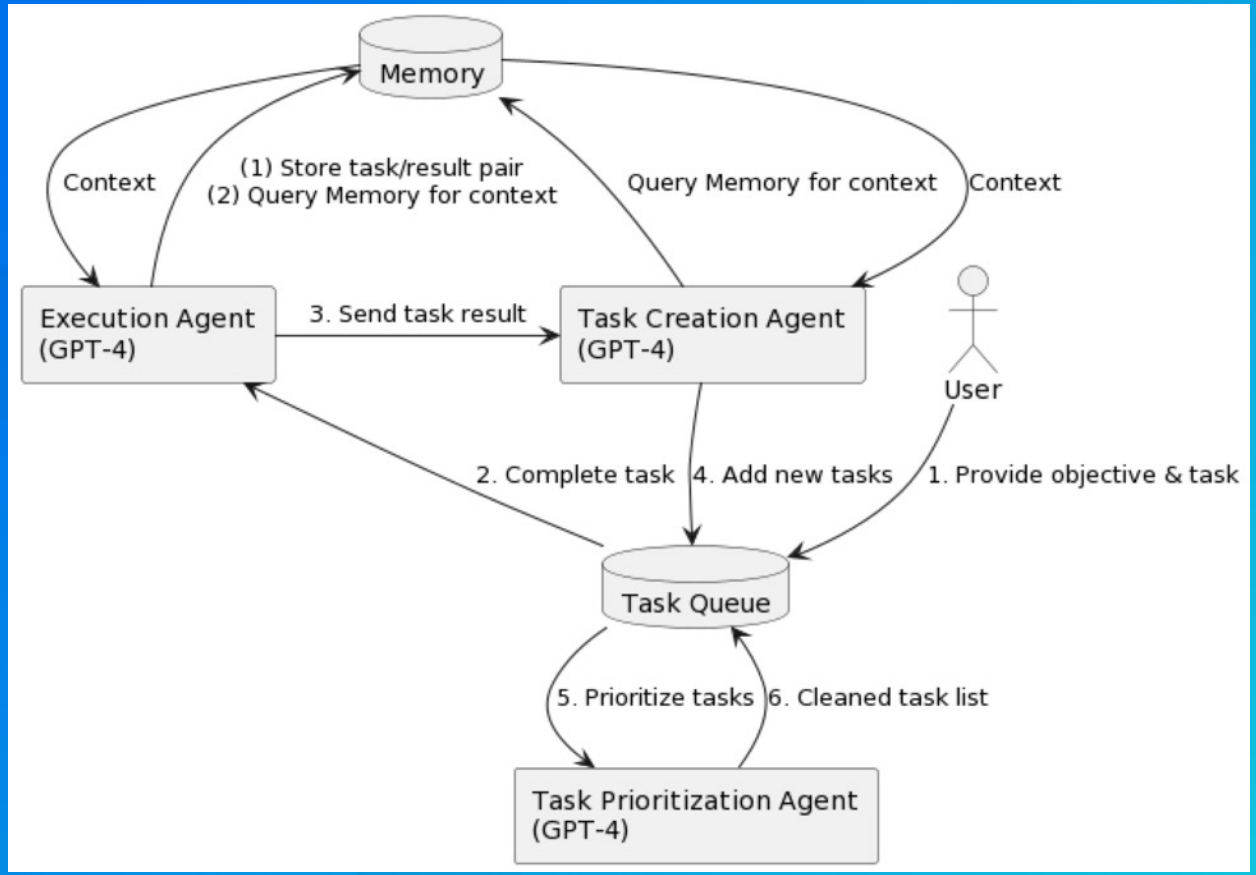
O'REILLY®

Building an Autonomous Agent

Evolution of BabyAGI - Yohei Nakajima

Who Am I?

- **Yohei Nakajima** (@yoheinakajima)
- **General Partner** at **Untapped Capital**
- I build to learn about new technologies
- Have been building on top of OpenAI API since August ‘22
- Creator of BabyAGI, which kickstarted the autonomous agent revolution in March.
- Today, I’m going to walk you through BabyAGI and it’s evolution since.


Memory
Context
(1) Store task/result pair
(2) Query Memory for context
Query Memory for context
Context
Execution Agent
(GPT-4)
3. Send task result
Task Creation Agent
(GPT-4)
User
2. Complete task
4. Add new tasks
1. Provide objective & task
Task Queue
5. Prioritize tasks
6. Cleaned task list
Task Prioritization Agent
(GPT-4)

```
#Set Variables
YOUR_TABLE_NAME = "test-table"
OBJECTIVE = "Solve world hunger."
YOUR_FIRST_TASK = "Develop a task list."
```

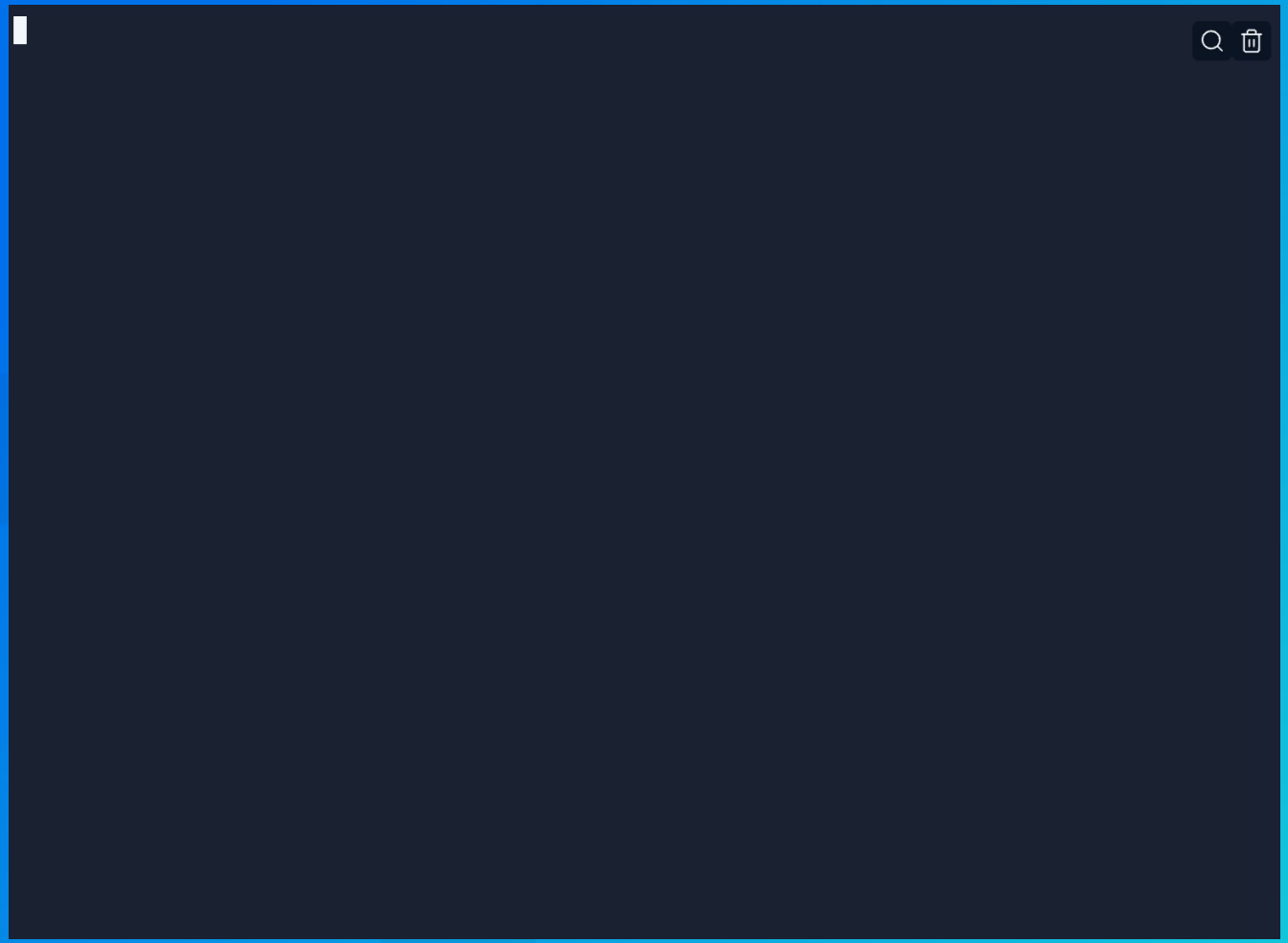
```
# Add the first task
first_task = {
    "task_id": 1,
    "task_name": YOUR_FIRST_TASK
}

add_task(first_task)
# Main loop
task_id_counter = 1
while True:
    if task_list:
        # Print the task list
        print("\033[95m\033[1m"+"\n*****TASK LIST*****\n"+"\033[0m\033[0m")
        for t in task_list:
            print(str(t['task_id'])+": "+t['task_name'])

        # Step 1: Pull the first task
        task = task_list.popleft()
        print("\033[92m\033[1m"+"\n*****NEXT TASK*****\n"+"\033[0m\033[0m")
        print(str(task['task_id'])+": "+task['task_name'])

        # Send to execution function to complete the task based on the context
        result = execution_agent(OBJECTIVE,task["task_name"])
        this_task_id = int(task["task_id"])
        print("\033[93m\033[1m"+"\n*****TASK RESULT*****\n"+"\033[0m\033[0m")
        print(result)

        # Step 2: Enrich result and store in Pinecone
        enriched_result = {'data': result}  # This is where you should enrich the result if needed
        result_id = f"result_{task['task_id']}"
        vector = enriched_result['data']  # extract the actual result from the dictionary
        index.upsert([(result_id, get_ada_embedding(vector),
{"task":task['task_name'],"result":result})])

    # Step 3: Create new tasks and reprioritize task list
    new_tasks = task_creation_agent(OBJECTIVE,enriched_result, task["task_name"], [t["task_name"]
for t in task_list])

    for new_task in new_tasks:
        task_id_counter += 1
        new_task.update({"task_id": task_id_counter})
        add_task(new_task)
    prioritization_agent(this_task_id)

time.sleep(1)  # Sleep before checking the task list again
```

```
def task_creation_agent(objective: str, result: Dict, task_description: str, task_list: List[str]):
    prompt = f"You are an task creation AI that uses the result of an execution agent to create new
tasks with the following objective: {objective}, The last completed task has the result: {result}.
This result was based on this task description: {task_description}. These are incomplete tasks: {',
'.join(task_list)}. Based on the result, create new tasks to be completed by the AI system that do
not overlap with incomplete tasks. Return the tasks as an array."
    response = openai.Completion.create(engine="text-davinci-
003",prompt=prompt,temperature=0.5,max_tokens=100,top_p=1,frequency_penalty=0,presence_penalty=0)
    new_tasks = response.choices[0].text.strip().split('\n')
    return [{"task_name": task_name} for task_name in new_tasks]
```

```
def prioritization_agent(this_task_id:int):
    global task_list
    task_names = [t["task_name"] for t in task_list]
    next_task_id = int(this_task_id)+1
    prompt = f"""You are an task prioritization AI tasked with cleaning the formatting of and
reprioritizing the following tasks: {task_names}. Consider the ultimate objective of your team:
{OBJECTIVE}. Do not remove any tasks. Return the result as a numbered list, like:
    #. First task
    #. Second task
    Start the task list with number {next_task_id}."""
    response = openai.Completion.create(engine="text-davinci-
003",prompt=prompt,temperature=0.5,max_tokens=1000,top_p=1,frequency_penalty=0,presence_penalty=0)
    new_tasks = response.choices[0].text.strip().split('\n')
    task_list = deque()
    for task_string in new_tasks:
        task_parts = task_string.strip().split(".", 1)
        if len(task_parts) == 2:
            task_id = task_parts[0].strip()
            task_name = task_parts[1].strip()
            task_list.append({"task_id": task_id, "task_name": task_name})
```

```
def execution_agent(objective:str,task: str) -> str:
    context=context_agent(index=YOUR_TABLE_NAME, query=objective, n=5)
    response = openai.Completion.create(
        engine="text-davinci-003",
        prompt=f"You are an AI who performs one task based on the following objective: {objective}.
Your task: {task}\nResponse:",
        temperature=0.7,
        max_tokens=2000,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0
    )
    return response.choices[0].text.strip()
```

```
def context_agent(query: str, index: str, n: int):
    query_embedding = get_ada_embedding(query)
    index = pinecone.Index(index_name=index)
    results = index.query(query_embedding, top_k=n,
    include_metadata=True)
    sorted_results = sorted(results.matches, key=lambda x: x.score, reverse=True)
    return [(str(item.metadata['task'])) for item in sorted_results]
```



BabyBeeAGI

A slower, buggier, but more powerful modification of the OG BabyAGI

Work-in-Progress
(Errors a lot)

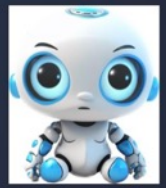
Upgraded Task Manager
(Combined prioritization & task creation)

Task Serialization

Perpetual Summary

Local Task List
(No Embeddings)

200 lines of code
(~300 total w spaces, prints, and comments)



Task Creation Agent

```
    prompt = f"You are an task creation AI that uses the
result of an execution agent to create new tasks with the
following objective: {objective}, The last completed task
has the result: {result}. This result was based on this task
description: {task_description}. These are incomplete tasks:
{', '.join(task_list)}. Based on the result, create new
tasks to be completed by the AI system that do not overlap
with incomplete tasks. Return the tasks as an array."
```

Task Prioritization Agent

```
    prompt = f"""You are an task prioritization AI tasked
with cleaning the formatting of and reprioritizing the
following tasks: {task_names}. Consider the ultimate
objective of your team:{OBJECTIVE}. Do not remove any tasks.
Return the result as a numbered list, like:
    #. First task
    #. Second task
    Start the task list with number {next_task_id}."""
```

Task Management Agent

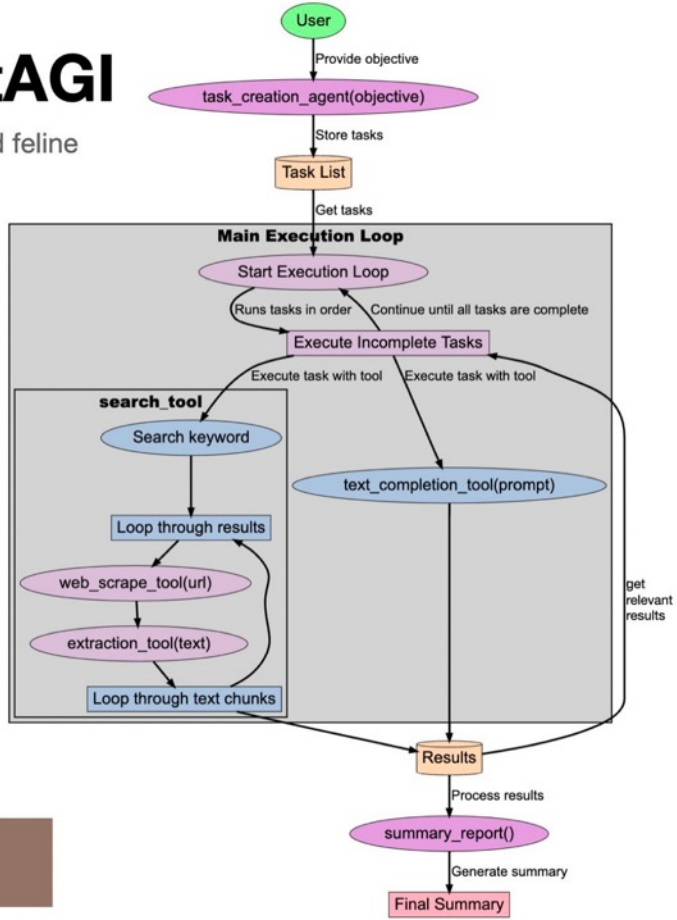
```
    prompt = (
        f"You are a task management AI tasked with cleaning the formatting of and
reprioritizing the following tasks: {minified_task_list}. "
        f"Consider the ultimate objective of your team: {OBJECTIVE}. "
        f"Do not remove any tasks. Return the result as a JSON-formatted list of
dictionaries.\n"
        f"Create new tasks based on the result of last task if necessary for the objective.
Limit tasks types to those that can be completed with the available tools listed below.
Task description should be detailed."
        f"The maximum task list length is 7. Do not add an 8th task."
        f"The last completed task has the following result: {result}. "
        f"Current tool option is [text-completion] {websearch_var} and [web-scrape] only."#
web-search is added automatically if SERPAPI exists
        f"For tasks using [web-scrape], provide only the URL to scrape as the task
description. Do not provide placeholder URLs, but use ones provided by a search step or the
initial objective."
        #f"If the objective is research related, use at least one [web-search] with the
query as the task description, and after, add up to three URLs from the search result as a
task with [web-scrape], then use [text-completion] to write a comprehensive summary of each
site thas has been scraped.'"
        f"For tasks using [web-search], provide the search query, and only the search query
to use (eg. not 'research waterproof shoes, but 'waterproof shoes')"
        f"dependent_task_id should always be null or a number."
        f"Do not reorder completed tasks. Only reorder and dedupe incomplete tasks.\n"
        f"Make sure all task IDs are in chronological order.\n"
        f"Do not provide example URLs for [web-scrape].\n"
        f"Do not include the result from the last task in the JSON, that will be added
after..\n"
        f"The last step is always to provide a final summary report of all tasks.\n"
        f"An example of the desired output format is: "
        "[{\"id\": 1, \"task\": \"https://untapped.vc\", \"tool\": \"web-scrape\",
\"dependent_task_id\": null, \"status\": \"incomplete\", \"result\": null,
\"result_summary\": null}, {\"id\": 2, \"task\": \"Analyze the contents of...\", \"tool\":
\"text-completion\", \"dependent_task_id\": 1, \"status\": \"incomplete\", \"result\":
null, \"result_summary\": null}, {\"id\": 3, \"task\": \"Untapped Capital\", \"tool\":
\"web-search\", \"dependent_task_id\": null, \"status\": \"incomplete\", \"result\": null,
\"result_summary\": null}]."
    )
```

BabyCatAGI

Faster, fiercer, and feline

- Task Creation Agent runs once
- Execution Agent loops through tasks
- Task dependencies for pulling relevant results
- Two tools: search_tool and text_completion
- “Mini-agent” as tool
  - Search tool combines search, scrape, chunking, and extraction.
- Results combined to create summary_report

~300 lines of code



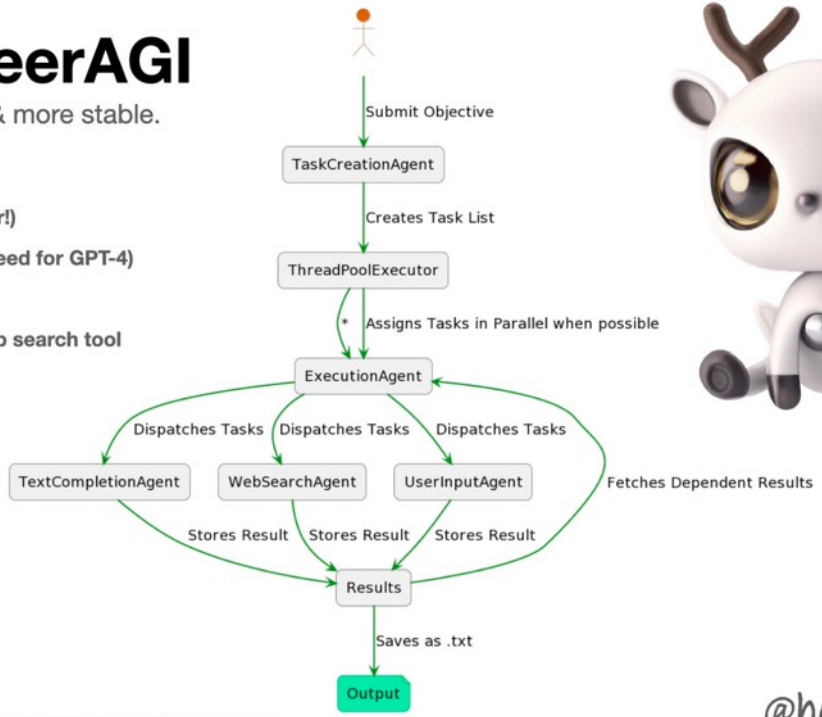
```
*****OBJECTIVE*****

Research recent AI news and write a poem about your findings.
```

```
*****TASK LIST*****

1: AI news 2022 [complete] [web-search]
2: Extract key points from the AI news articles found [complete] [text-completion] <dependencies: #1>
3: AI advancements in 2022 [complete] [web-search]
4: Summarize the advancements in AI in 2022 from the articles found [complete] [text-completion] <dependencies: #3>
5: Write a poem about the recent AI news and advancements [complete] [text-completion] <dependencies: #2, #4>
6: Provide a final summary report including tasks executed and summary of knowledge acquired [incomplete] [text-completion]
 <dependencies: #1, #2, #3, #4, #5>
```

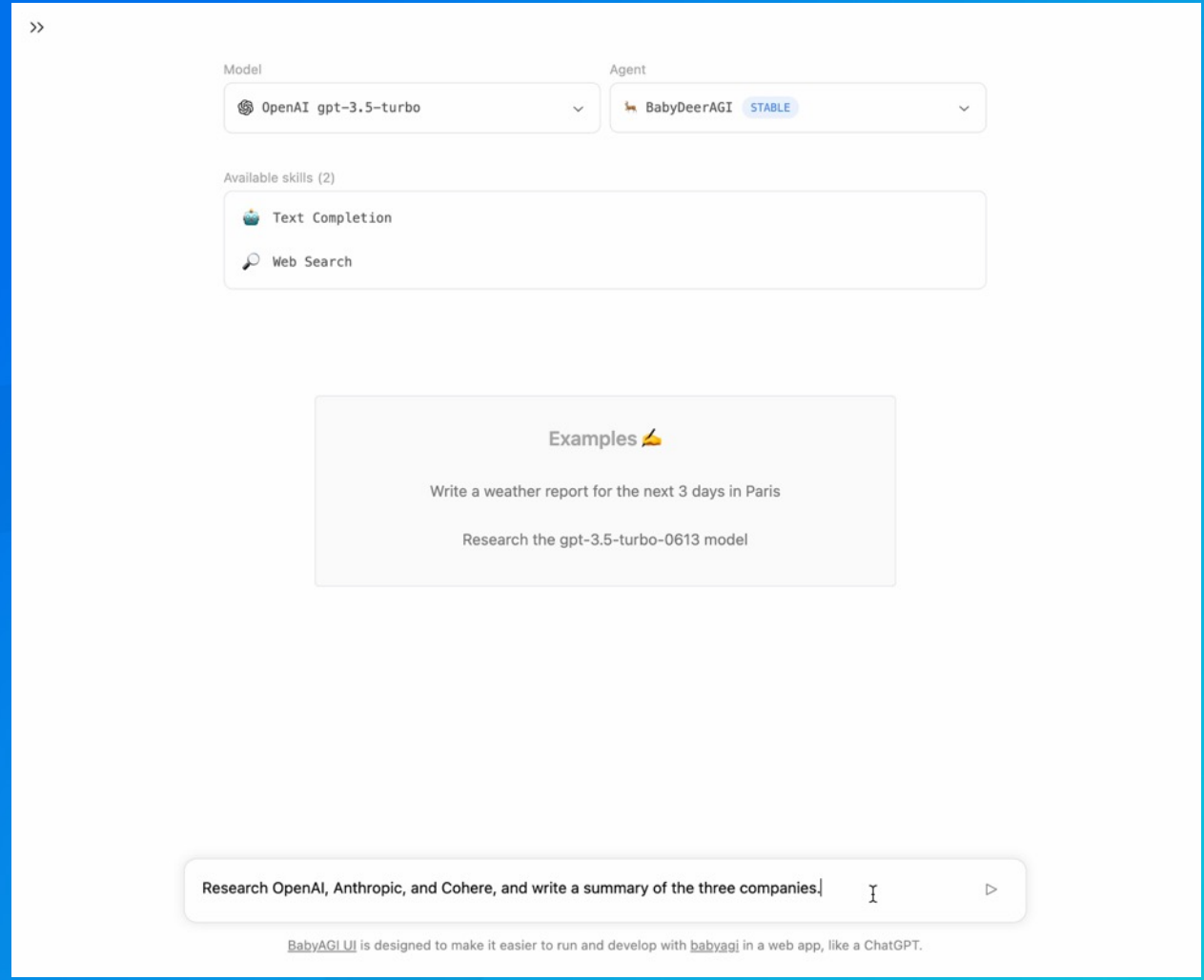
BabyDeerAGI

Faster, cheaper, & more stable.

- Parallel tasks (faster!)
- 3.5-turbo only (no need for GPT-4)
- User input tool
- Query rewrite in web search tool
- Saves results



@babyagi_


Model
OpenAI gpt-3.5-turbo
Agent
BabyDeerAGI
STABLE
Available skills (2)
Text Completion
Web Search
Examples ✍️
Write a weather report for the next 3 days in Paris
Research the gpt-3.5-turbo-0613 model
Research OpenAI, Anthropic, and Cohere, and write a summary of the three companies.
BabyAGI UI is designed to make it easier to run and develop with babyagi in a web app, like a ChatGPT.

BabyElfAGI

MAIN.PY

TASKS

- TASK_REGISTRY.PY

SKILLS

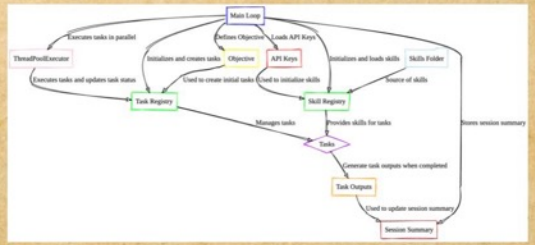
- SKILL_REGISTRY.PY
- SKILL.PY
- TEXT_COMPLETION.PY
- WEB_SEARCH.PY
- SKILL_SAVER.PY
- OBJECTIVE_SAVER.PY

Example skill

OUTPUTS

EXAMPLE OBJECTIVES

- EXAMPLE1.JSON
- EXAMPLE2.JSON
- EXAMPLE3.JSON



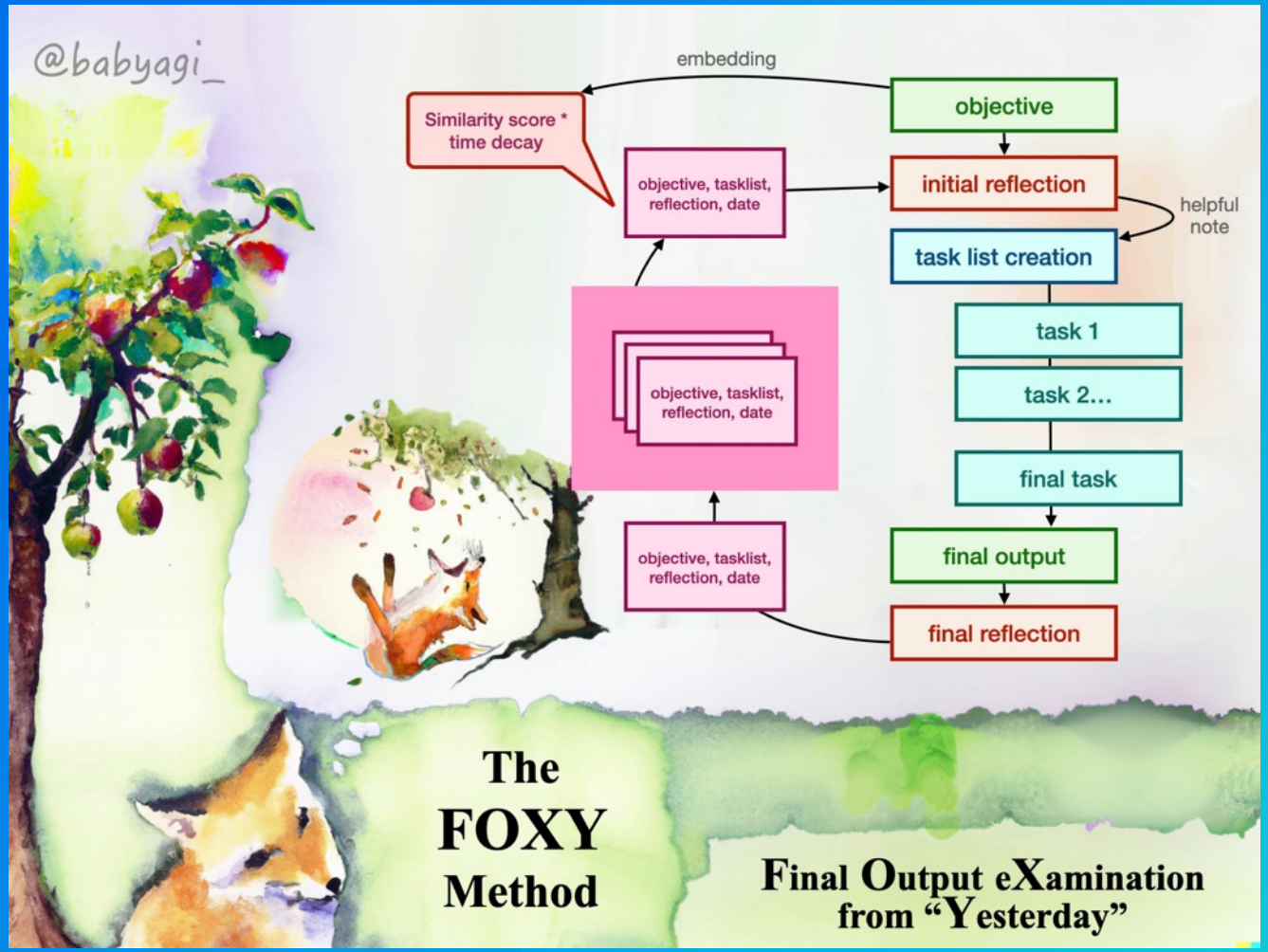
BabyElfAGI is able to read and write its own code, including writing new skills and example task lists.

```
>
```

```
*****OBJECTIVE*****
Read text_completion.py, look up documentation for SendGrid, then write a new
 skill using the two outputs (text_completion skill), then save this using th
e skill_saver skill.
```

```
*****TASK LIST*****
1: Read text_completion.py [incomplete] [code_reader]
2: Look up SendGrid documentation online [incomplete] [documentation_search]
3: Write a new skill using the outputs from tasks 1 and 2 [incomplete] [text_
completion] <dependencies: #1, #2>
4: Save the new skill [incomplete] [skill_saver] <dependencies: #3>
```

```python
from skills.skill import Skill
import openai
import sendgrid

class NewSkill(Skill):
    name = 'new_skill'
    description = "A new skill that combines text completion and SendGrid API to
generate and send emails."

    def __init__(self, api_keys, main_loop_function):
        super().__init__(api_keys, main_loop_function)

    def execute(self, params, dependent_task_outputs, objective):
        if not self.valid:
            return

        # Extract relevant information from dependent_task_outputs
        sendgrid_code = dependent_task_outputs[2]

        # Extract SendGrid API key from params
        sendgrid_api_key = params['sendgrid_api_key']

        # Create a new email using SendGrid API
        message = sendgrid.Mail(
            from_email='from@example.com',
            to_emails='to@example.com',
            subject='Sending with SendGrid',
            plain_text_content='This is the content of the email.'
        )

        # Set SendGrid API key
        sg = sendgrid.SendGridAPIClient(api_key=sendgrid_api_key)

        # Send the email
        response = sg.send(message)

        # Print the response
        print(response.status_code)
        print(response.body)
        print(response.headers)

        return "Email sent successfully."
```


@babyagi_
embedding
objective
Similarity score * time decay
objective, tasklist, reflection, date
initial reflection
helpful note
task list creation
task 1
task 2...
final task
objective, tasklist, reflection, date
objective, tasklist, reflection, date
final output
final reflection
The FOXY Method
Final Output eXamination from "Yesterday"

hey

Hey

Who won the world cup this year?

My name is Yohei

Please do.

What is your fees?

Type your message...

Send

Tasks and Output

QUESTIONS?

Follow me on Twitter at **@yoheinakajima**

Find BabyAGI on Github at:

https://github.com/yoheinakajima/babyagi

Find each version in the “classic” folder.


O'REILLY®