorRole.ASSISTANT: The employees who work in sales are:

1. Mariam Jaslyn - Sales representative
2. Hicran Bea - Sales manager
3. Angelino Embla - Sales representative【8:0†employees.pdf】.
# AuthorRole.USER: 'I have a customer request, who can help me?'
# AuthorRole.ASSISTANT: For customer requests, the best contacts would be the sales team. The employees in sales are:

1. Mariam Jaslyn - Sales representative
2. Hicran Bea - Sales manager
3. Angelino Embla - Sales representative

You can reach out to any of these individuals for assistance with customer requests【8:0†employees.pdf】.
```

| Execution result screenshot taken by Author

3.3 AG2:

You can install and run AG2 in Docker, here I used my local.

```
pip install ag2
```

Before we start, we get the config file ready so agents can work with the models.

```
[
    {
        "model": "gpt-4o",
        "api_key": "<<your-azure-openai-key>>",
        "base_url": "https://<<your-azure-openai-resource>>.openai.azure.com/",
        "api_type": "azure",
        "api_version": "2024-08-01-preview"
    }
]
```

3.3.1 Two Agents Chat: This initiates an automated chat between the two agents to solve a simple task. It created a coding directory, placed the python file.

```
from autogen import AssistantAgent, UserProxyAgent, config_list_from_json
config_list = config_list_from_json(env_or_file="OAI_CONFIG_LIST.json")
# You can also set config_list directly as a list, for example, config_list = [
assistant = AssistantAgent("assistant", llm_config={"config_list": config_list}
user_proxy = UserProxyAgent("user_proxy", code_execution_config={"work_dir": "c
user_proxy.initiate_chat(assistant, message="Plot a chart of NVDA and TESLA stc
```

```
 user_proxy [0m (to assistant):

Plot a chart of NVDA and TESLA stock price change YTD.

--------------------------------------------------------------------------------
 assistant [0m (to user_proxy):

To plot a chart of NVIDIA (NVDA) and Tesla (TSLA) stock price change year-to-da

1. Collect historical stock price data for NVDA and TSLA for the current year.
2. Calculate the YTD percentage change for both stocks.
3. Plot the YTD percentage change on a chart.
```

```
We can use Python along with the `pandas`, `yfinance`, and `matplotlib` librari

Here's the complete script to do this:

```python
# filename: nvda_tsla_ytd_plot.py
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
from datetime import datetime

# Set the start and end dates
start_date = f"{datetime.now().year}-01-01"
end_date = datetime.now().strftime("%Y-%m-%d")

# Fetch historical stock data
nvda_data = yf.download('NVDA', start=start_date, end=end_date)
tsla_data = yf.download('TSLA', start=start_date, end=end_date)

# Calculate YTD price change percentage
nvda_data['YTD Change %'] = (nvda_data['Close'] / nvda_data['Close'].iloc[0] -
tsla_data['YTD Change %'] = (tsla_data['Close'] / tsla_data['Close'].iloc[0] -

# Plot the data
plt.figure(figsize=(10, 6))
plt.plot(nvda_data.index, nvda_data['YTD Change %'], label='NVDA YTD Change %')
plt.plot(tsla_data.index, tsla_data['YTD Change %'], label='TSLA YTD Change %')
plt.xlabel('Date')
plt.ylabel('Year-To-Date Change (%)')
plt.title('NVDA and TSLA YTD Price Change')
plt.legend()
plt.grid(True)
plt.show()
```

Steps:
1. Save the above code into a file named `nvda_tsla_ytd_plot.py`.
2. Execute the script to fetch the stock prices, calculate the YTD changes and

Please execute the script and provide the output.

```

Chart comparing 2 stocks, output generated by the Agent.—Author

3.3.2 Multi-Agent—Group Chat: In this sample, group of agents work together to create a snake game

```
from autogen import AssistantAgent, UserProxyAgent, config_list_from_json
config_list = config_list_from_json(env_or_file="OAI_CONFIG_LIST.json")

llm_config={
    'temperature':0,
    'config_list':config_list,
    'timeout':60,
}

coder=autogen.AssistantAgent(
    name='AIAssistant',
    system_message="You are a coder. You will write the code for the project. y
    llm_config=llm_config

)

userProxy=autogen.UserProxyAgent(
    name='UserProxy',
    system_message="You are an executor. you will help execute the code writter
    human_input_mode="NEVER",
    code_execution_config={
        "work_dir": "coding",
        "use_docker":False
    }
)

planner=autogen.AssistantAgent(
```

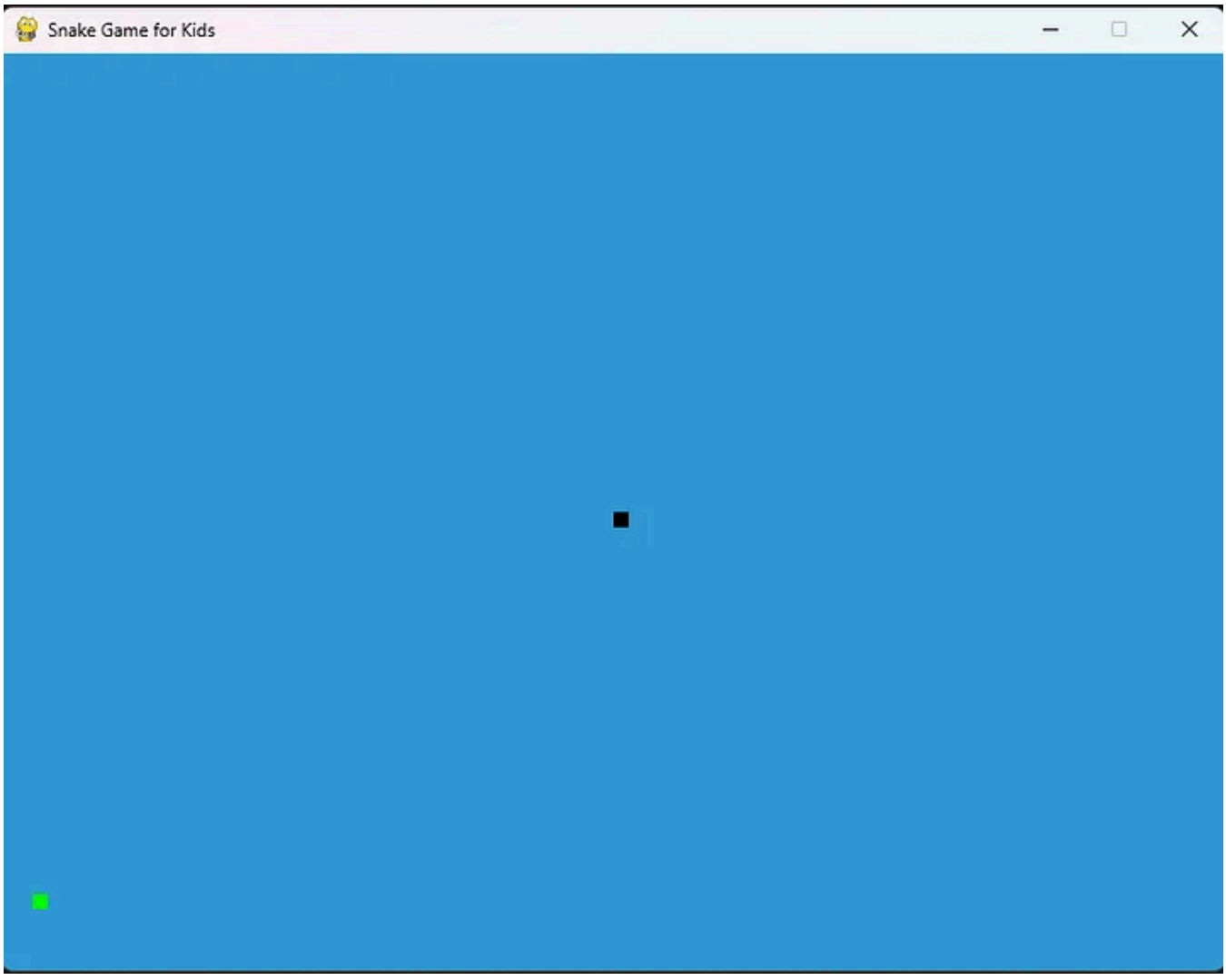
```
    name='Planner',
    system_message="You are a planner. You will help plan the project and make
    llm_config=llm_config
)

group_chat=autogen.GroupChat(
    agents=[userProxy, coder, planner],
    messages=[],
    max_round=15
)

groupChatManager=autogen.GroupChatManager(groupchat=group_chat,llm_config=llm_c

userProxy.initiate_chat(
    groupChatManager,
    message="I want to play a snake game. create a snake game. It's for 6 year
)
```

Result: I just pasted only the output as the conversations are too big.



Snake Game, code generated by Coder Agent and executed by UserProxy Agent. Screenshot taken by the Author

4. Conclusion 🎯

In conclusion, **AG2, Semantic Kernel,** and **AutoGen** all three frameworks offer both common features and advantages for specific AI development needs.

AG2 — Is good for who is seeking a **community-driven framework** more focused on **agent orchestration and collaboration.**

Semantic Kernel is best for who are **integrating AI functionalities** into your existing **enterprise apps** as it supports **C#, Python and Java.**

AutoGen is majorly for **innovators** needing to **build intelligent, autonomous agents** capable of **complex decision-making.**

By understanding the **unique strengths** of each framework, you can now **choose the right one** ✅ that best fits your **AI projects and goals** 🚀!

ChatGPT | AI | Data Science | Machine Learning | Technology