



Open-Source Deep Dive

# OpenAI Swarm

October 23, 2024





shyamal ✓

@shyamalanadkat



introducing swarm: an experimental framework for building, orchestrating, and deploying multi-agent systems. 🐝

## openai/swarm



Educational framework exploring ergonomic, lightweight multi-agent orchestration. Managed by OpenAI Solution team.

👤 15

Contributors

🗨️ 7

Issues

★ 13k

Stars

🍴 1k

Forks



GitHub - openai/swarm: Educational framework exploring ergonomic, lightweight multi...

From github.com

Last edited 4:51 PM · Oct 11, 2024 · **818K** Views

# Who Are We?

---



**John Gilhuly**  
Developer Advocate



**Xander Song**  
Engineer

# Agenda

- **Swarm: What and why?**
- **Concepts and Implementation**
- **Example and Demo**
- **Discussion**
  - control flow
  - comparisons with other frameworks

# What is Swarm?

---



- multi-agent Python orchestration framework
  - open-source
  - unpublished
- lightweight (few abstractions)
- Python library intended for educational purposes

```
pip install git+ssh://git@github.com/openai/swarm.git
```

# Why Swarm?

- refreshingly simple
- opinions and lessons can be adopted whether you use Swarm or not

# Concepts

---

- **Agents have:**
  - **tools** they can use (defined as Python functions)
  - **instructions** they must follow (a system prompt)
- Control flow is a while loop
- Exactly one agent is in control at all times
- Some tools **hand off** control from one agent to another

```
class Agent(BaseModel):  
    name: str = "Agent"  
    model: str = "gpt-4o"  
    instructions: Union[str, Callable[[], str]] = "You are a helpful agent."  
    functions: List[AgentFunction] = []  
    tool_choice: str = None  
    parallel_tool_calls: bool = True
```

# Tools

```
Tool #4
1 {
2   "type": "function",
3   "function": {
4     "name": "transfer_to_triage",
5     "description": "Call this function when a user needs
to be transferred to a differnt agent and a different
policy.\n  For instance, if a user is asking about a
topic that is not handled by the current agent, call
this function.\n  ",
6     "parameters": {
7       "type": "object",
8       "properties": {},
9       "required": []
10    }
11  }
12 }
```

```
def transfer_to_triage():
    """Call this function when a user needs to be transferred to a differnt agent and a different policy.
    For instance, if a user is asking about a topic that is not handled by the current agent, call this
    function.
    """
    return triage_agent
```

- Python functions automatically translated to JSON schema
- Functions can execute arbitrary logic
- Some functions *hand off* control to other agents

# Airline Customer Support

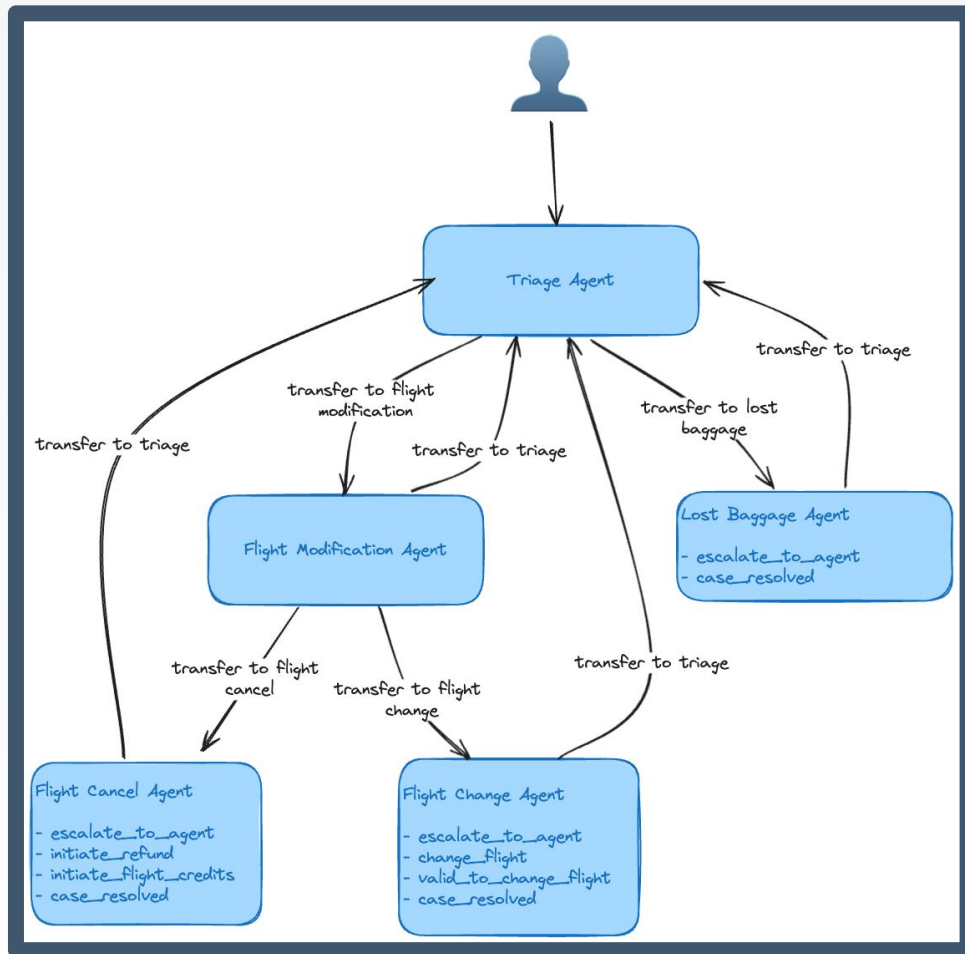
five agents:

- triage
- flight modification
- flight cancel
- flight change
- lost baggage

tasks:

- top-level agents route requests
- lower-level agents have task-specific tools

[source code](#)



# Entrypoint

---

- Pass context into `run_demo_loop`

```
from configs.agents import *
from swarm.repl import run_demo_loop

context_variables = {
    "customer_context": """Here is what you know about the customer's details:
1. CUSTOMER_ID: customer_12345
2. NAME: John Doe
3. PHONE_NUMBER: (123) 456-7890
4. EMAIL: johndoe@example.com
5. STATUS: Premium
6. ACCOUNT_STATUS: Active
7. BALANCE: $0.00
8. LOCATION: 1234 Main St, San Francisco, CA 94123, USA
""",
    "flight_context": """The customer has an upcoming flight from LGA (Laguardia) in NYC to LAX in Los Angeles.
The flight # is 1919. The flight departure date is 3pm ET, 5/21/2024."""
}

if __name__ == "__main__":
    run_demo_loop(triage_agent, context_variables=context_variables, debug=True)
    parallel_tool_calls: bool = True
```

# Control Flow

---

Control flow is a while loop

Each iteration:

- accepts user input
- invokes LLM
- invokes tools (if called)
- hands off control to another agent (if handoff tool called)

```
def run_demo_loop(
    starting_agent, context_variables=None, stream=False, debug=False
) -> None:
    client = Swarm()
    print("Starting Swarm CLI 🐛")

    messages = []
    agent = starting_agent

    while True:
        user_input = input("\033[90mUser\033[0m: ")
        messages.append({"role": "user", "content": user_input})

        response = client.run(
            agent=agent,
            messages=messages,
            context_variables=context_variables or {},
            stream=stream,
            debug=debug,
        )

        if stream:
            response = process_and_print_streaming_response(response)
        else:
            pretty_print_messages(response.messages)

        messages.extend(response.messages)
        agent = response.agent
```

# Demo

---

# Takeaways

- **KISS**
- **syntactic sugar for tool calling**
- **multi-agent makes prompts and routing simple**

# Comparing Swarm to Other Frameworks

```
#Swarm
agent = Agent(
    name='Data Analyst',
    instructions="""You're a data analyst at a large company.
    You're responsible for analyzing data and providing insights
    to the business.
    You're currently working on a project to analyze the
    performance of our marketing campaigns."""
    functions=[my_tool1, my_tool2],
)

#CrewAI
agent = Agent(
    role='Data Analyst',
    goal='Extract actionable insights',
    backstory="""You're a data analyst at a large company.
    You're responsible for analyzing data and providing insights
    to the business.
    You're currently working on a project to analyze the
    performance of our marketing campaigns."""
    tools=[my_tool1, my_tool2],
)

#Autogen
agent = ConversableAgent(
    "Data Analyst",
    system_message="""You're a data analyst at a large company.
    You're responsible for analyzing data and providing insights to the business.
    You're currently working on a project to analyze the performance of our marketing campaigns."""
    llm_config={"config_list": [{"model": "gpt-4", "temperature": 0.9, "api_key":
os.environ.get("OPENAI_API_KEY")}]},
    human_input_mode="NEVER",
    # tools are registered separately
)
```

```
#Swarm
def my_tool(question: str) -> str:
    """Clear description for what this tool is useful for, your agent will need this information to use
    it."""
    # Function logic here
    return "Result from your custom tool"

#CrewAI
from crewai_tools import tool
@tool("Name of my tool")
def my_tool(question: str) -> str:
    """Clear description for what this tool is useful for, your agent will need this information to use
    it."""
    # Function logic here
    return "Result from your custom tool"

#Autogen
def my_tool(question: Annotated[str, "the unchanged question from the user"]) -> str:
    """Clear description for what this tool is useful for, your agent will need this information to use
    it."""
    # Function logic here
    return "Result from your custom tool"
```

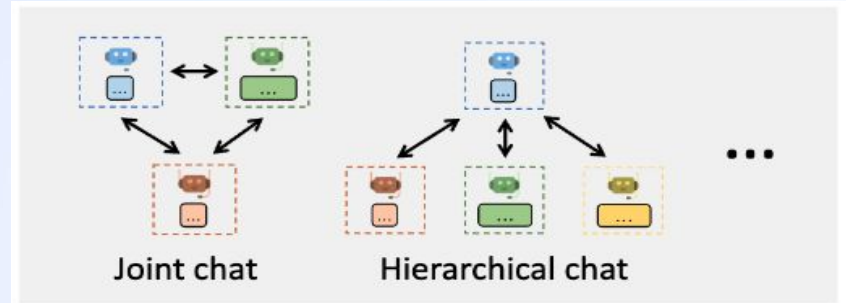
# Comparing Swarm to Other Frameworks

**Control flow** is the largest difference between the frameworks

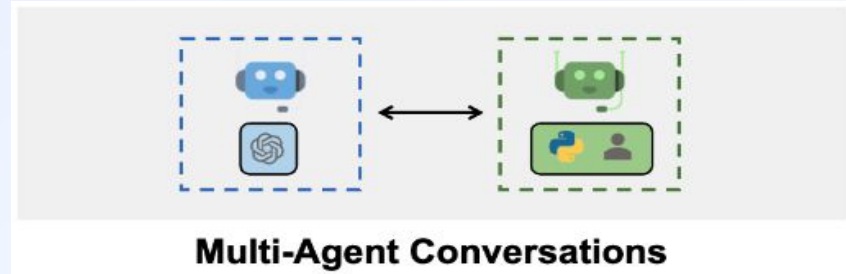
Autogen and CrewAI both use predefined flows:

- Free conversation
- GroupChats
- Hierarchical structures
- Pipelines

Swarm doesn't define control flow



**Flexible Conversation Patterns**



**Multi-Agent Conversations**

# Discussion topics

---

- Are we converging on a common definition of what a single agent in a multi-agent workflow looks like?
- How can we make it easy to mix manual and auto-instrumentation to explore a new library like Swarm?



# Building an Agent or Assistant: Real World Examples



## Our New Agents Series

Join us for a free 5 week series on building an agent or assistant!

Each week we'll unpack a real-life agent deployed in production.

**Register here:**

[bit.ly/build-an-agent](https://bit.ly/build-an-agent)