TaskGen -> AgentJo

TaskGen: A Task-Based, Memory-Infused Agentic Framework building on StrictJSON
https://github.com/simbianai/taskgen

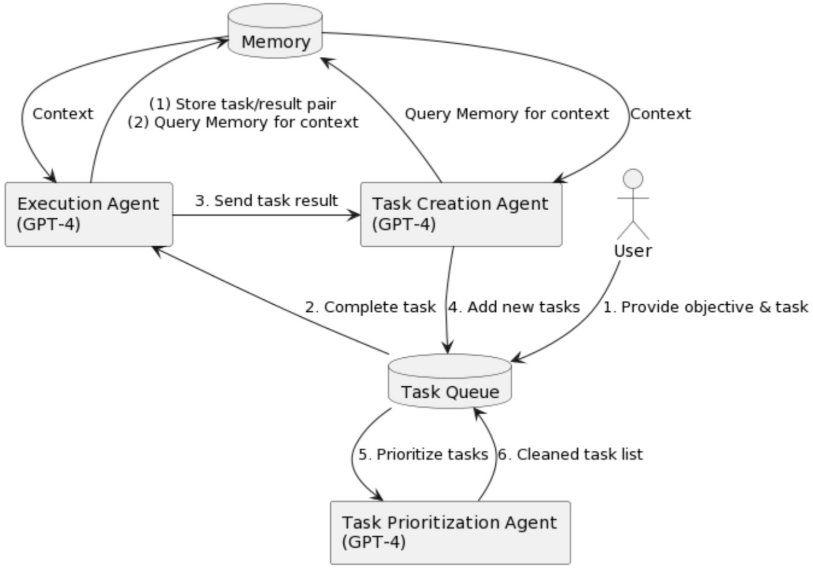
AgentJo: **Human-Friendly**, Fast Learning and Adaptable Agent Communities building on TaskGen
https://github.com/tanchongmin/agentjo

Current System & Future Plans

John Tan Chong Min

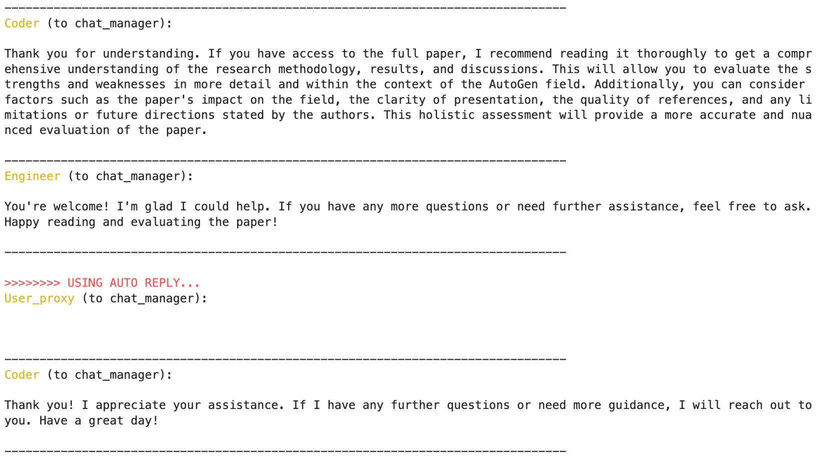
Problems with existing Agentic Framework

- Never-ending loop of tasks



Yohei Nakajima's BabyAGI
Related - AutoGPT

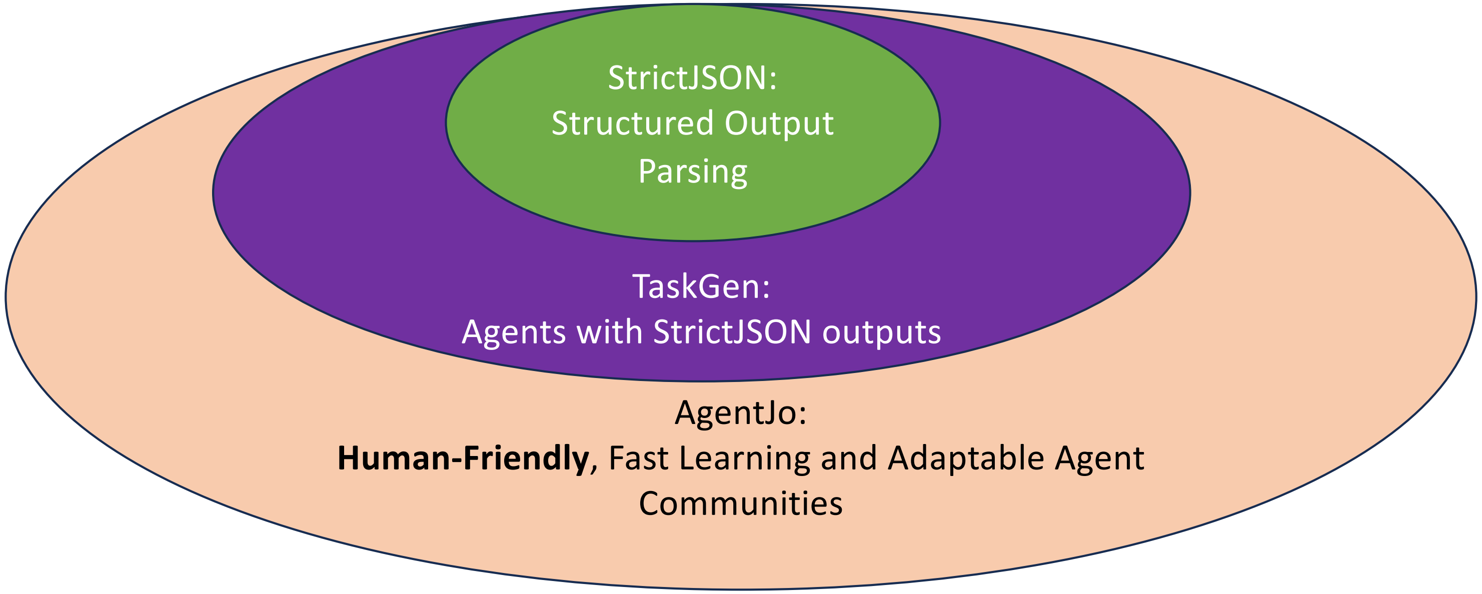
- Too verbose
- Agents respond to tasks they are not meant to



```
--------------------------------------------------------------------------------
Coder (to chat_manager):

Thank you for understanding. If you have access to the full paper, I recommend reading it thoroughly to get a compr
ehensive understanding of the research methodology, results, and discussions. This will allow you to evaluate the s
trengths and weaknesses in more detail and within the context of the AutoGen field. Additionally, you can consider
factors such as the paper's impact on the field, the clarity of presentation, the quality of references, and any li
mitations or future directions stated by the authors. This holistic assessment will provide a more accurate and nua
nced evaluation of the paper.

--------------------------------------------------------------------------------
Engineer (to chat_manager):

You're welcome! I'm glad I could help. If you have any more questions or need further assistance, feel free to ask.
Happy reading and evaluating the paper!

--------------------------------------------------------------------------------

>>>>>>>> USING AUTO REPLY...
User_proxy (to chat_manager):



--------------------------------------------------------------------------------
Coder (to chat_manager):

Thank you! I appreciate your assistance. If I have any further questions or need more guidance, I will reach out to
you. Have a great day!

--------------------------------------------------------------------------------
```

AutoGen
Related – crew.ai

Overview: PhD work (TaskGen) to Future Plans (AgentJo)



StrictJSON:
Structured Output
Parsing

TaskGen:
Agents with StrictJSON outputs

AgentJo:
Human-Friendly, Fast Learning and Adaptable Agent Communities

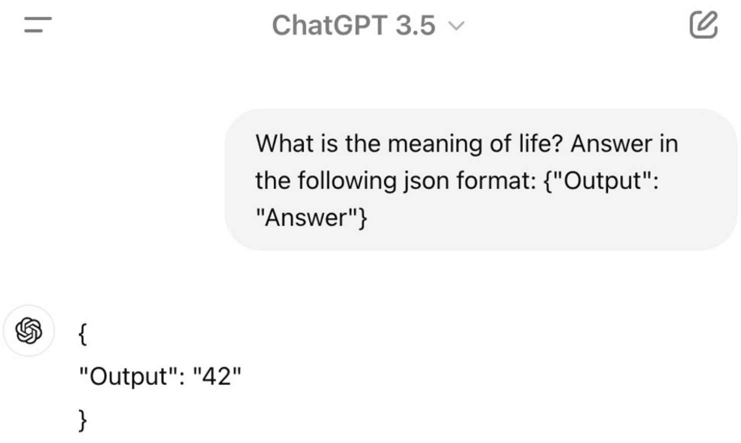
StrictJSON

A JSON parser for preserving all keys in JSON with type checking and more

https://github.com/tanchongmin/strictjson

Why StrictJSON?

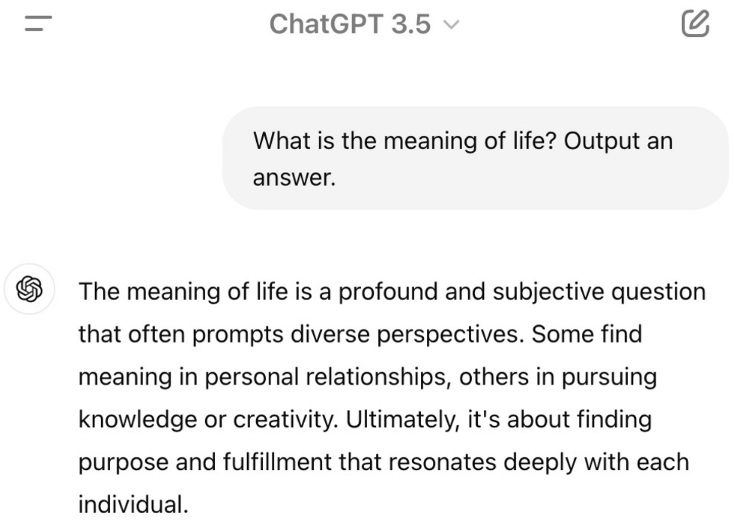
- JSON is a much less verbose method of output for LLM



What is the meaning of life? Answer in the following json format: {"Output": "Answer"}

```
{
"Output": "42"
}
```

JSON



What is the meaning of life? Output an answer.

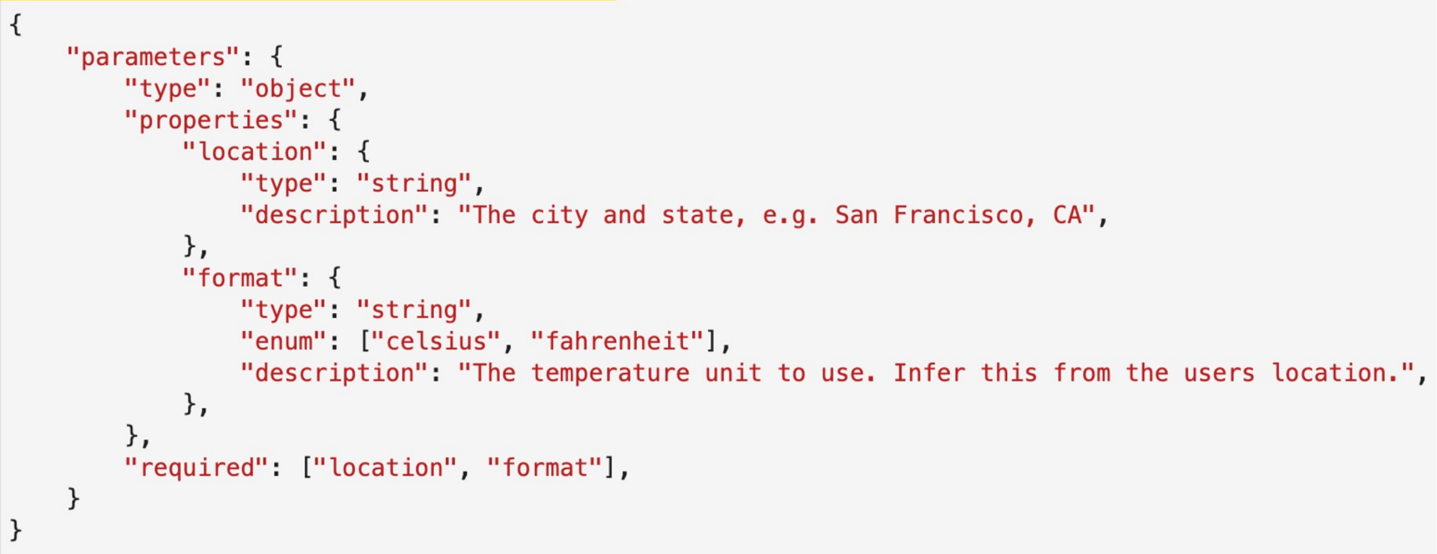
The meaning of life is a profound and subjective question that often prompts diverse perspectives. Some find meaning in personal relationships, others in pursuing knowledge or creativity. Ultimately, it's about finding purpose and fulfillment that resonates deeply with each individual.

Free Text

Why StrictJSON?

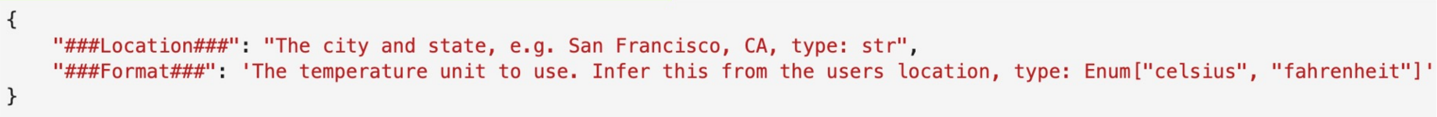
- Much lower token usage

JSON Schema for Parameters – 110 tokens



```
{
    "parameters": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA",
            },
            "format": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "description": "The temperature unit to use. Infer this from the users location.",
            },
        },
        "required": ["location", "format"],
    }
}
```

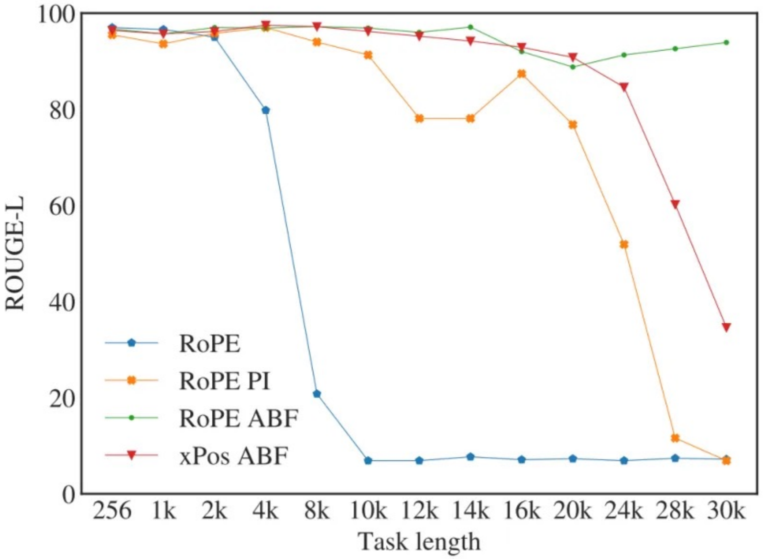
StrictJSON Schema for Parameters – 58 tokens



```
{
    "###Location###": "The city and state, e.g. San Francisco, CA, type: str",
    "###Format###": 'The temperature unit to use. Infer this from the users location, type: Enum["celsius", "fahrenheit"]'
}
```

Tokens impact not just cost, but performance

- Performance sharply degrades after 2-3k tokens
  - For Rotary Positional Embeddings (RoPE) in Llama 2

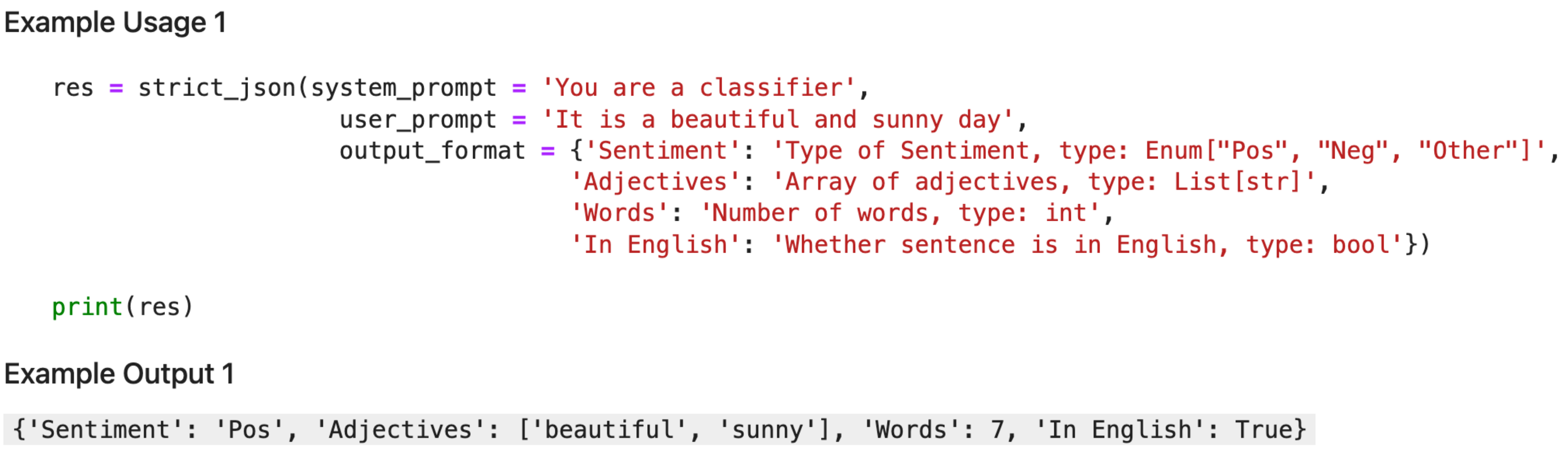


(b) Performance on FIRST-SENTENCE-RETRIEVAL task.

Effective Long-Context Scaling of Foundation Models. 2023. Xiong et. al.

StrictJSON Ensures Keys and Types of Output

- Currently supported types:
  - \`int\`, \`float\`, \`str\`, \`dict\`, \`list\`, \`array\`, \`Dict[]\`, \`List[]\`, \`Array[]\`, \`Enum[]\`, \`bool\`



Example Usage 1

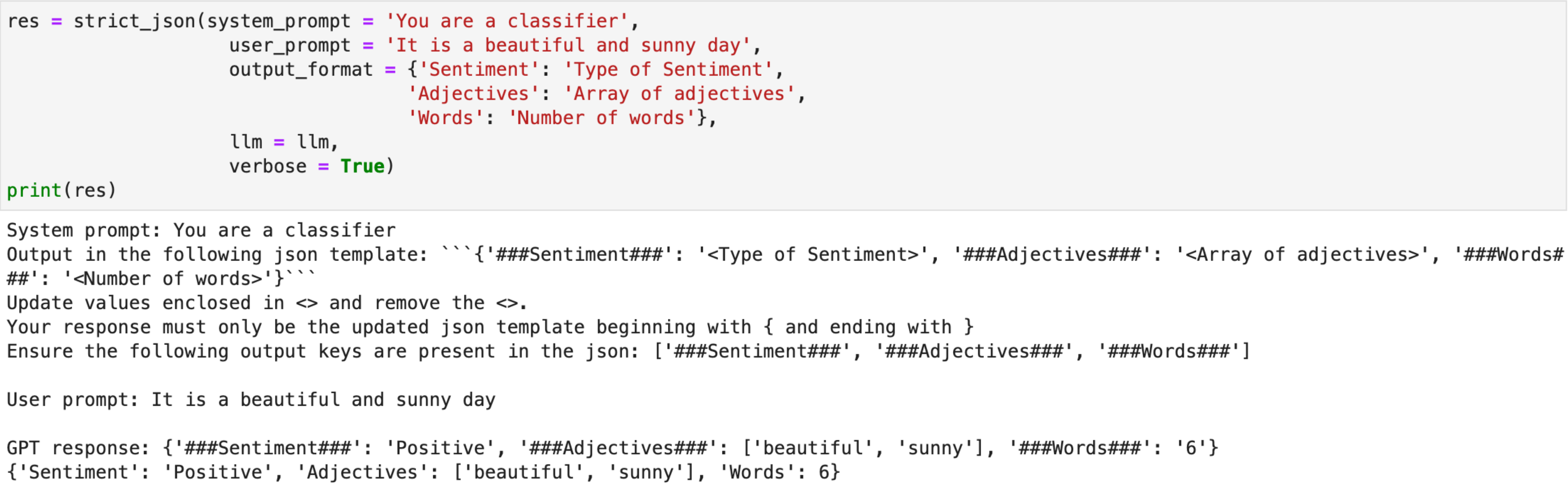
```
res = strict_json(system_prompt = 'You are a classifier',
                    user_prompt = 'It is a beautiful and sunny day',
                    output_format = {'Sentiment': 'Type of Sentiment, type: Enum["Pos", "Neg", "Other"]',
                                    'Adjectives': 'Array of adjectives, type: List[str]',
                                    'Words': 'Number of words, type: int',
                                    'In English': 'Whether sentence is in English, type: bool'})

print(res)
```

Example Output 1

```
{'Sentiment': 'Pos', 'Adjectives': ['beautiful', 'sunny'], 'Words': 7, 'In English': True}
```

How StrictJSON works

- Uses ### delimiters to enclose keys, then extract them via regex



````
res = strict_json(system_prompt = 'You are a classifier',
                    user_prompt = 'It is a beautiful and sunny day',
                    output_format = {'Sentiment': 'Type of Sentiment',
                                    'Adjectives': 'Array of adjectives',
                                    'Words': 'Number of words'},
                    llm = llm,
                    verbose = True)
print(res)
````

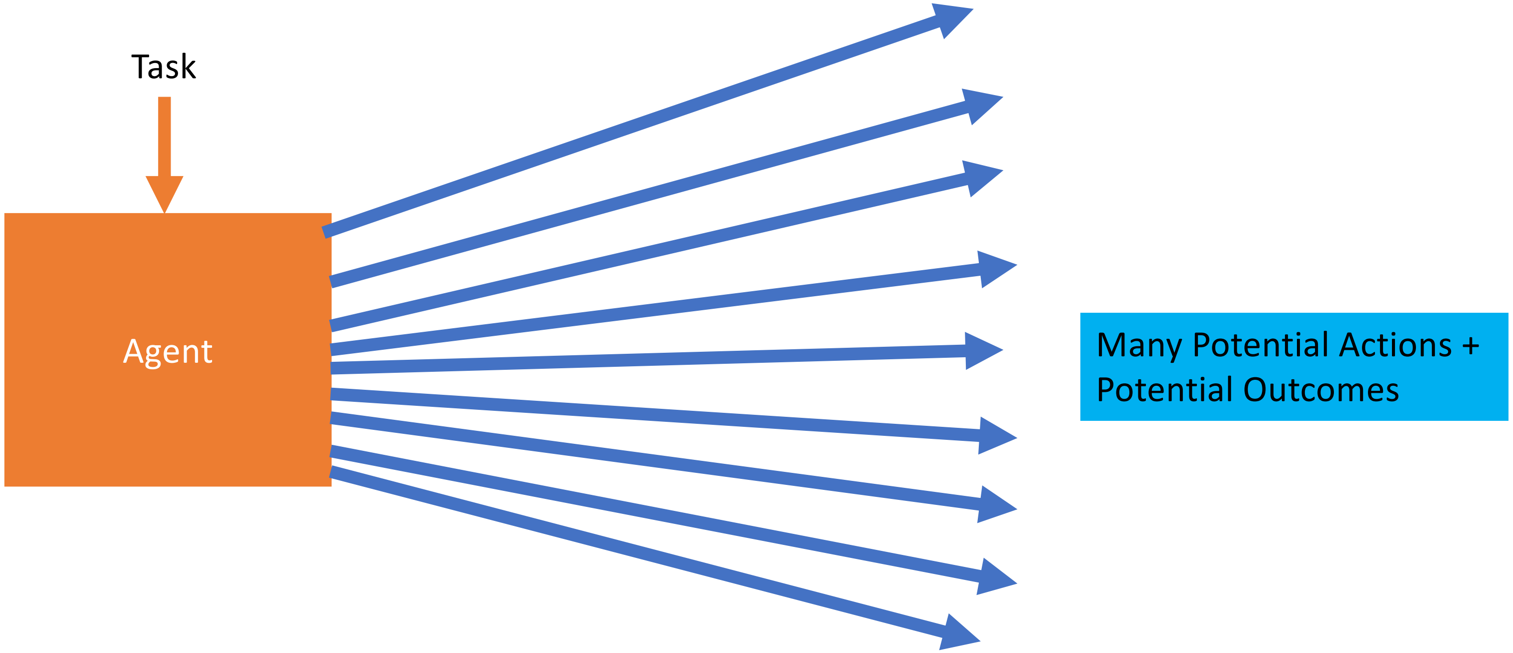
````
System prompt: You are a classifier
Output in the following json template: ```{'###Sentiment###': '<Type of Sentiment>', '###Adjectives###': '<Array of adjectives>', '###Words#
##': '<Number of words>'}```
Update values enclosed in <> and remove the <>.
Your response must only be the updated json template beginning with { and ending with }
Ensure the following output keys are present in the json: ['###Sentiment###', '###Adjectives###', '###Words###']

User prompt: It is a beautiful and sunny day

GPT response: {'###Sentiment###': 'Positive', '###Adjectives###': ['beautiful', 'sunny'], '###Words###': '6'}
{'Sentiment': 'Positive', 'Adjectives': ['beautiful', 'sunny'], 'Words': 6}
````

TaskGen

Building on StrictJSON for Agentic Task Solving!

https://github.com/simbianai/taskgen

Key Question: How does an agent know what to do?

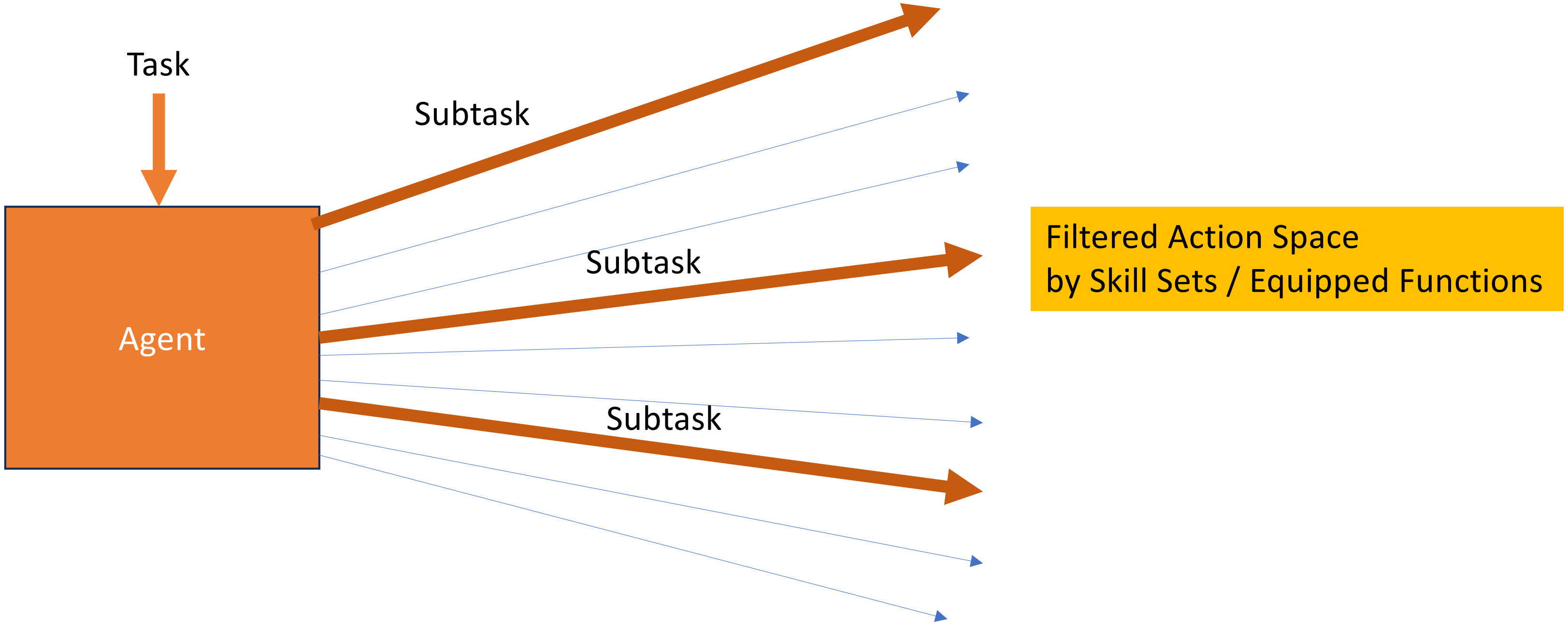


Task

Agent

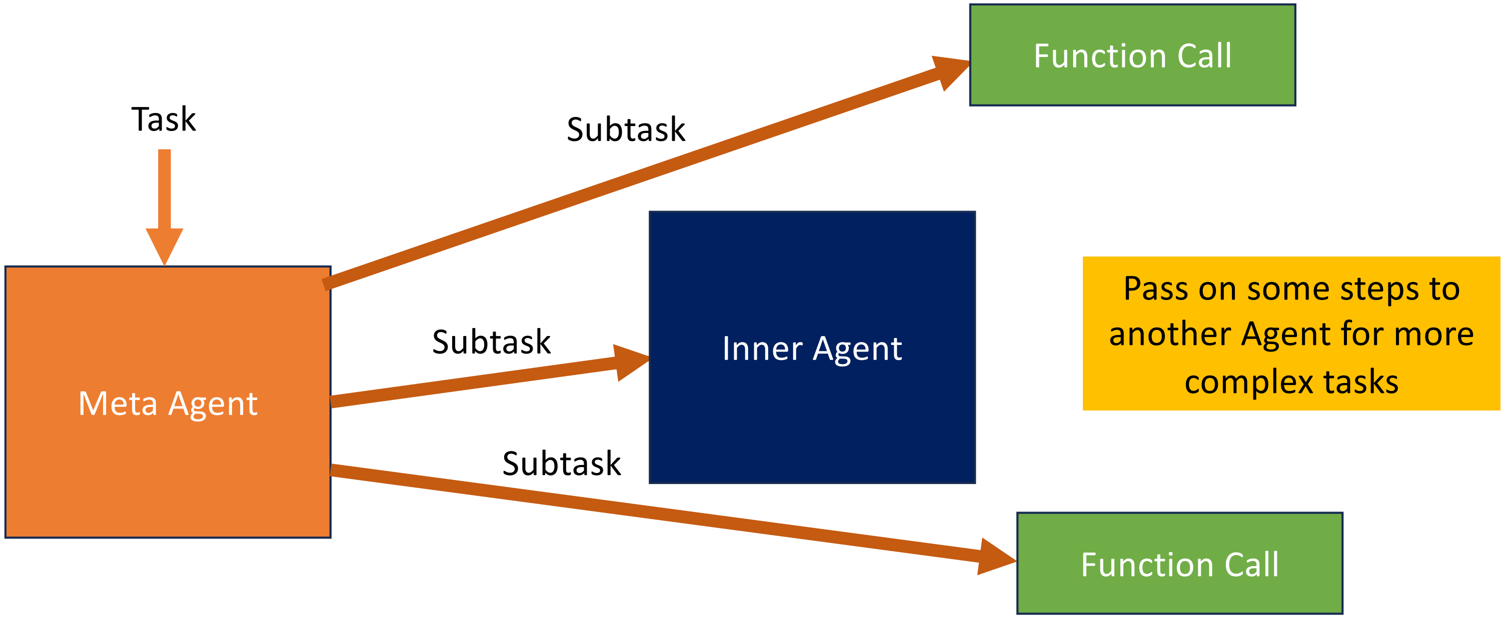
Many Potential Actions + Potential Outcomes

Key Question: How does an agent know what to do?



Filtered Action Space
by Skill Sets / Equipped Functions

Key Question: How does an agent know what to do?



Task

Meta Agent

Subtask

Function Call

Subtask

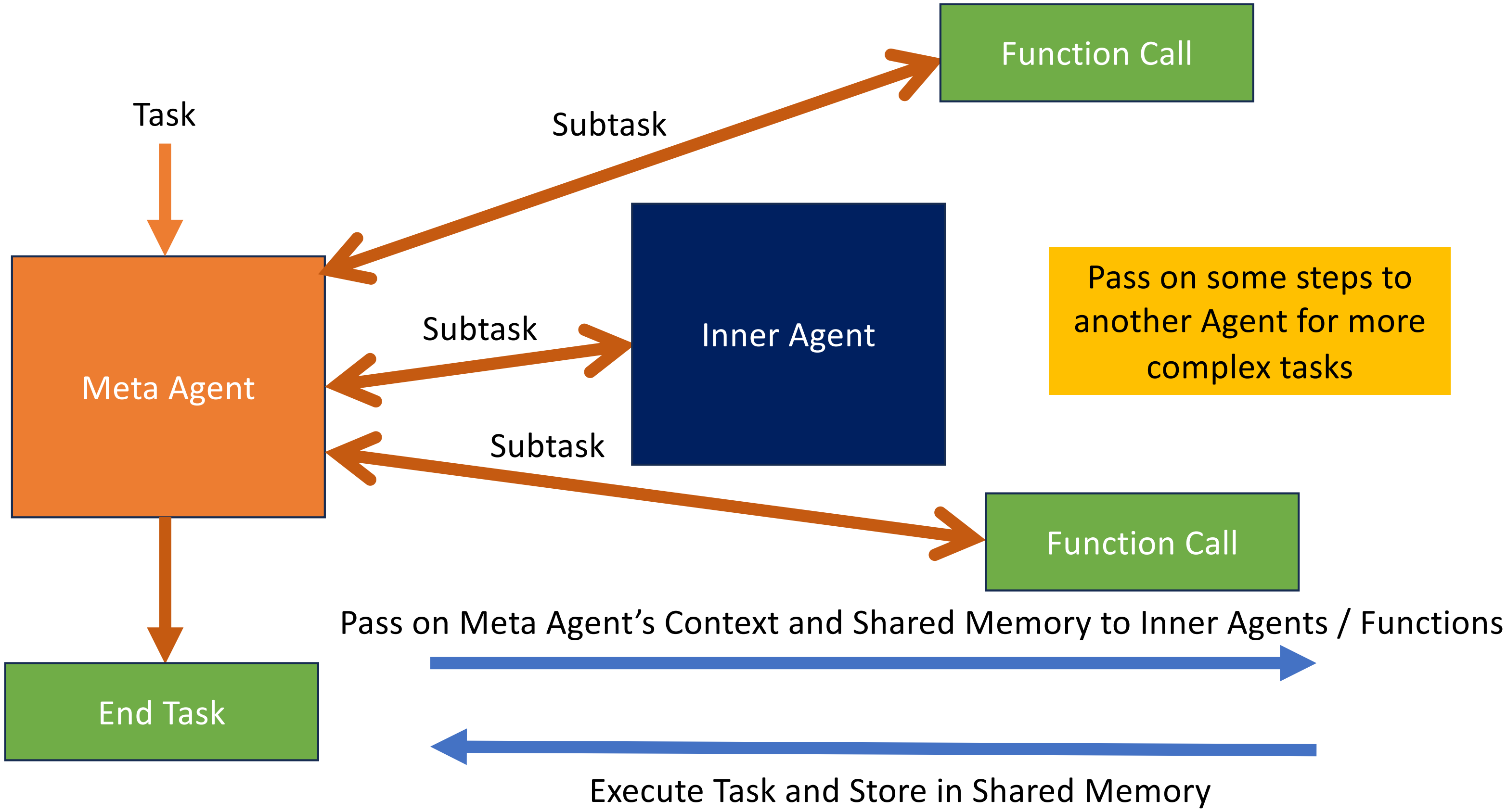
Inner Agent

Subtask

Function Call

Pass on some steps to another Agent for more complex tasks

Key Question: How does an agent know what to do?


Task
Subtask
Function Call
Meta Agent
Subtask
Inner Agent
Pass on some steps to another Agent for more complex tasks
Subtask
Function Call
End Task
Pass on Meta Agent's Context and Shared Memory to Inner Agents / Functions
Execute Task and Store in Shared Memory

TaskGen design Philosophies

Instructions as concise as possible for minimal token use

One Subtask mapped to One Equipped Function / Inner Agent

An Agent can only call an Equipped that is not above it in the hierarchy

Each Agent gets context and Equipped Functions relevant to its own processing abstraction space

Information shared between Agents / Equipped Function on a need-to-know basis

Future Plans - AgentJo

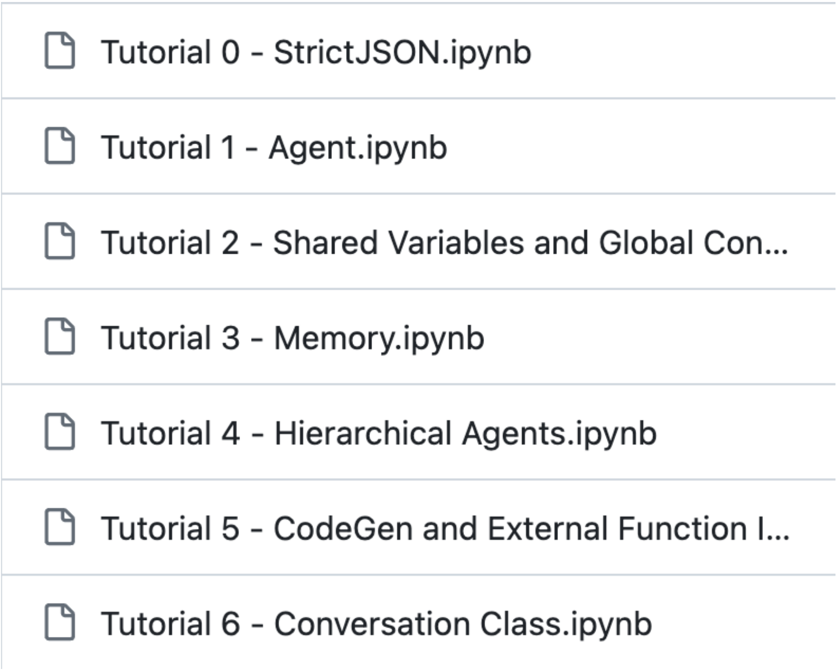
Follow-up work building on TaskGen

https://github.com/tanchongmin/agentjo

Current Features of AgentJo

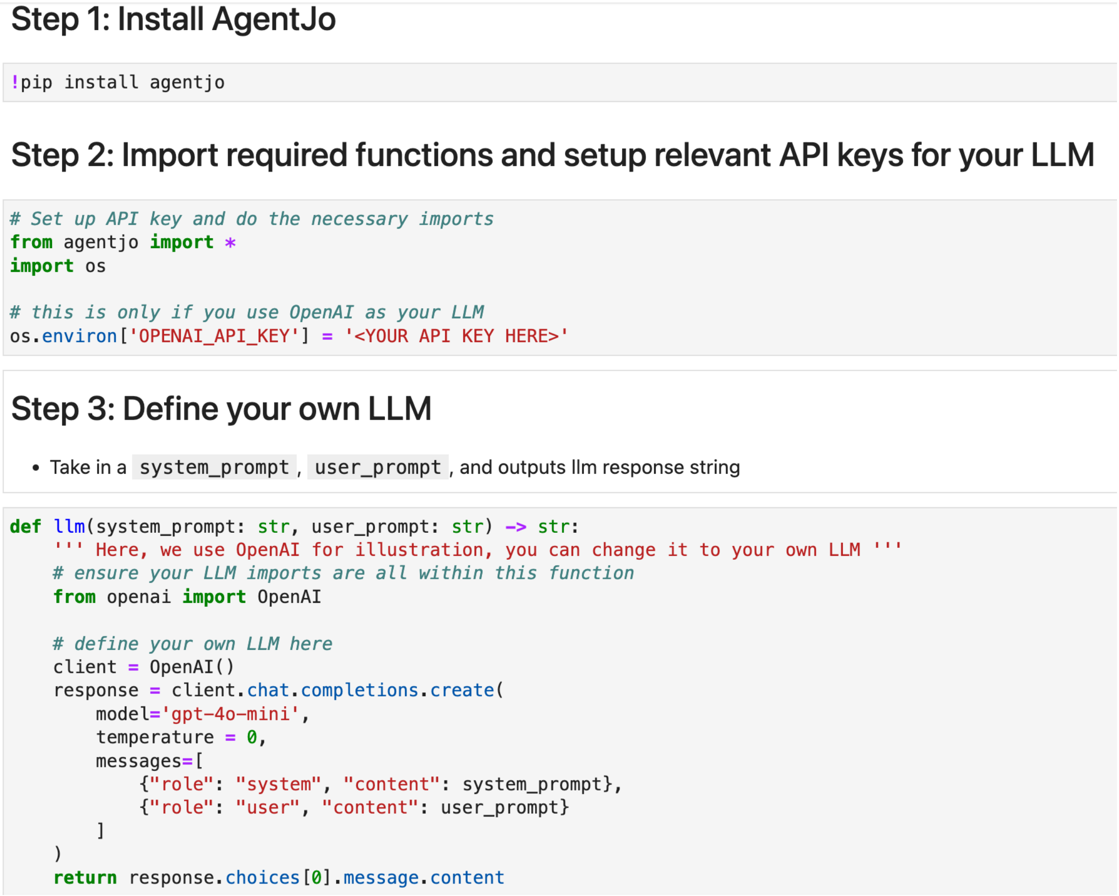
- Splitting of Tasks into subtasks for bite-sized solutions for each subtask
- Tutorial 1 - Agent with **Equipped Functions**
- Tutorial 2 - **Shared Variables** and **Global Context** for sharing of information
- Tutorial 3 - **Memory** that appending and retrieval of functions, file chunks, other information types dynamically according to task
- Tutorial 4 - Meta Agent with **Inner Agents**
- Tutorial 5 - **Code Generation** Interface
- Tutorial 6 - **Conversational** Interface

Refer to Tutorials at
https://github.com/tanchongmin/agentjo



Tutorial 0 - StrictJSON.ipynb
Tutorial 1 - Agent.ipynb
Tutorial 2 - Shared Variables and Global Con...
Tutorial 3 - Memory.ipynb
Tutorial 4 - Hierarchical Agents.ipynb
Tutorial 5 - CodeGen and External Function I...
Tutorial 6 - Conversation Class.ipynb

Setup in 3 easy steps



Step 1: Install AgentJo

```
!pip install agentjo
```

Step 2: Import required functions and setup relevant API keys for your LLM

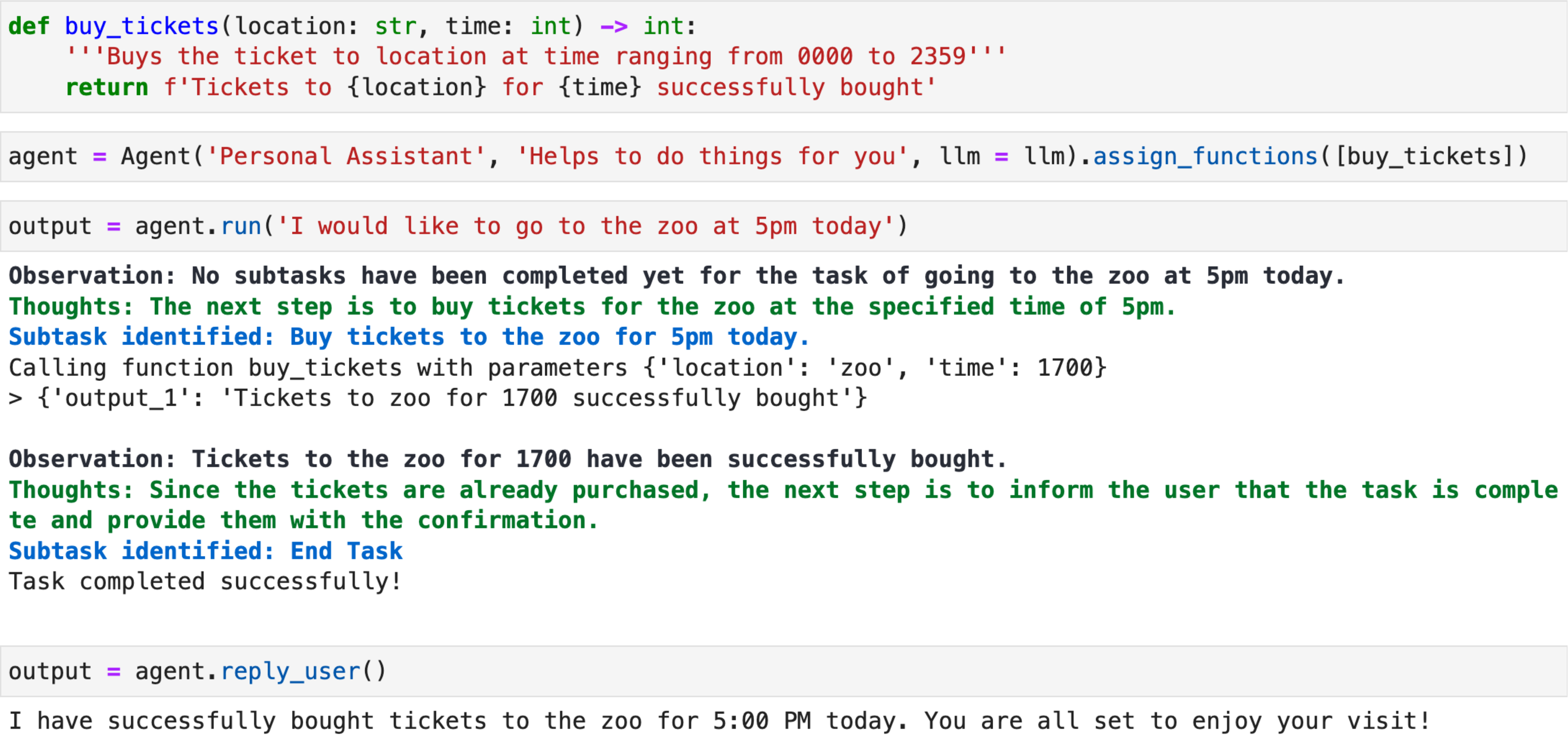
```
# Set up API key and do the necessary imports
from agentjo import *
import os

# this is only if you use OpenAI as your LLM
os.environ['OPENAI_API_KEY'] = '<YOUR API KEY HERE>'
```

Step 3: Define your own LLM

- Take in a `system_prompt`, `user_prompt`, and outputs llm response string

```
def llm(system_prompt: str, user_prompt: str) -> str:
    ''' Here, we use OpenAI for illustration, you can change it to your own LLM '''
    # ensure your LLM imports are all within this function
    from openai import OpenAI

    # define your own LLM here
    client = OpenAI()
    response = client.chat.completions.create(
        model='gpt-4o-mini',
        temperature = 0,
        messages=[
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_prompt}
        ]
    )
    return response.choices[0].message.content
```

Sample Agent Usage



```
def buy_tickets(location: str, time: int) -> int:
    '''Buys the ticket to location at time ranging from 0000 to 2359'''
    return f'Tickets to {location} for {time} successfully bought'
```

```
agent = Agent('Personal Assistant', 'Helps to do things for you', llm = llm).assign_functions([buy_tickets])
```

```
output = agent.run('I would like to go to the zoo at 5pm today')
```

```
Observation: No subtasks have been completed yet for the task of going to the zoo at 5pm today.
Thoughts: The next step is to buy tickets for the zoo at the specified time of 5pm.
Subtask identified: Buy tickets to the zoo for 5pm today.
Calling function buy_tickets with parameters {'location': 'zoo', 'time': 1700}
> {'output_1': 'Tickets to zoo for 1700 successfully bought'}

Observation: Tickets to the zoo for 1700 have been successfully bought.
Thoughts: Since the tickets are already purchased, the next step is to inform the user that the task is comple
te and provide them with the confirmation.
Subtask identified: End Task
Task completed successfully!
```

```
output = agent.reply_user()
```

```
I have successfully bought tickets to the zoo for 5:00 PM today. You are all set to enjoy your visit!
```

Comparison between LangChain / LlamaIndex / CrewAI / AutoGen / OpenAI Swarm / AgentJo

- **LangChain** / **LlamaIndex** does not have native JSON parsing – the JSON formatters implemented in LangChain use Pydantic which is too verbose

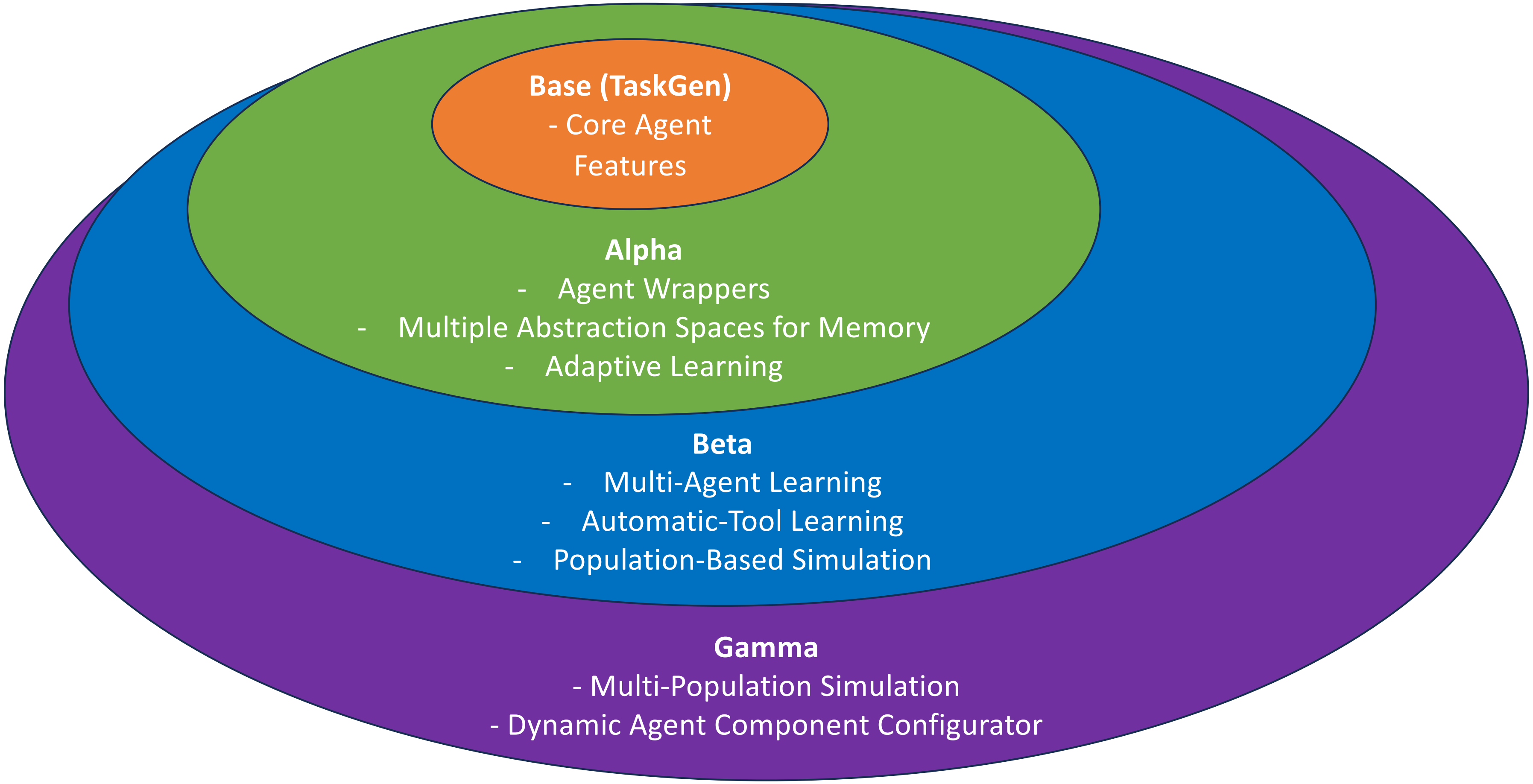
- **LangGraph** has non-natural graph forming and compilation

- **Crew.ai** and **AutoGen** use conversation as a way to problem solve – may take very long to solve problems

- **OpenAI Swarm** only for OpenAI API (uses *Shared Variables* like **AgentJo**)

- **AgentJo** is possibly the easiest to use out of all agentic frameworks

AgentJo - Future Plans



Base (TaskGen)
- Core Agent Features

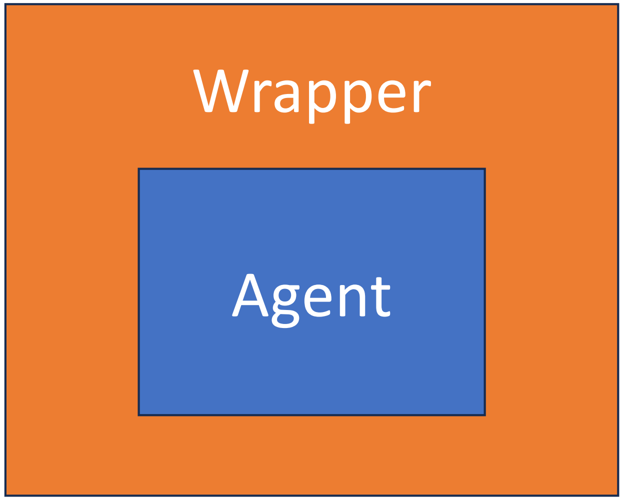
Alpha
- Agent Wrappers
- Multiple Abstraction Spaces for Memory
- Adaptive Learning

Beta
- Multi-Agent Learning
- Automatic-Tool Learning
- Population-Based Simulation

Gamma
- Multi-Population Simulation
- Dynamic Agent Component Configurator

Agent Wrappers

- Base Agent functionality is kept simple to minimise overhead
- Agents are meant to be modular and many can be “spawned” for usage in various pipelines
- For different tasks, can augment with extra functionalities via wrappers:
  - **PlanningWrappers:** How to plan and execute the plan
  - **ReflectionWrappers:** How to reflect and learn
  - **VerifierWrappers:** How to verify agent's outputs
  - **ConversationWrappers:** How to converse with the agent
  - **MultiAgentWrappers:** How multiple agents can converse

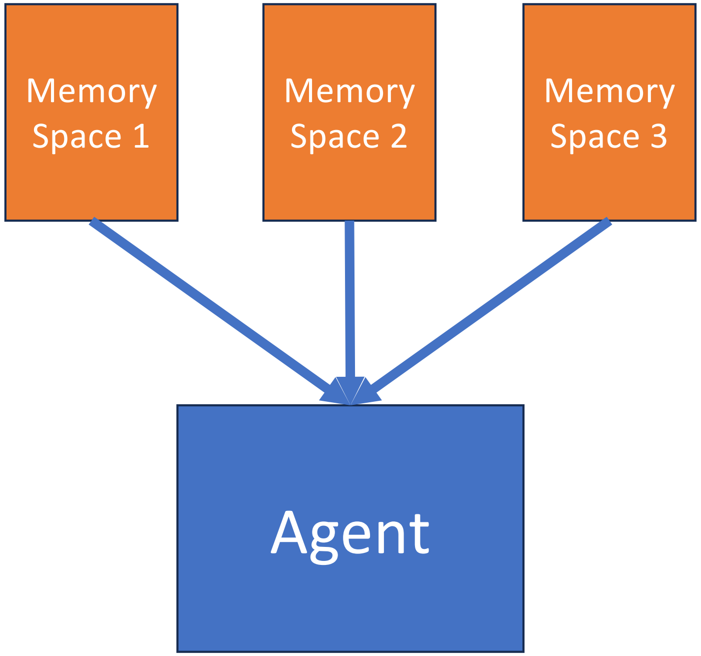


Wrapper

Agent

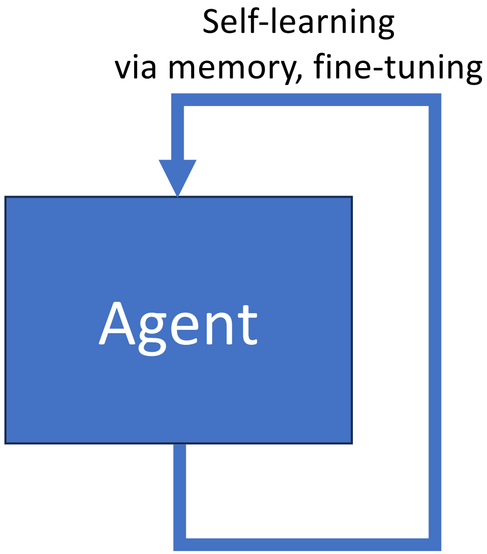
Memory Abstraction Spaces

- Memory is important for learning
- Memory is stored in different abstraction spaces, different modality
- Retrieve what is needed at each space to solve the task



Adaptive Learning

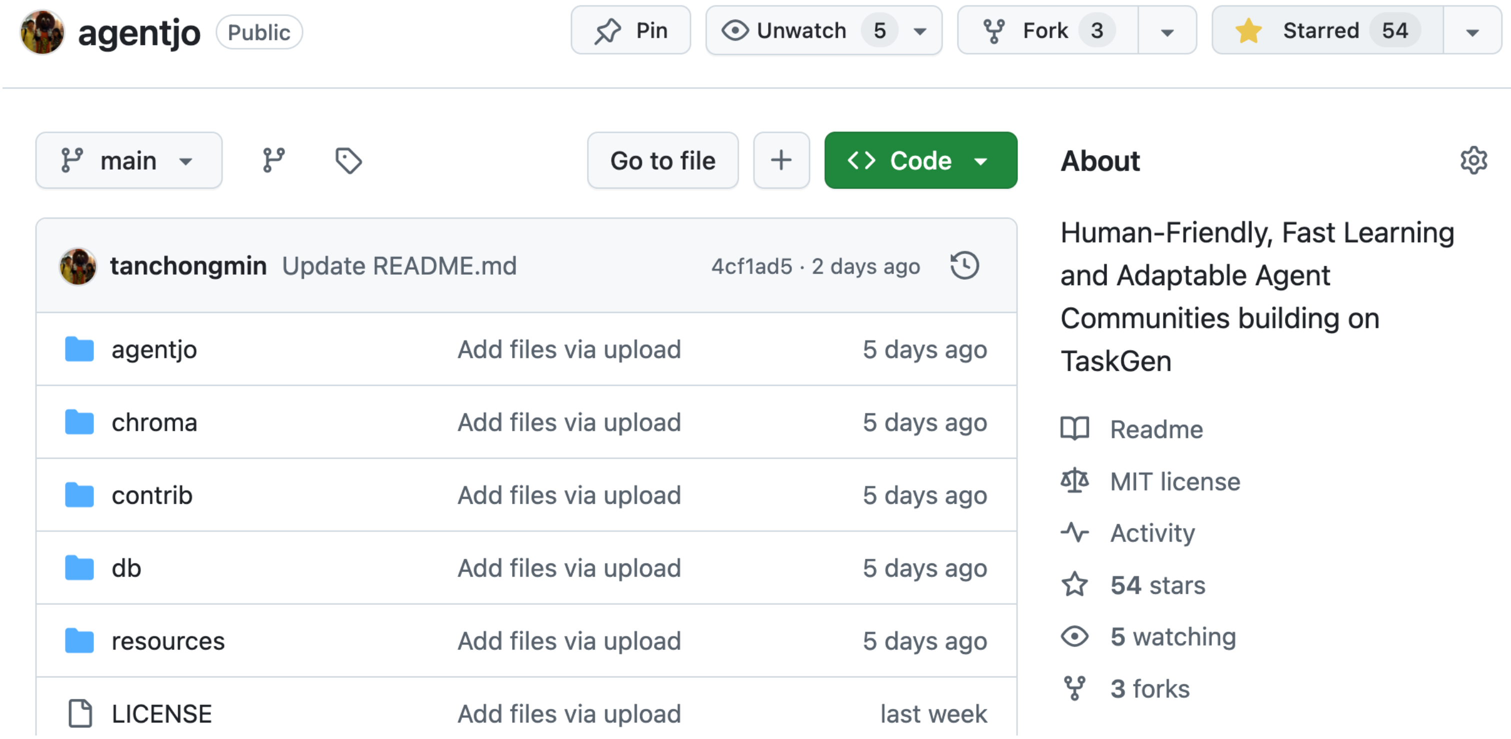
- Agent is able to consolidate and store reflections in memory/fine-tuning, and use it for future tasks
- Agent is able to configure its own functions, context according to need
- Agent is able to learn **within a task**, and **through tasks**



Multi-Agent Learning

- Each agent interacts with others and **shares knowledge**
- **Not all knowledge is shared** with everyone, only some of knowledge shared with neighbours if agent is performant
- Agents intentionally **kept different** and not homogeneous so that there is adaptability should environment change

Discussion



tanchongmin	Update README.md	4cf1ad5 · 2 days ago
agentjo	Add files via upload	5 days ago
chroma	Add files via upload	5 days ago
contrib	Add files via upload	5 days ago
db	Add files via upload	5 days ago
resources	Add files via upload	5 days ago
LICENSE	Add files via upload	last week

About

Human-Friendly, Fast Learning and Adaptable Agent Communities building on TaskGen

- Readme
- MIT license
- Activity
- 54 stars
- 5 watching
- 3 forks

https://github.com/tanchongmin/agentjo