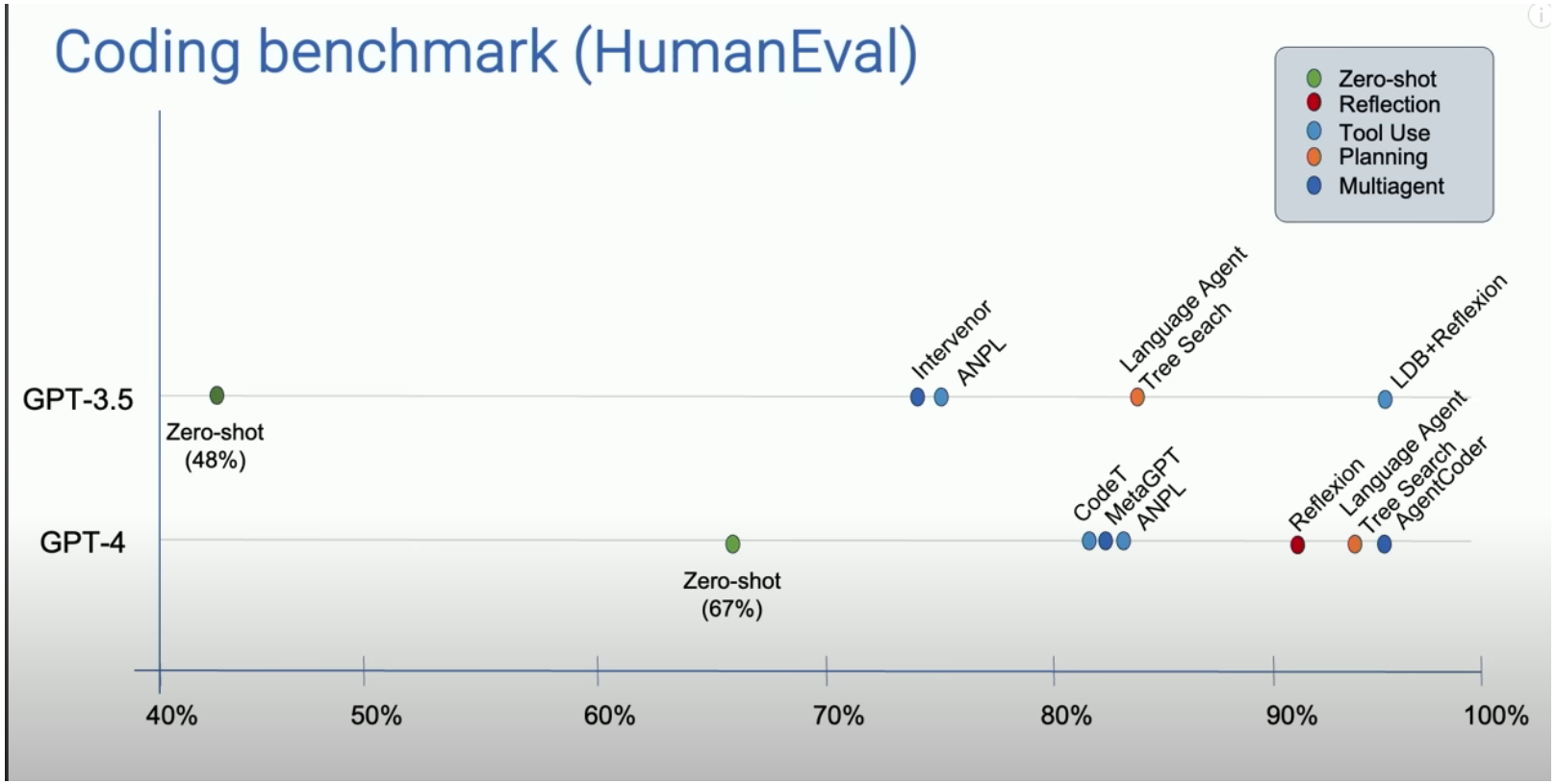
AI Agentic Workflow using

Autogen

Data/AI Architect Hakjun Min

AI Agentic workflow

Andrew Ng's Vision for AI's Future: Unlocking Agentic Workflows



AI Agentic pattern

Agentic Reasoning Design Patterns

1. Reflection
2. Tool use

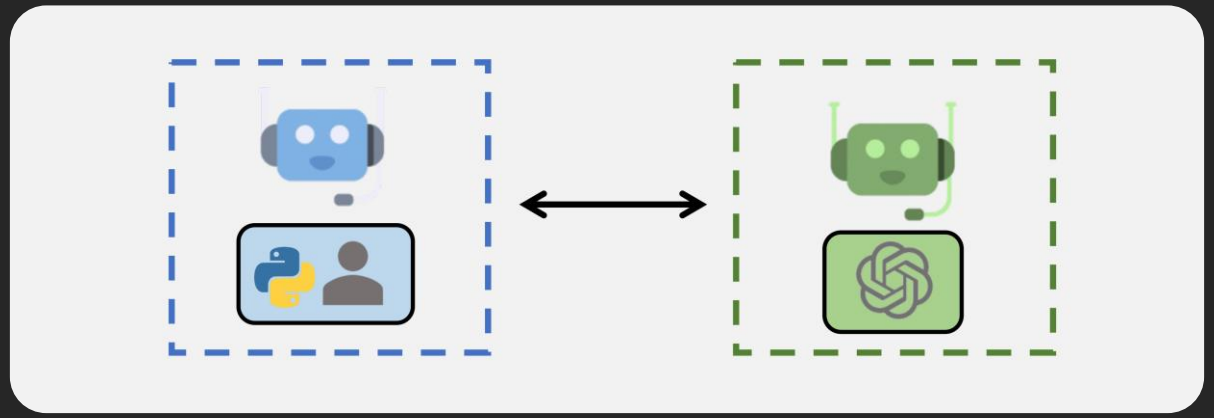
robust technology

3. Planning
4. Multi-agent collaboration

emerging technology

AutoGen

A programming framework
for agentic AI



Initially developed in FLAML (Nov 2022)
Spined off to a standalone repo (October 2023)

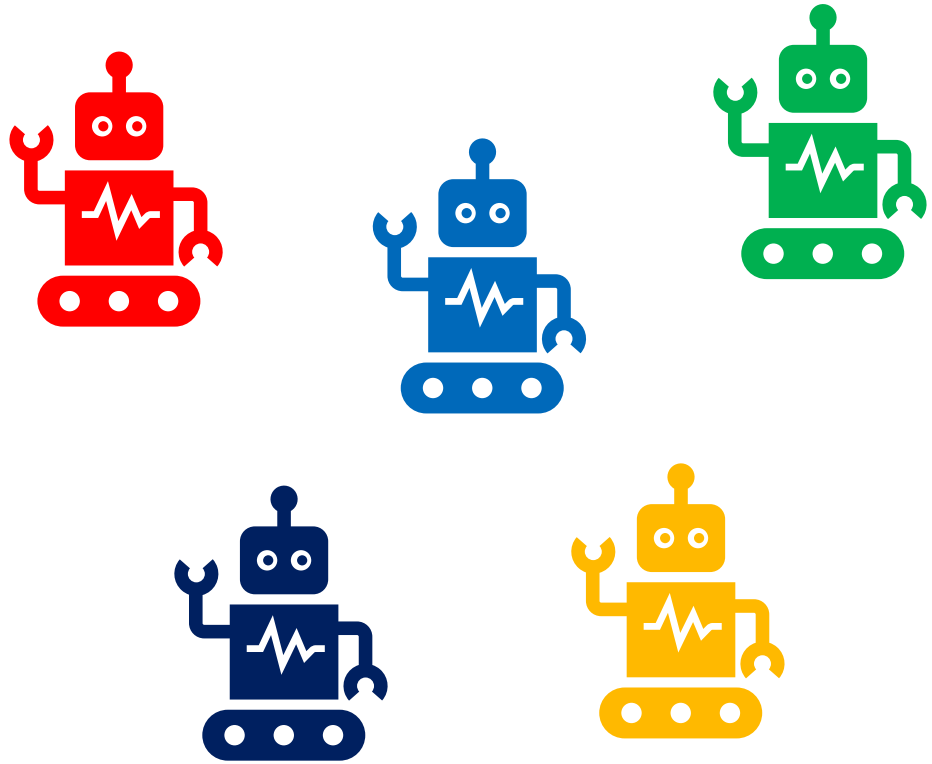
aka.ms/autogen

Results: Test | Results: Validation

Model name	Average score (%)	Level 1 score (%)	Level 2 score (%)	Level 3 score (%)	organisation	Model family
Multi-Agent Experiment v0.1	32.33	47.31	28.93	14.58	MSR AI Frontiers	GPT-4-turbo
FRIDAY	24.25	40.86	20.13	6.12	OS-Copilot	GPT-4-turbo
FRIDAY without learning	21.59	36.56	17.61	6.12	OS-Copilot	GPT-4-turbo
GPT4 + manually selected plu	14.6	30.3	9.7	0	GAIA authors	GPT4
Clarity v1	14.05	23.91	10.69	6.25		GPT4
Warm-up Act	12.96	22.58	10.69	2.04		GPT-4-Turbo
stealth3	9.3	18.28	6.92	0		
stealth2	8.97	17.2	6.29	2.04		
stealth	8.64	17.2	6.29	0		
GPT4 Turbo	6.67	9.68	6.92	0	GAIA authors	GPT4
AutoGPT4	5	15.05	0.63	0	AutoGPT	GPT4 + AutoGPT
GPT4	4	9.68	1.89	0	GAIA authors	GPT4

Autonomous Agents

- Copilot is a special agent
  - Human in the loop
  - Natural language interface
  - One step at a time
- Agents can be autonomous
  - Self-directed, human optional
  - Structured message interface
  - Execute workflow end-to-end
  - **Collaborate with other agents**
- Agents can be the "system two" of a copilot

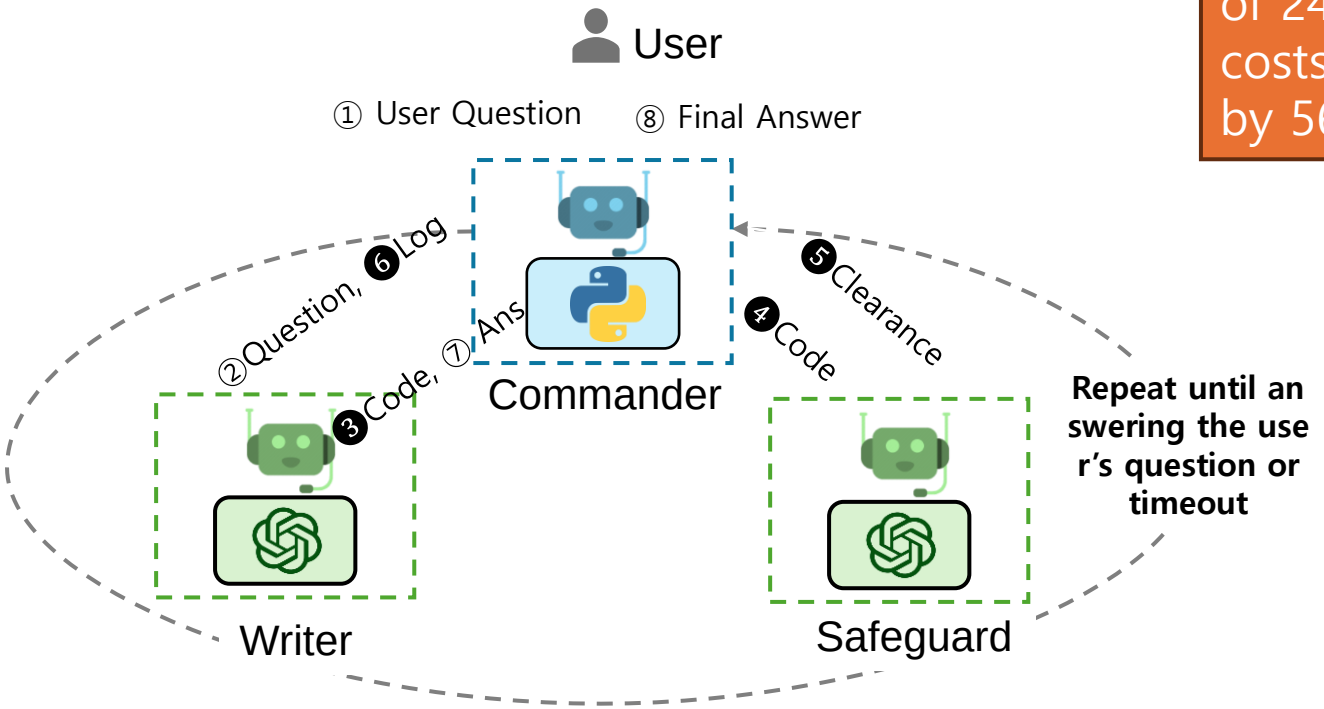


Demo

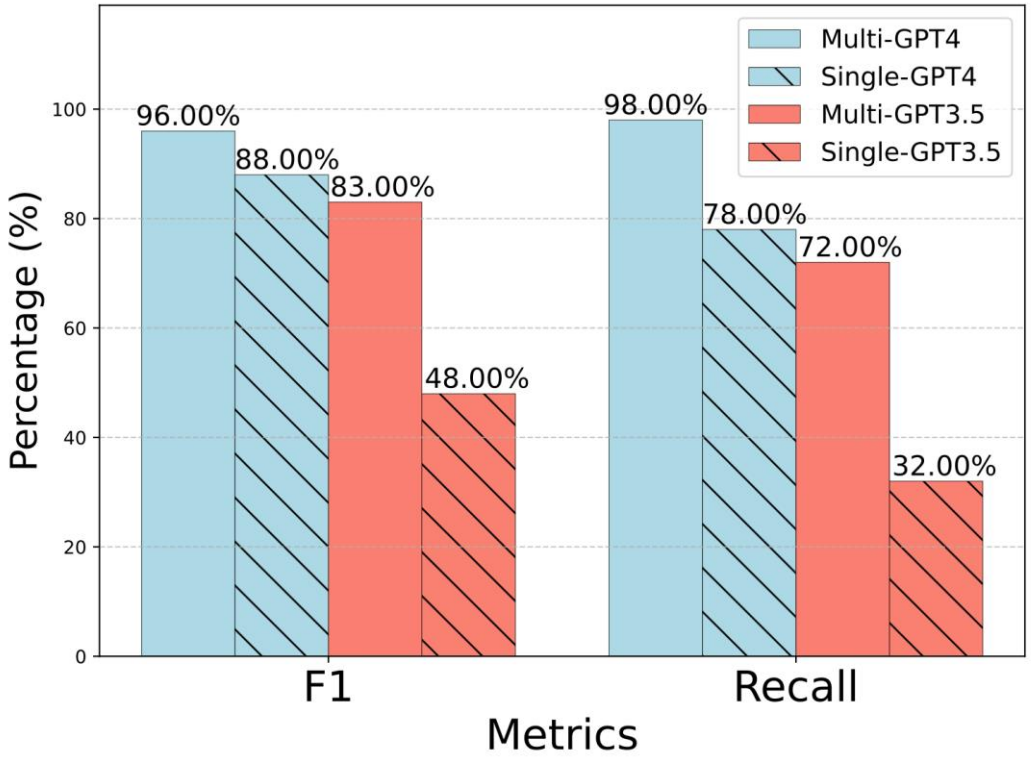
https://github.com/HakjunMIN/autogen_sample

Safeguard agent

What if the roasting cost is increased by 5% because of the potential salary increase?

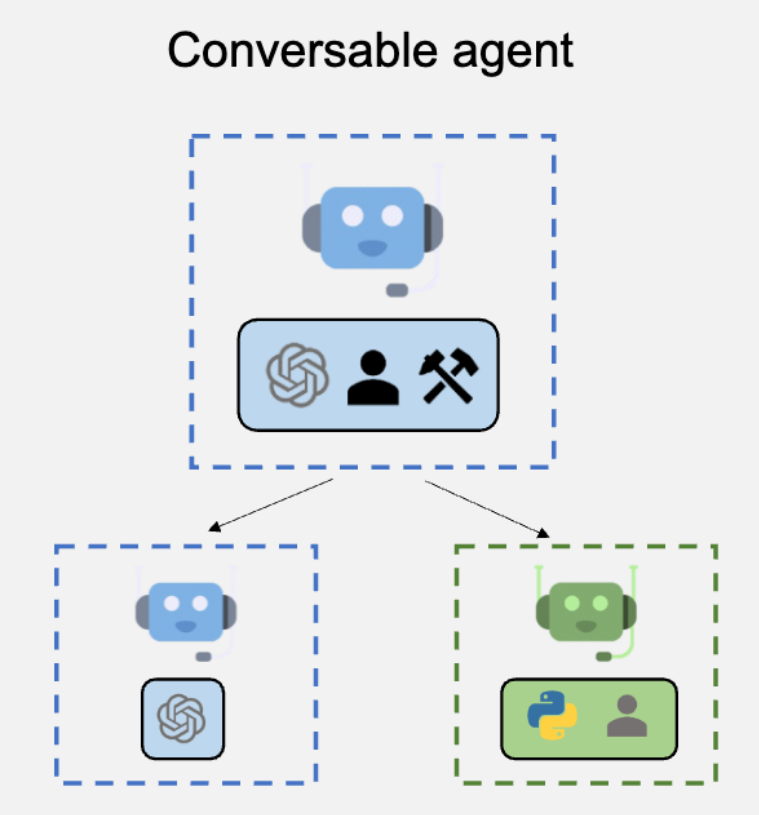


After increasing the roasting costs by 5%, I've re-calculated the optimal coffee distribution solution for the supply chain. As a result, the new optimal total cost has become 2526.5, compared to the initial cost of 2470.0. This indicates that the increased roasting costs have caused the overall distribution cost to rise by 56.5 units.

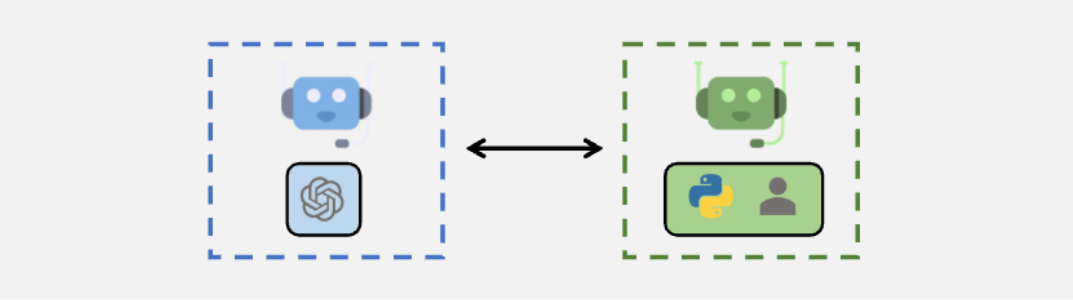


Autogen Basic Concept

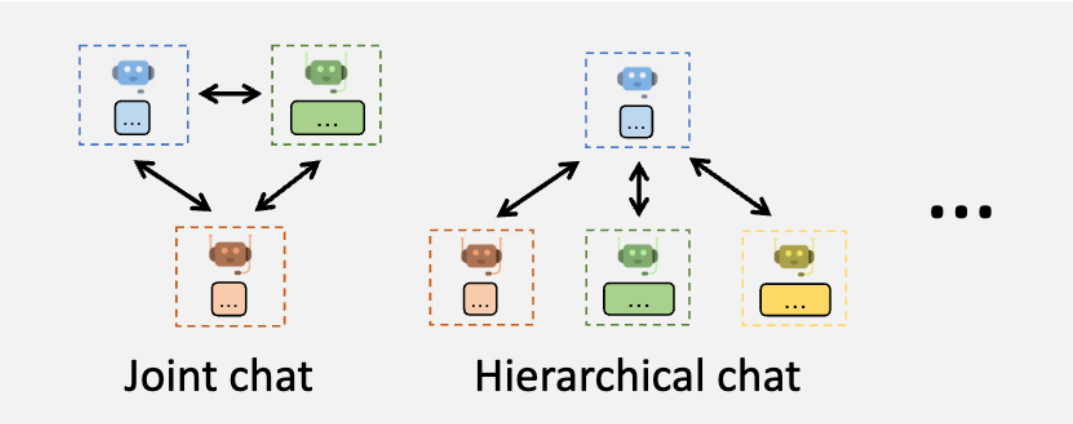
AutoGen Framework for Agents



Agent Customization



Multi-Agent Conversations



Flexible Conversation Patterns

Open-Source
Framework & Samples

Customizable
Conversable Agents, LLMs

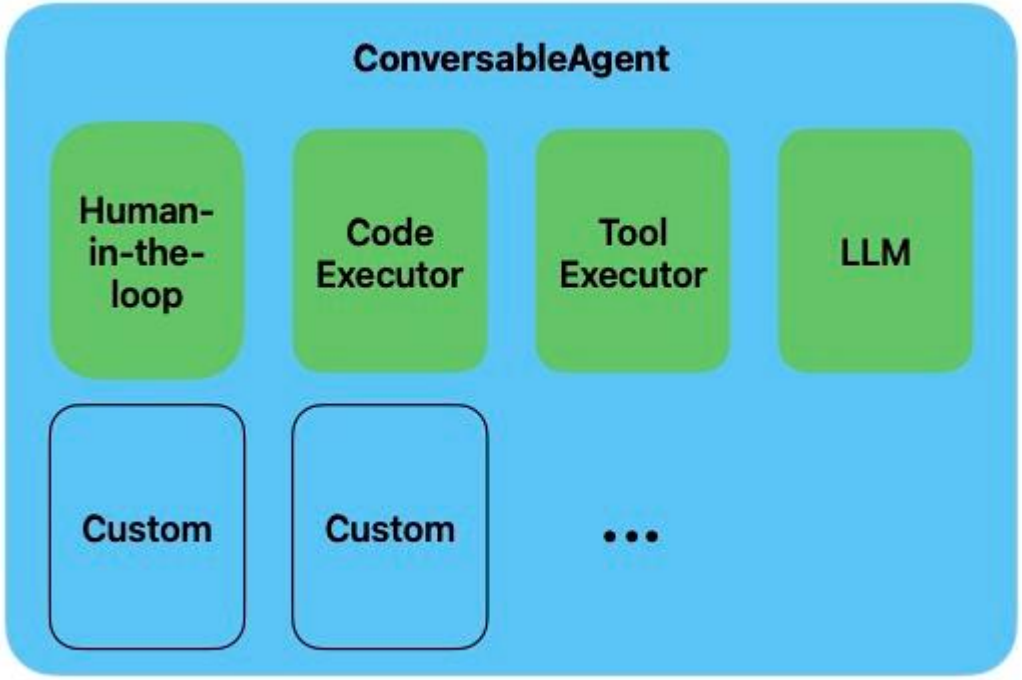
Research-Driven
Tools & Patterns

No-Code and Code-First
Development

Docs: **https://aka.ms/autogen/website**

Discord: **https://aka.ms/autogen/discord**

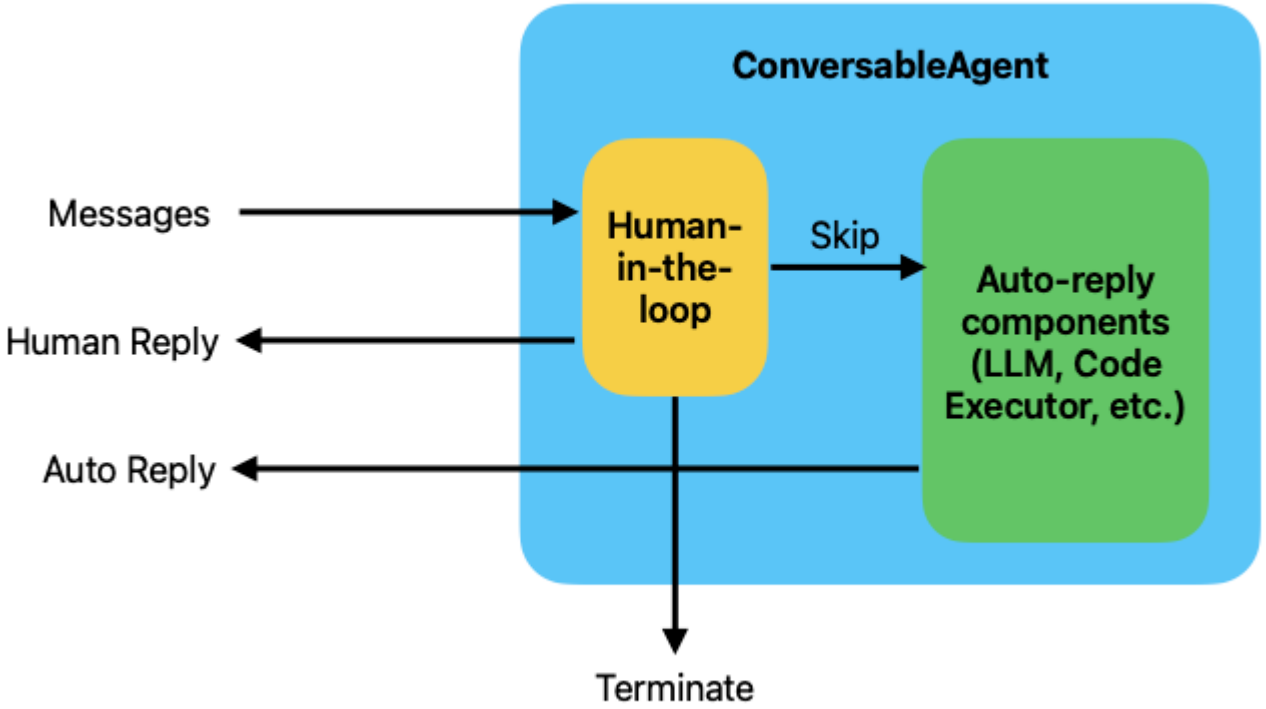
Agents



An agent is an entity that can send and receive messages to and from other agents in its environment.

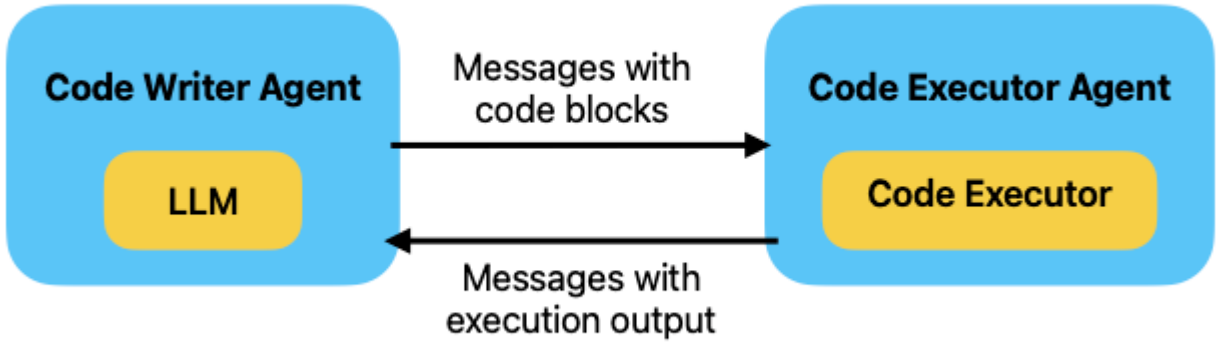
An agent can be powered by models (such as a large language model like GPT-4), code executors (such as an IPython kernel), human, or a combination of these and other pluggable and customizable components.

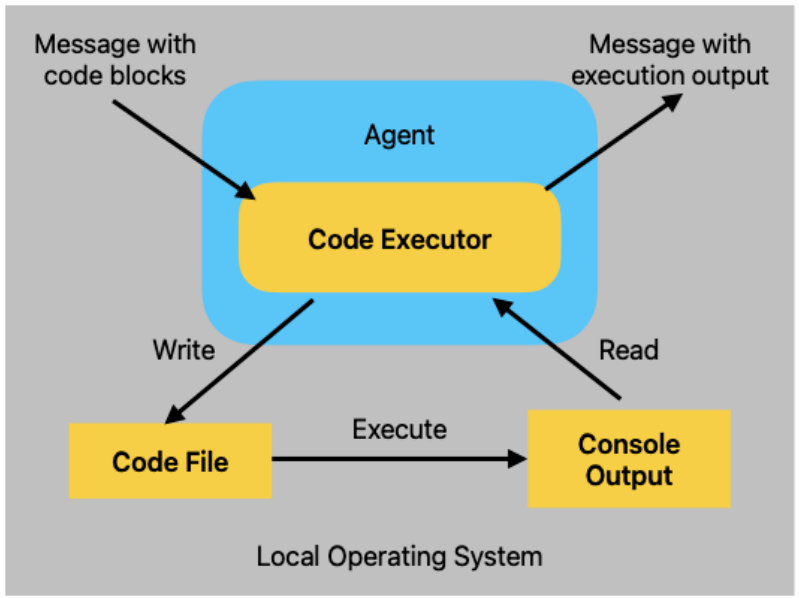
Human feedback

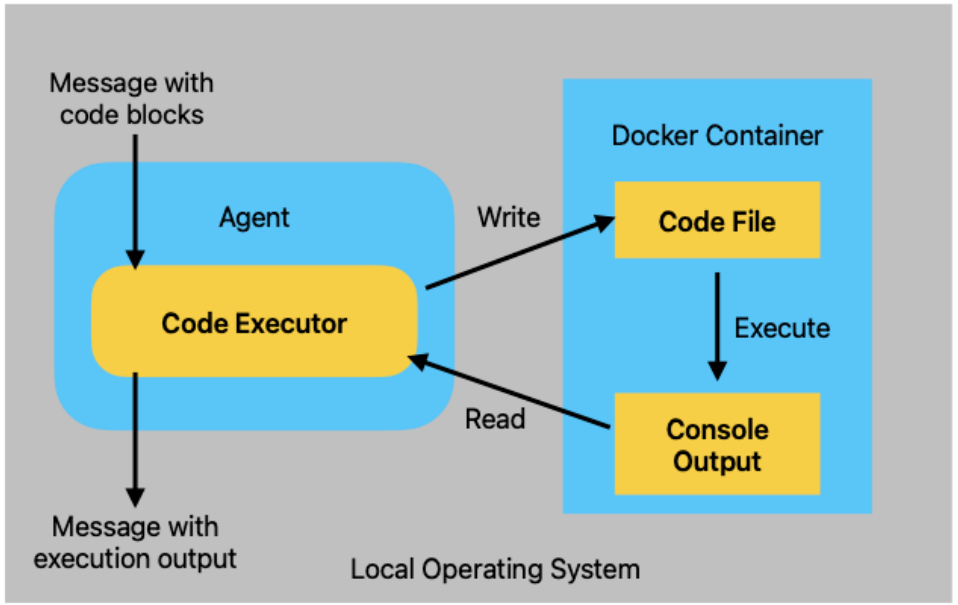


- NEVER: human input is never requested.
- TERMINATE (default): human input is only requested when a termination condition is met. Note that in this mode if the human chooses to intercept and reply, the conversation continues and the counter used by max_consecutive_auto_reply is reset.
- ALWAYS: human input is always requested and the human can choose to skip and trigger an auto-reply, intercept and provide feedback, or terminate the conversation. Note that in this mode termination based on max_consecutive_auto_reply is ignored.

Code Executor







UserProxyAgent vs. AssistantAgent

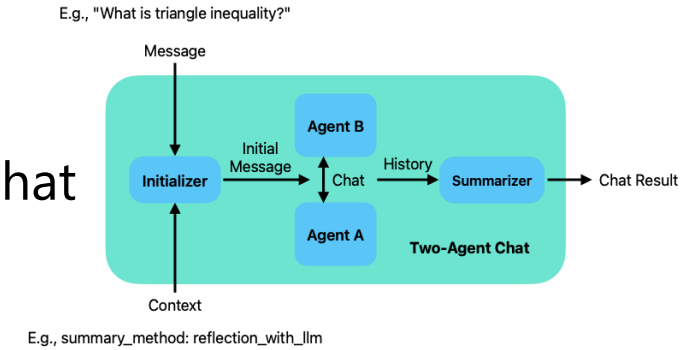
- ✓ UserProxyAgent as Human Input Proxy
  - Acts as a proxy for humans, soliciting human input for agent replies
- ✓ Automatic Code Execution Capability
  - Triggers code execution when executable code blocks are detected
  - Code execution can be disabled with 'code_execution_config'
- ✓ LLM-Based Response Options
  - Disabled by default, can be enabled with 'llm_config' dictionary
  - Generates replies using LLM when code execution is not needed

- ✓ AssistantAgent Functionality
  - Acts as an AI assistant using LLMs
  - Does not require human input or code execution
- ✓ Python Code Generation
  - Writes Python code for user execution based on received messages
- ✓ Execution Results Handling
  - Receives execution results and suggests corrections or bug fixes
- ✓ Customizable Behavior
  - Behavior alteration via system messages
- ✓ LLM Inference Configuration
  - Configurable via [llm_config]

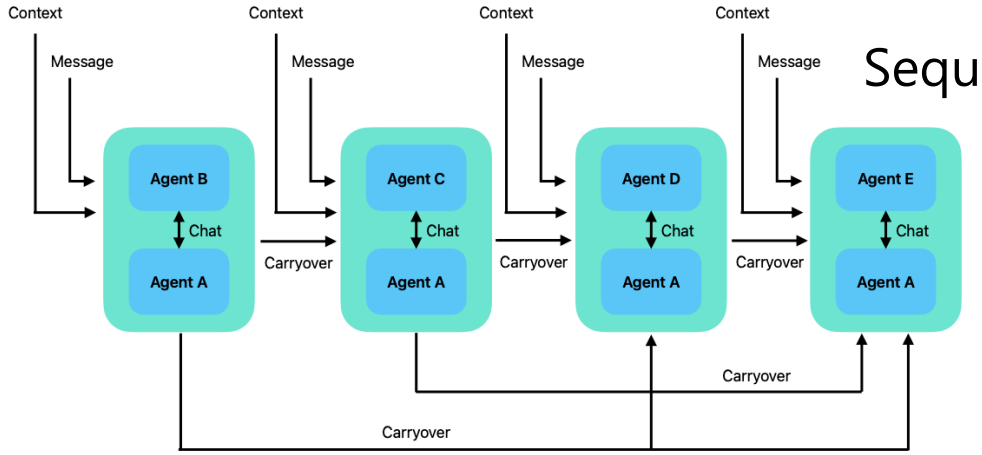
Conversation Patterns – Building Blocks

Multi-Agent workflows are built using conversation patterns.

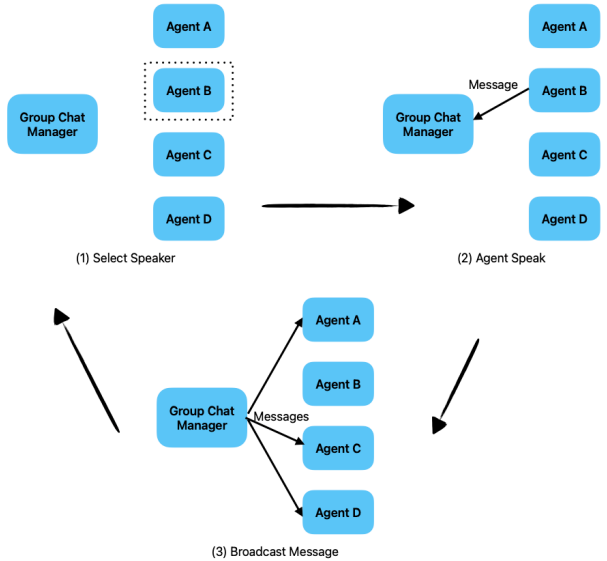
Two-Agent Chat



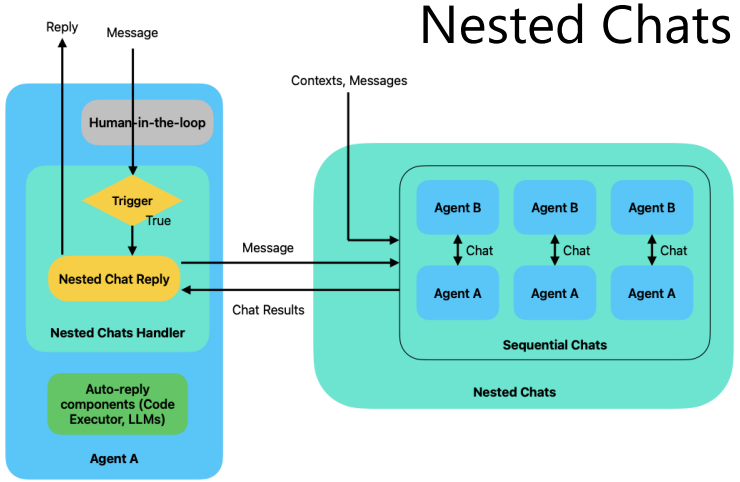
Sequential Chats



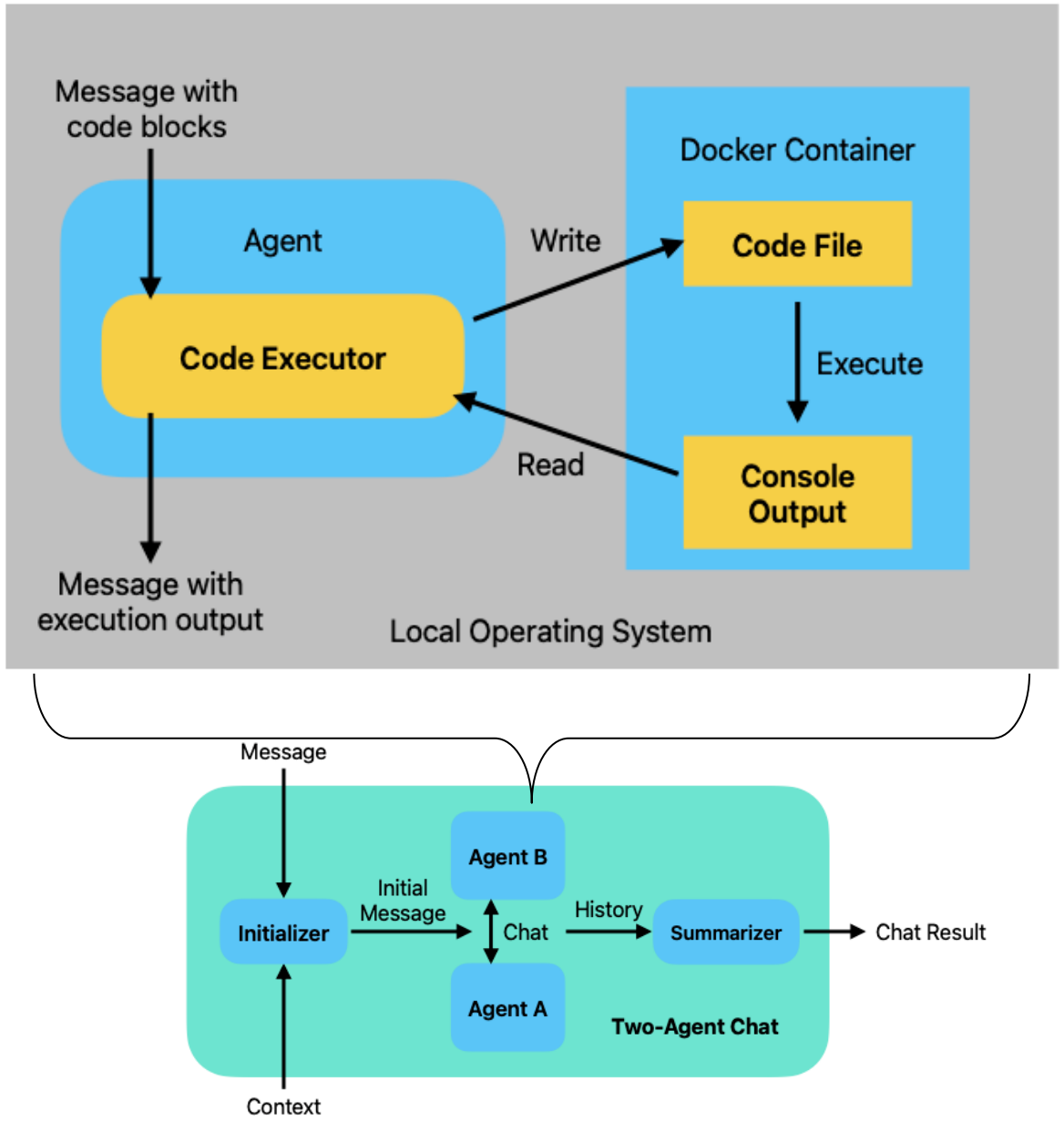
Group Chat



Nested Chats



two-agent-chat.py



Two-Agent Chat

Two agents converse back-and-forth:

1. assistant suggests code
2. executor runs code

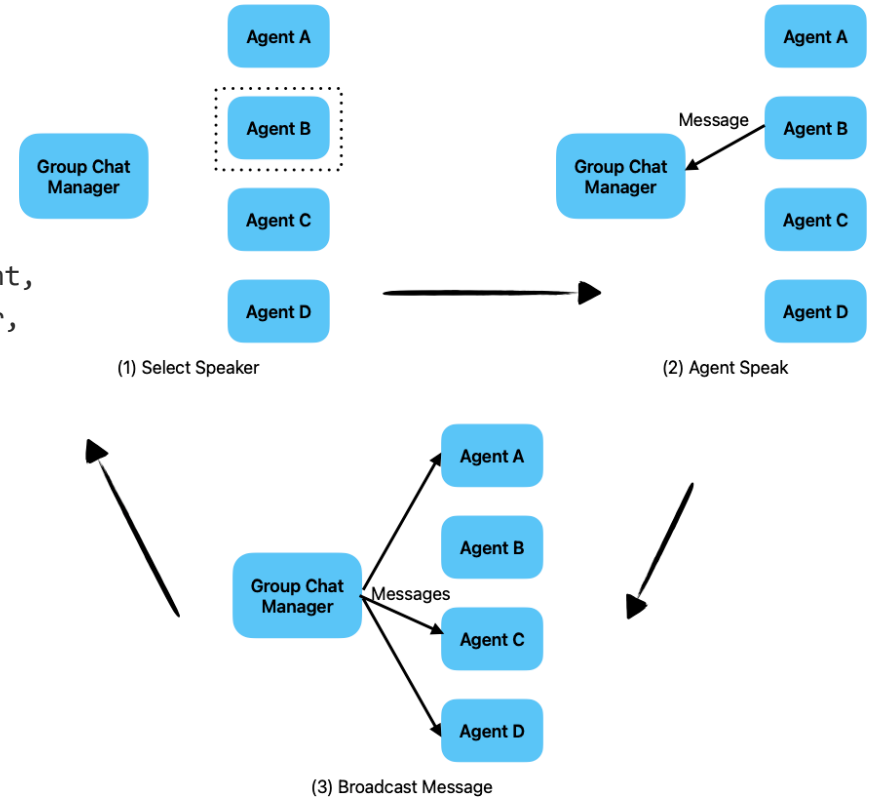
Safety:

- Code execution in a Docker sandbox
- Human-in-the-loop for executor

Lab 1 – Agentic Chatbot

group-chat.py

```
programming_assistant = ...
programming_executor = ...
research_assistant = ...
research_tool = ...
planner = ConversableAgent(
    system_message="You are a planner for complex tasks. "
    "You come up with a plan and assign subtasks to different agents. "
    "You also check for the completion of the tasks and provide feedback to
the agents."
    ...
)
admin = ...

group_chat = GroupChat(
    agents=[
        programming_assistant,
        programming_executor,
        research_assistant,
        research_tool,
        planner,
        admin,
    ],
)
```



Group Chat

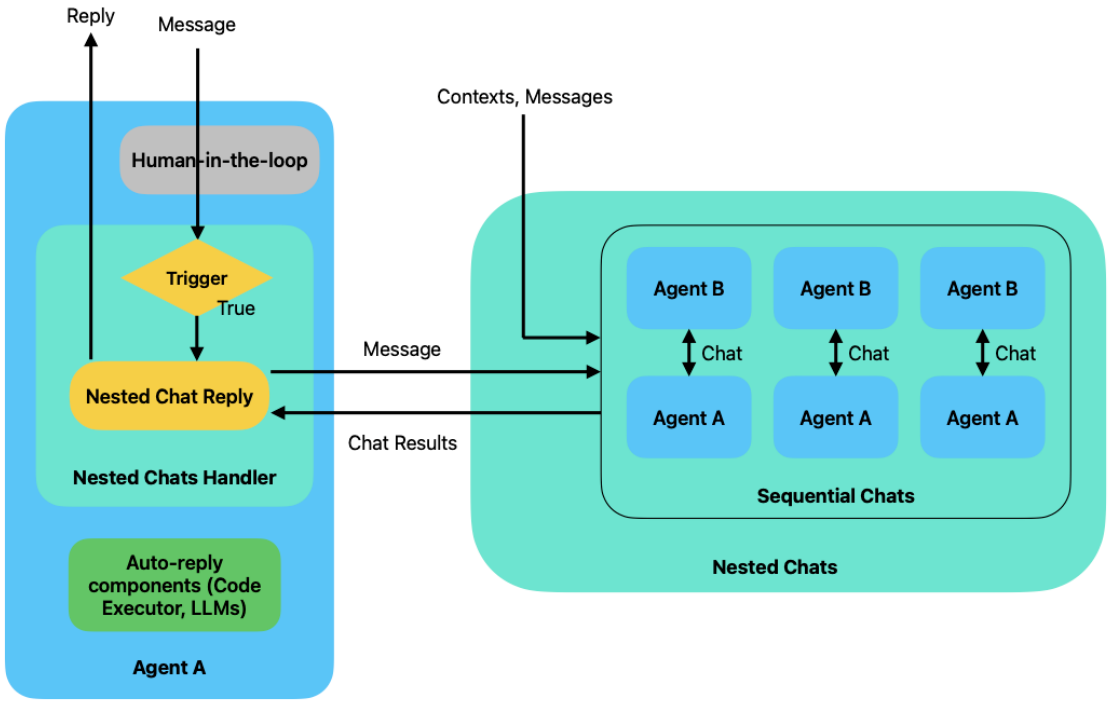
Participant agents takes turn to speak

- Speaker selected by a group chat manager
- A planner agent to plan and guide other agents
- An admin agent for collecting human feedback.

Lab 2 - Notebook

nested-chat.py

```
programming_team = ConversableAgent(name="Programming Team", …)
programming_team.register_nested_chats(
    [{"sender": programming_executor,
      "recipient": programming_assistant,
      "summary_method": "reflection_with_llm",
      "summary_args": {"summary_prompt": "Provide a detailed summary of the
code execution."},}],
    trigger=lambda sender: sender not in [programming_assistant,
programming_executor],
)
reply = programming_team.generate_reply(
    [{"content": "What is the binomial distribution with 10 trials and a
probability of 0.5?", "role": "user"}]
)
```



Nested Chat

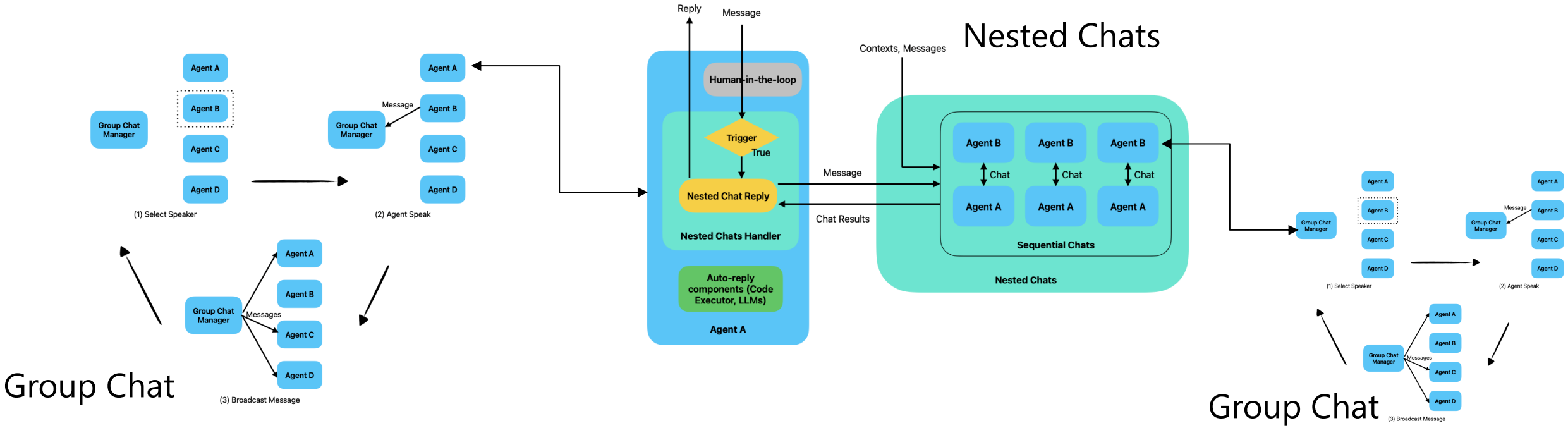
An agent is packaged with an inner conversation

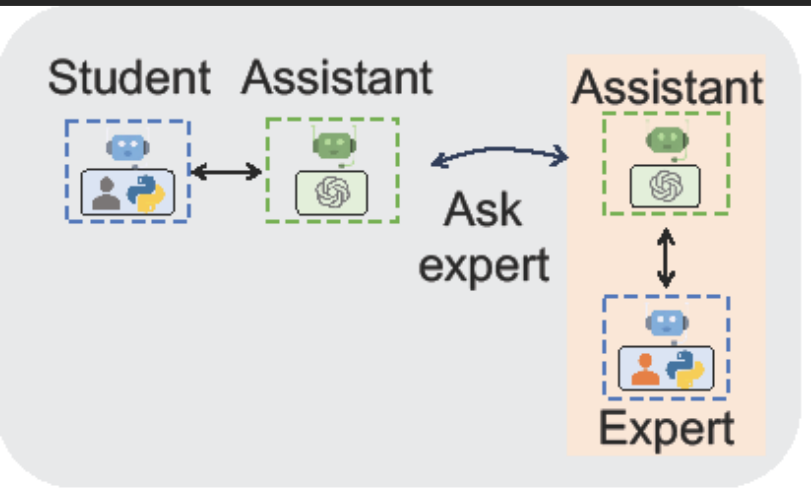
- Triggered by an external message
- Inner conversation can be a sequence of chats
- Summary of inner conversation is used as response

Lab 3 - Notebook

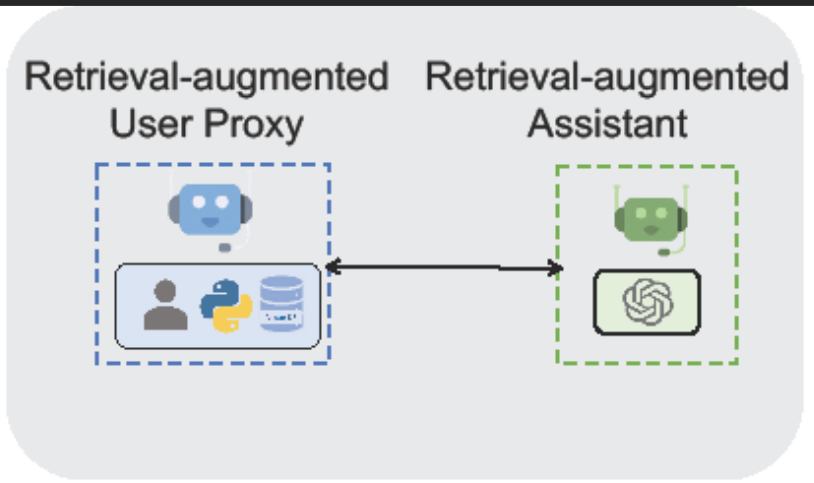
Conversation Patterns are Composable

E.g., each agent in the group chat can be a nested sequential chats.

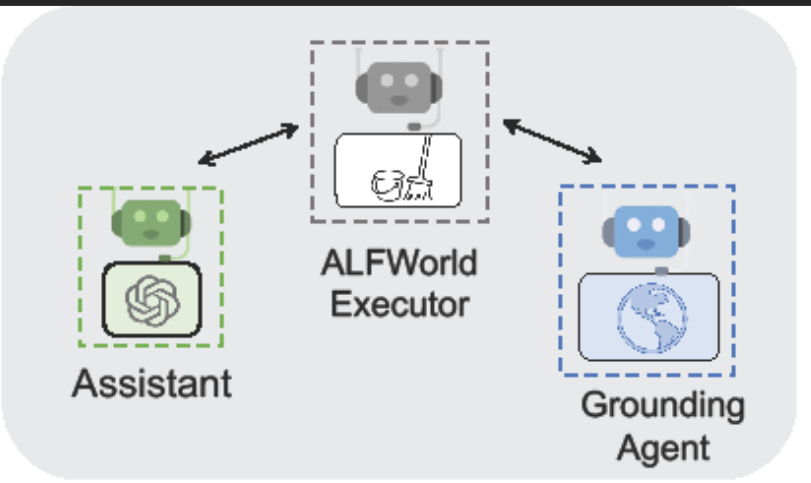




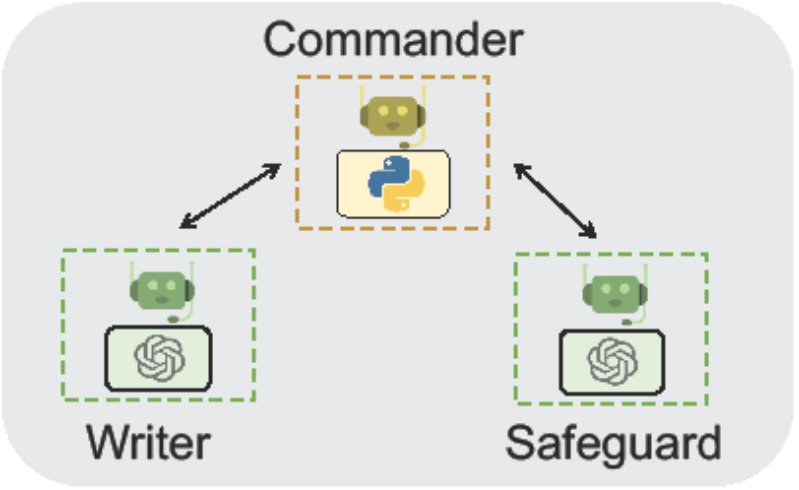
A1. Math Problem Solving



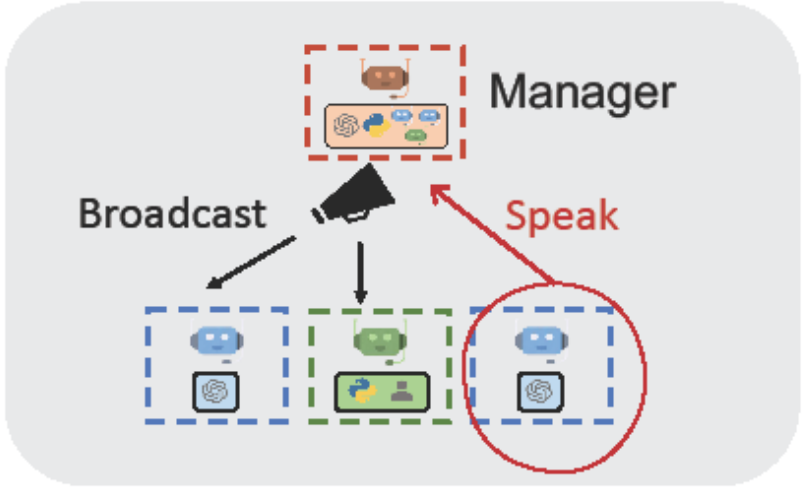
A2. Retrieval-augmented Q&A



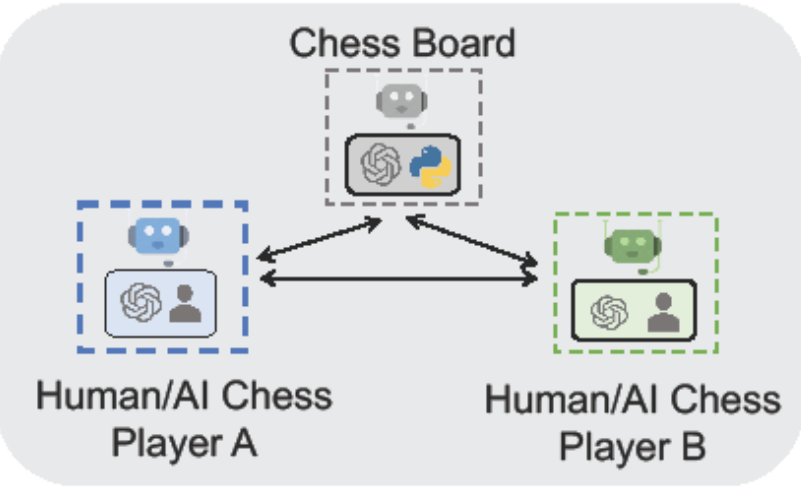
A3. Decision Making in Household Tasks



A4. Supply-Chain Optimization



A5. Dynamic Task Solving with Group Chat



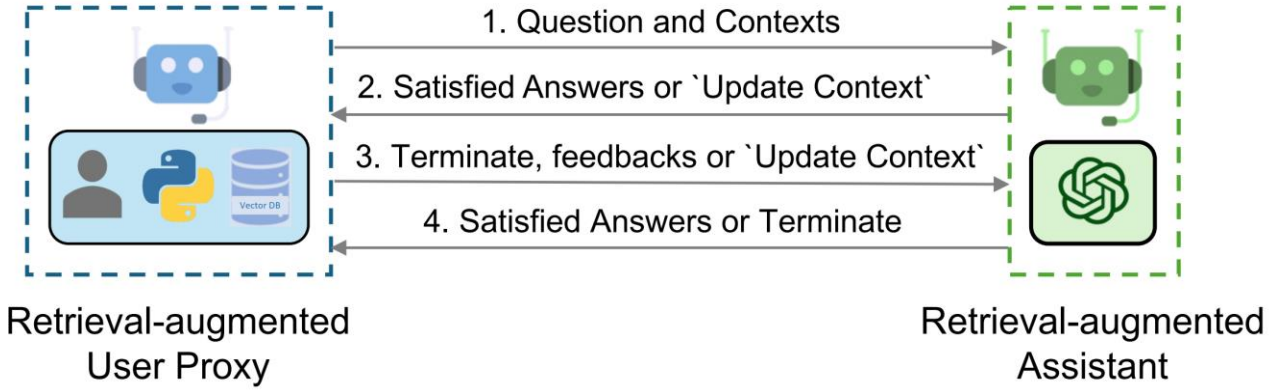
A6. Conversational Chess

Lab 4 - Notebook

Advanced Autogen

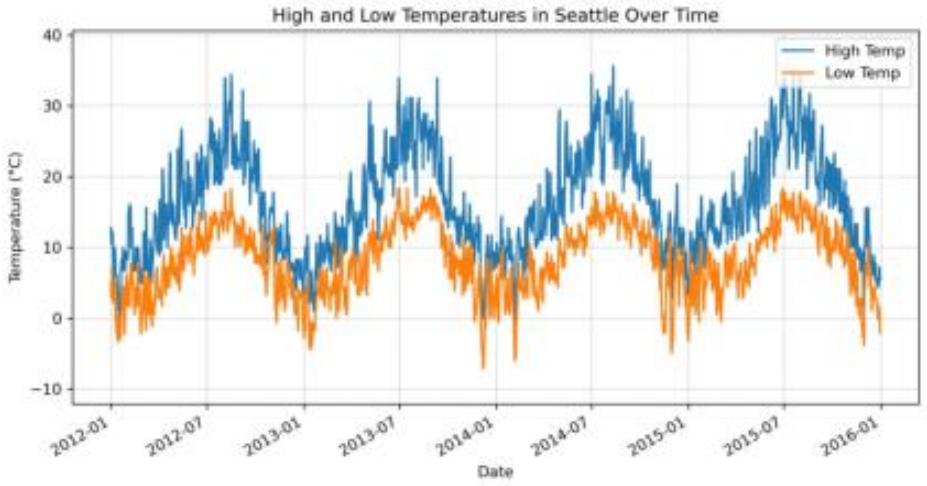
RAG Agent

```
assistant = RetrieveAssistantAgent(
    name="assistant",
    system_message="You are a helpful
assistant.",
    llm_config=llm_config,)

ragproxyagent = RetrieveUserProxyAgent(
    name="ragproxyagent",
    retrieve_config={
        "task": "qa",
        "docs_path":
https://raw.githubusercontent.com/microsoft/autogen/main/README.md
    }
)
```



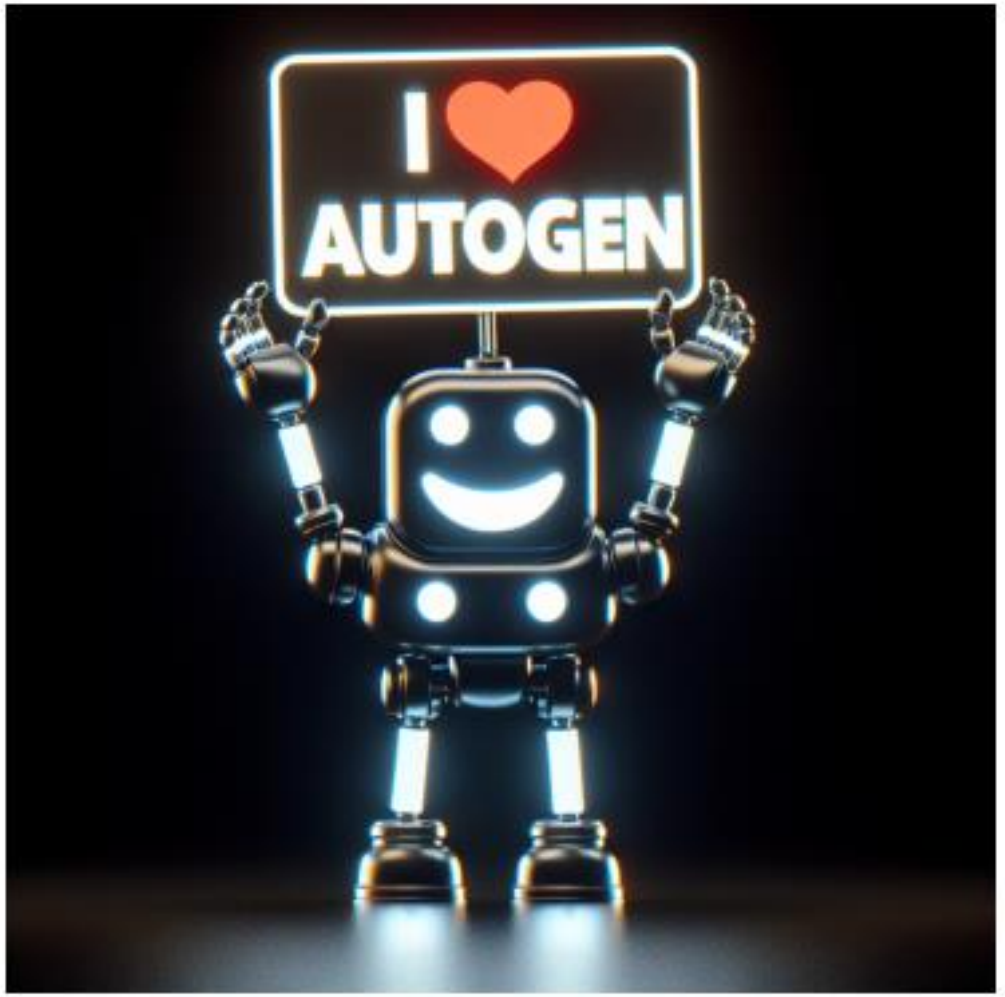
Multimodal Conversable Agent

```
prompt = """
Give me some suggestions to the image visualization in <img x.jpg>. Thanks!
"""

agent = MultimodalConversableAgent()

user = UserProxyAgent()

user.initiate_chat(agent, message=prompt)
```



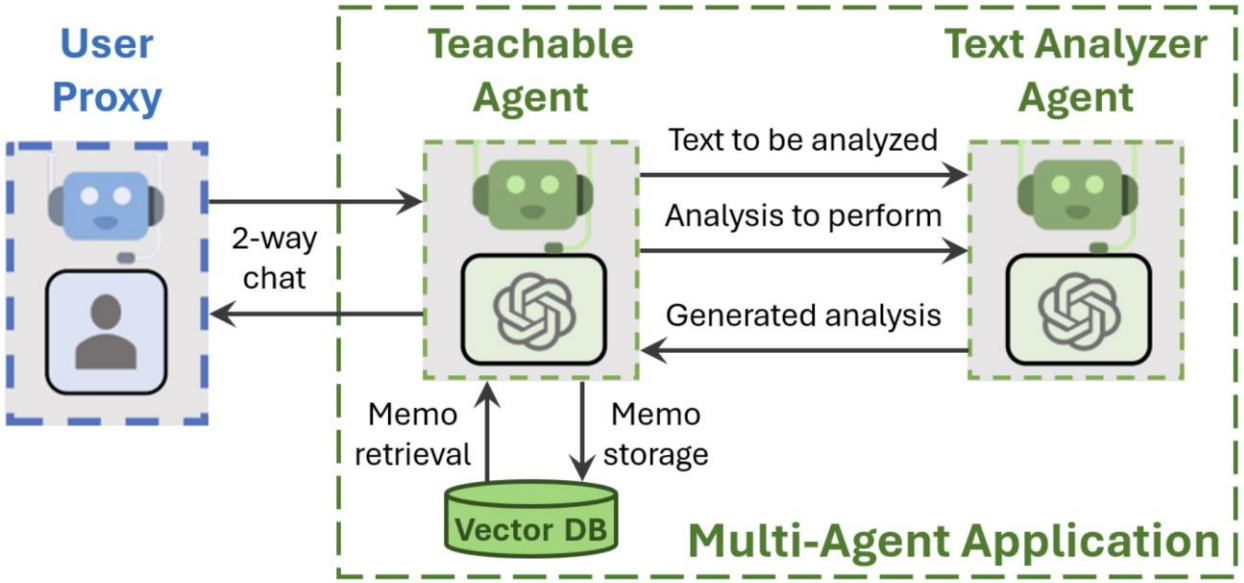
DALLE Agent

```
dalle = DALLEAgent(name="Dalle",
llm_config={"config_list": config_list_dalle})

user_proxy = UserProxyAgent(
    name="User_proxy",
    system_message="A human admin.",
    human_input_mode="NEVER",
    max_consecutive_auto_reply=0)

user_proxy.initiate_chat(
    dalle,
    message="Create an image with black
background, a happy robot is showing a sign with
"I Love AutoGen"."
)
```



Teachable Agent

Teachability enables persistence of learning across chat boundaries using a vector DB that allows storage and retrieval of "memos" that contain facts, preferences and skills.




```
# Start by instantiating any agent that inherits from ConversableAgent, which we use directly here for simplicity.

teachable_agent = ConversableAgent (
name="teachable_agent", # The name can be anything.
llm_config=llm_config
)

# Instantiate a Teachability object. Parameters are all optional.
teachability = Teachability(
# Use True to force-reset the memo DB, and False to use an existing DB.
reset_db = False,
path_to_db_dir = "./tmp/interactive/teachability_db“ )

# Now add teachability to the agent.
teachability.add_to_agent(teachable_agent)
```

LLM Caching

```
# Use Redis as cache
with Cache.redis(redis_url="redis://localhost:6379/0") as cache:
)

# Use DiskCache as cache
with Cache.disk() as cache:

# Use Azure Cosmos DB as cache
with Cache.cosmos_db(connection_string="your_connection_string", database_id="your_database_id", container_id="your_container_id") as cache:
```

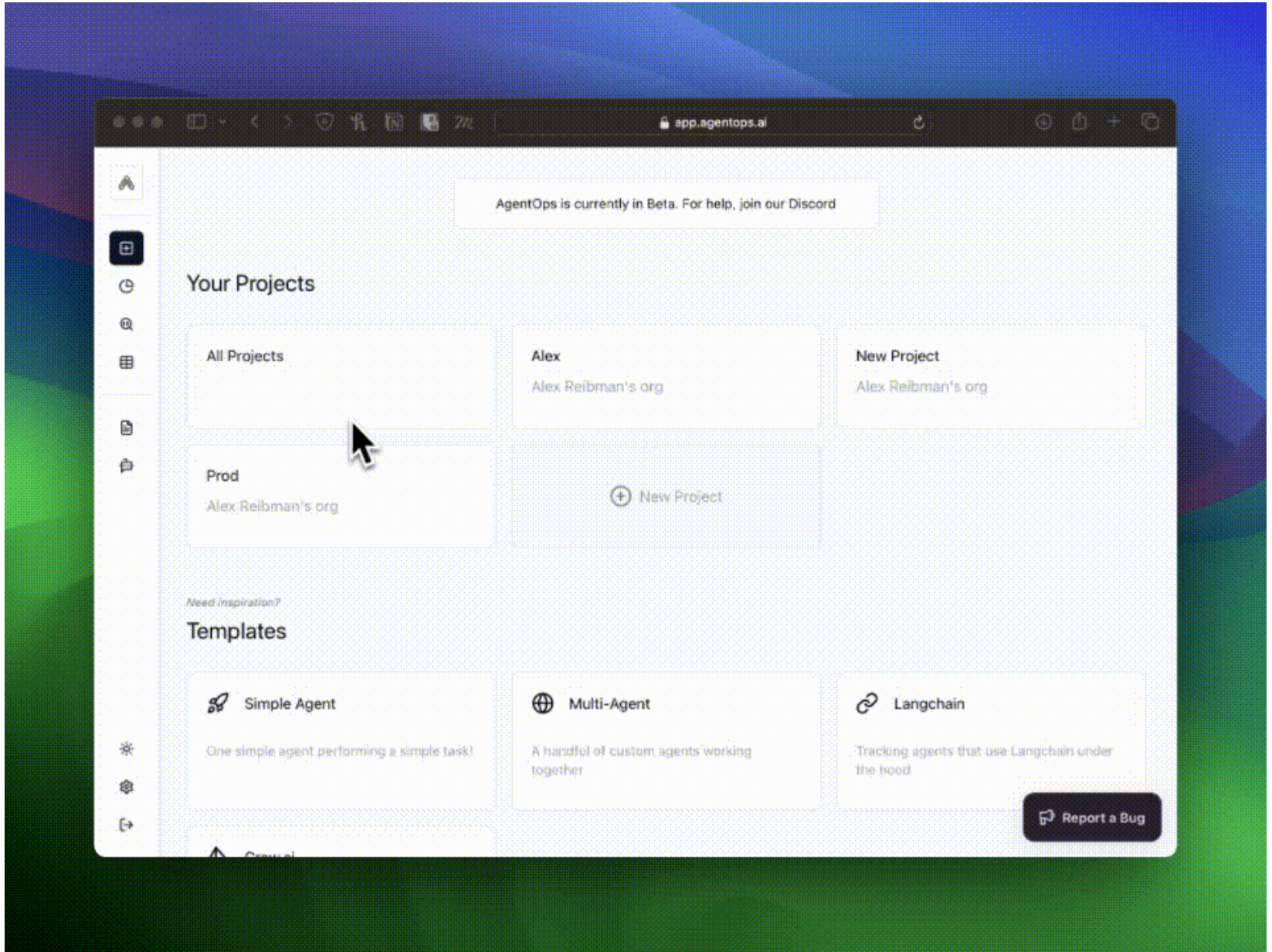
```
with Cache.disk(cache_seed=1) as cache:
user.initiate_chat(assistant, message=coding_task, cache=cache)
```

Cache on redis, disk, cosmos and etc

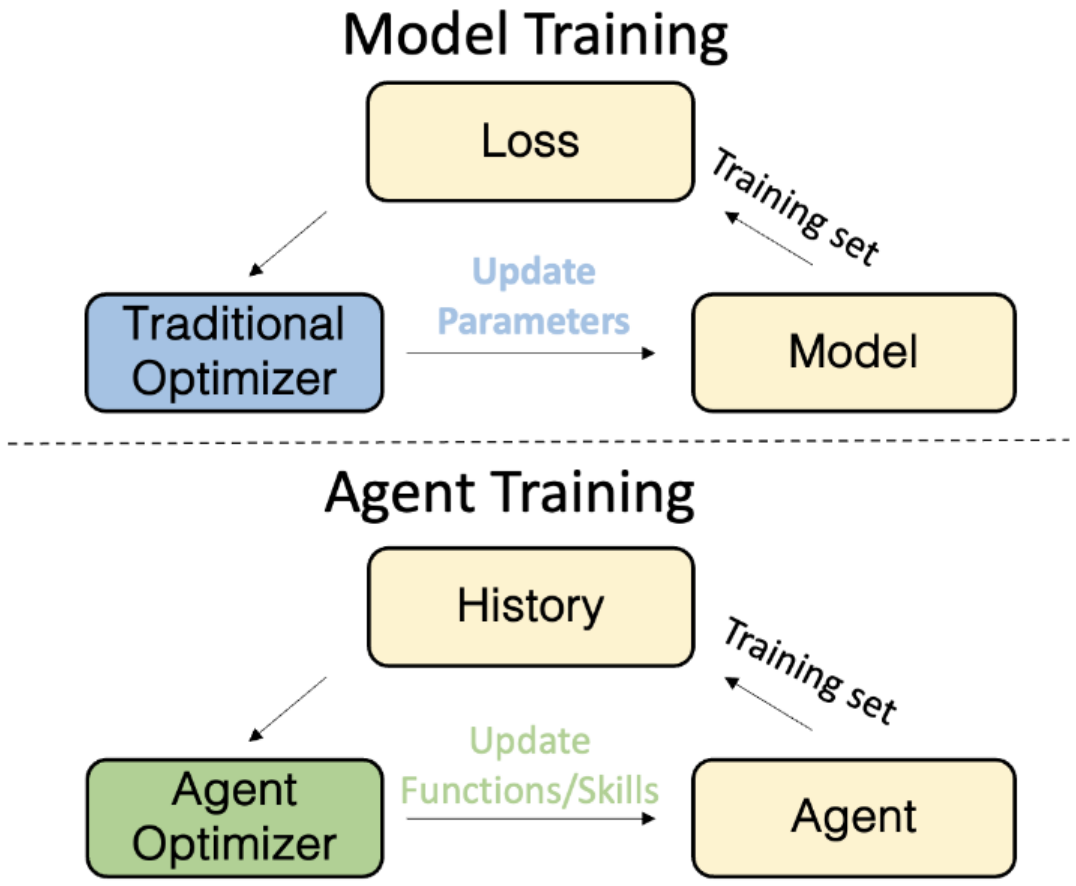
`cache_seed`

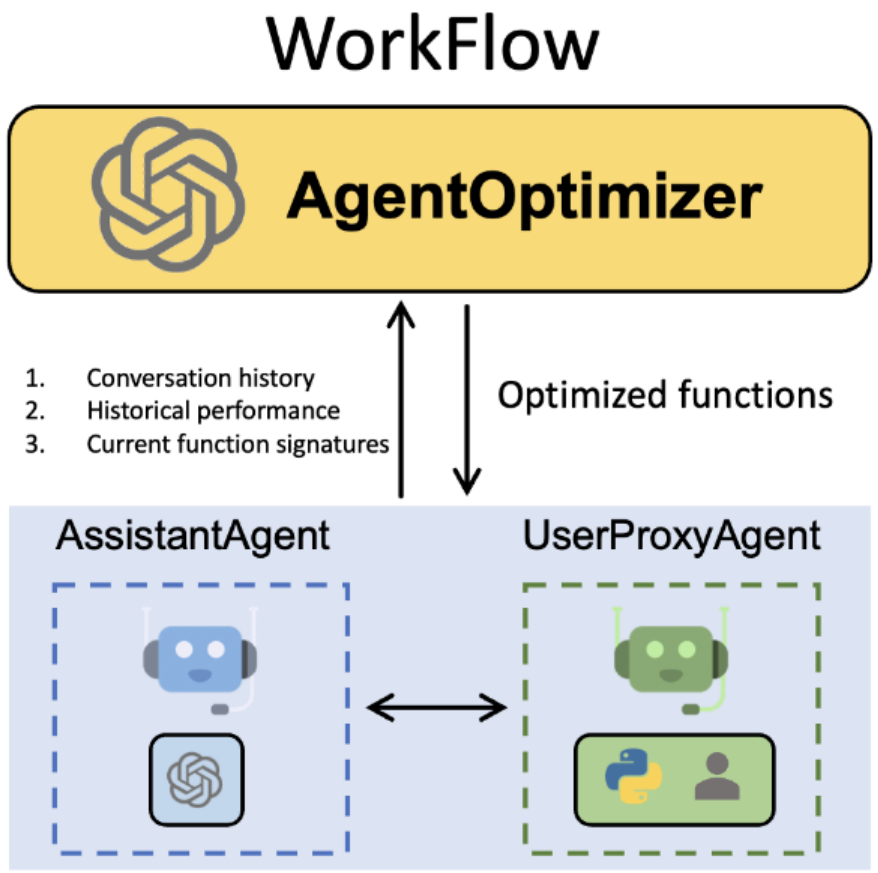
Different from OpenAI Seed

AgentOps

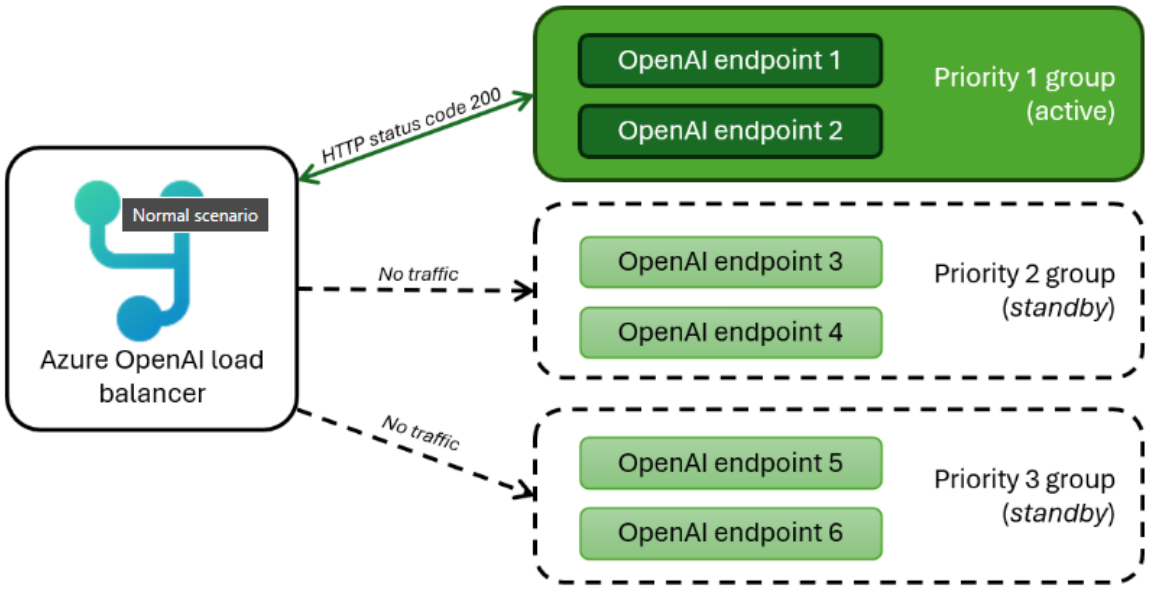

app.agentops.ai
AgentOps is currently in Beta. For help, join our Discord
Your Projects
All Projects
Alex
Alex Reibman's org
New Project
Alex Reibman's org
Prod
Alex Reibman's org
New Project
Need inspiration?
Templates
Simple Agent
One simple agent performing a simple task
Multi-Agent
A handful of custom agents working together
Langchain
Tracking agents that use Langchain under the hood
Report a Bug

Agentic Optimizer





LLM Configuration (comparing AOAI smart LB)




```
llm_config = {
  "config_list": [
    {
      "model": "my-gpt-4-deployment",
      "api_key": os.environ.get("AZURE_OPENAI_API_KEY"),
      "api_type": "azure",
      "base_url": os.environ.get("AZURE_OPENAI_API_BASE"),
      "api_version": "2024-02-01",
    },
    {
      "model": "llama-7B",
      "base_url": "http://127.0.0.1:8080",
      "api_type": "openai",
    },
  ],
  "temperature": 0.9,
  "timeout": 300,
}
```

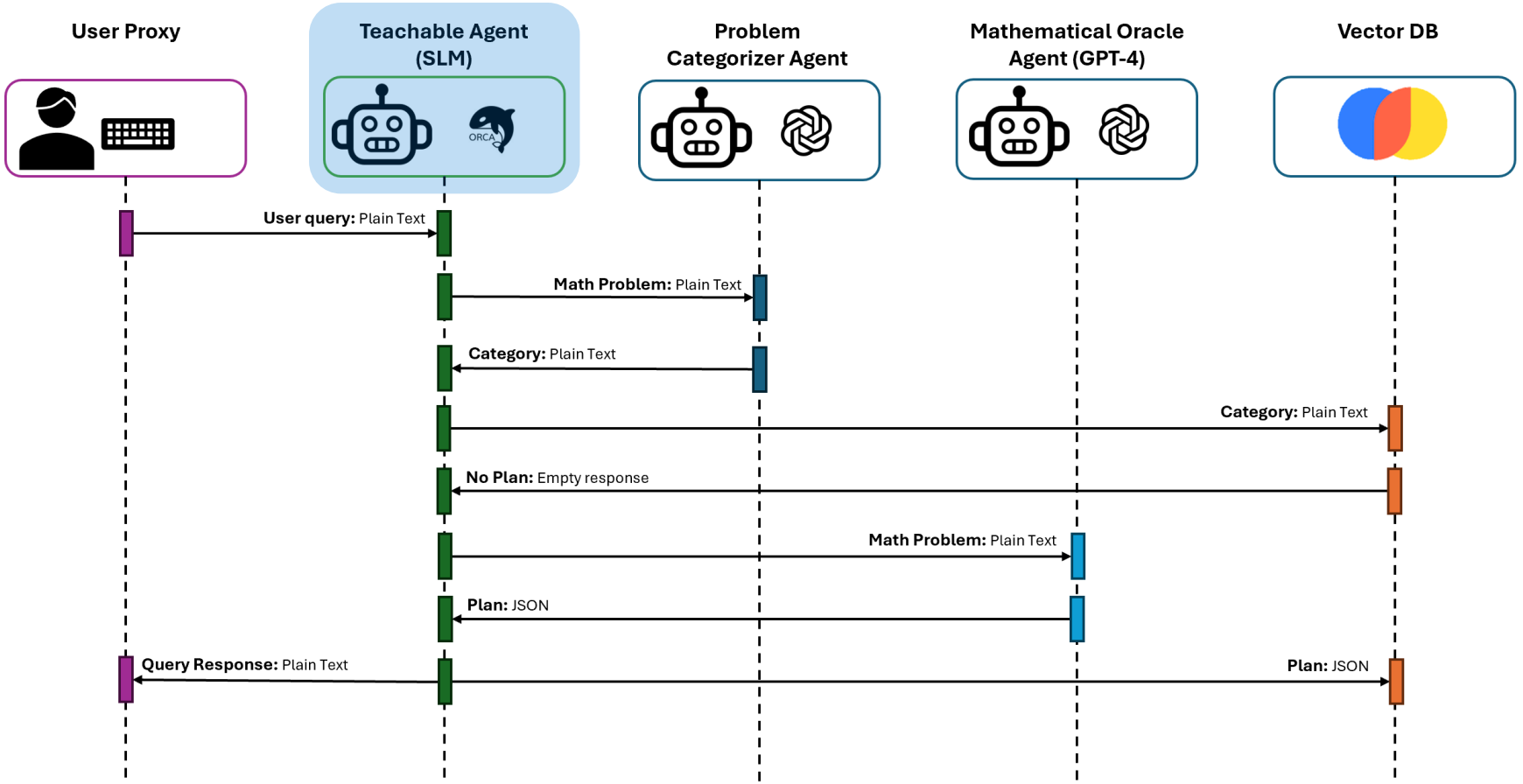
Efficient AI

- The use of SLM (e.g., Phi-2, Orca-2) to perform the duty of Teachable Agents can greatly reduce the reliance on expensive LLM calls.

Key research area of
M365 Applied Research

For more contact:

- Daniel Madrigal Diaz
  danielmad@microsoft.com
- Molly Xia
  mollyxia@microsoft.com
- Victor Rühle
  virueh@microsoft.com



Final Lab

Requirement

임솔은 KB금융그룹 마케팅팀의 신입사원입니다. 유튜브 콘텐츠 제작에 재능이 있어 그룹 홍보용 콘텐츠를 여러 건 제작해왔습니다. 어느 날, 마케팅 본부장 류선재 상무가 임솔 사원에게 말했습니다. “유튜브 콘텐츠 좋던데, 몇 개는 디지털 사보나 기업 블로그에도 실렸으면 해. 한 번 고민해봐.”

임솔은 블로그 작성 경험이 없어 고민하던 중, MS의 Agent AI 워크샵에 참석하게 되었습니다. 이 워크샵을 통해 무언가 쉽게 자동화할 수 있을 것 같다는 생각이 들었습니다. 우리 모두 솔이 사원을 도와줍시다.

<<이미 제작된 Youtube URL을 넣으면 Caption을 추출하여 블로그용 콘텐츠를 만들어주는 챗봇을 개발하자!>>

Wrap up and Q&A

Thank you