TaskGen

A Task-Based, Memory-Infused Agentic Framework building on StrictJSON

https://github.com/simbianai/taskgen

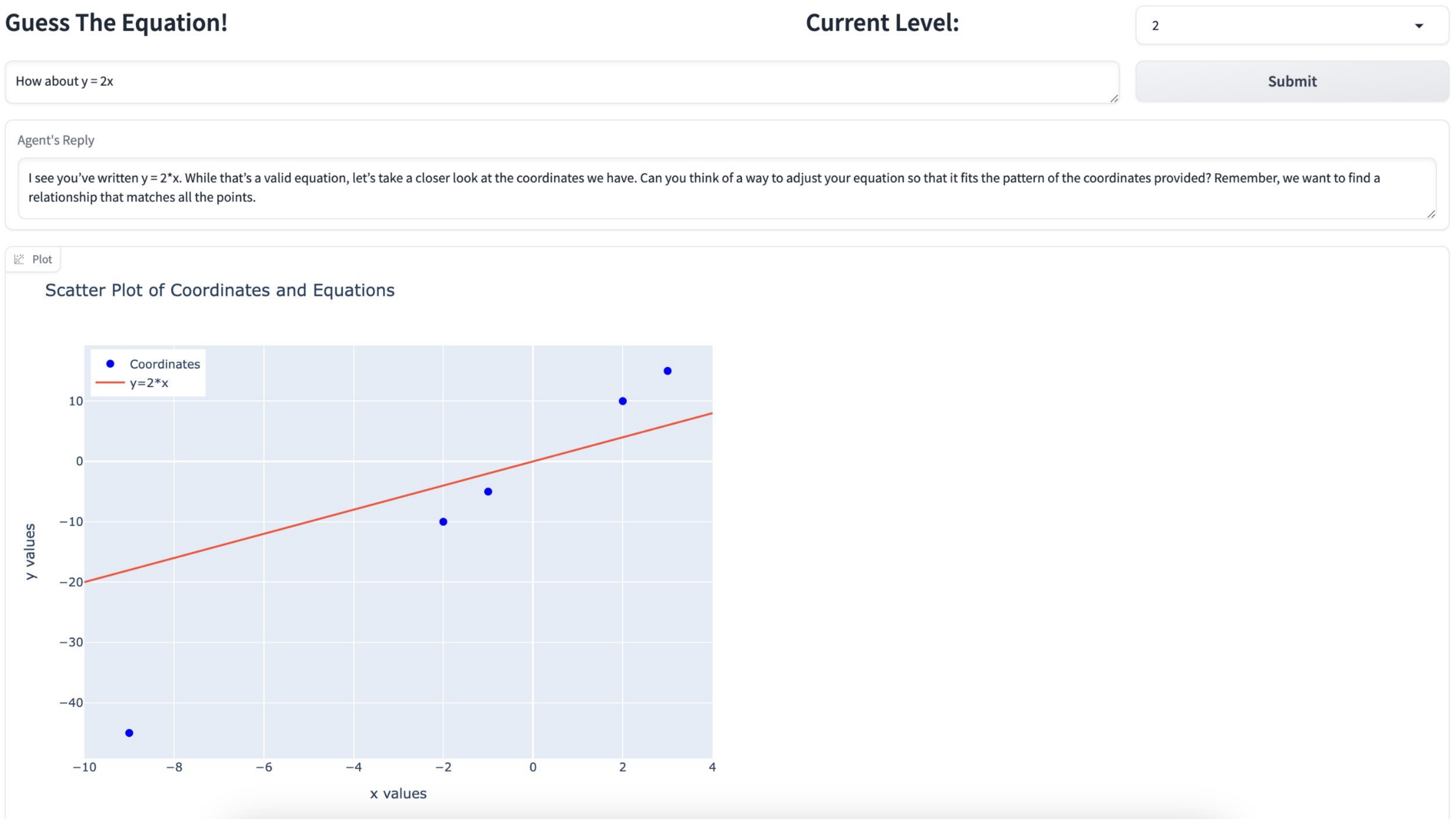
John Tan Chong Min

Prince Saroj, Hardik Maheshwari, Bharat Runwal

Brian Lim Yi Sheng, Richard Cottrill

Mehul Motani, Alankrit Chona, Ambuj Kumar

Example TaskGen Project – Equation Game



Guess The Equation!

Current Level: 2

How about y = 2x

Submit

Agent's Reply

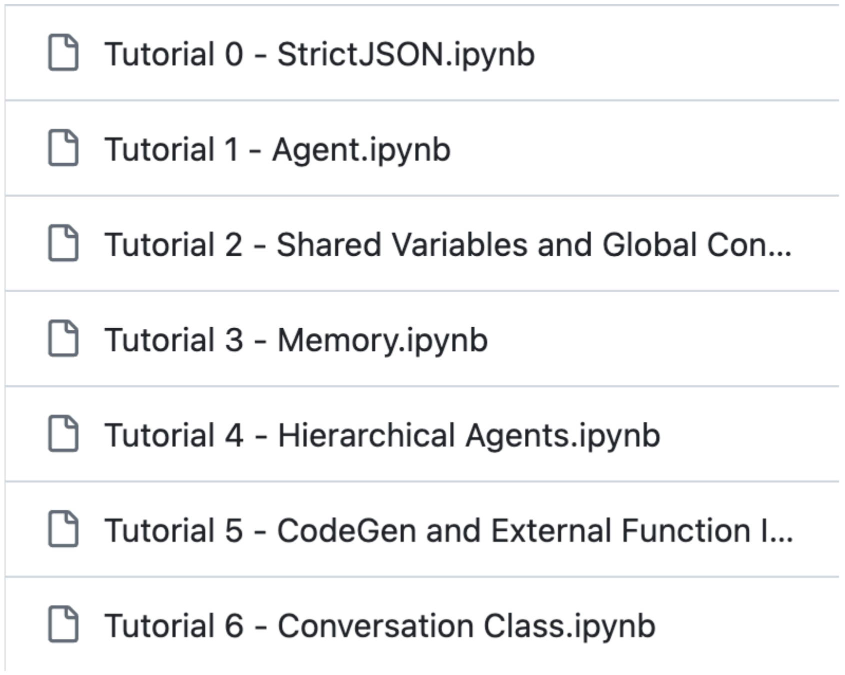
I see you've written y = 2*x. While that's a valid equation, let's take a closer look at the coordinates we have. Can you think of a way to adjust your equation so that it fits the pattern of the coordinates provided? Remember, we want to find a relationship that matches all the points.

https://github.com/simbianai/taskgen/blob/main/contrib/Fun_TaskGen_Projects/Equation_Game.ipynb

Features of TaskGen

- 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 appends and retrieves 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/simbianai/taskgen


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

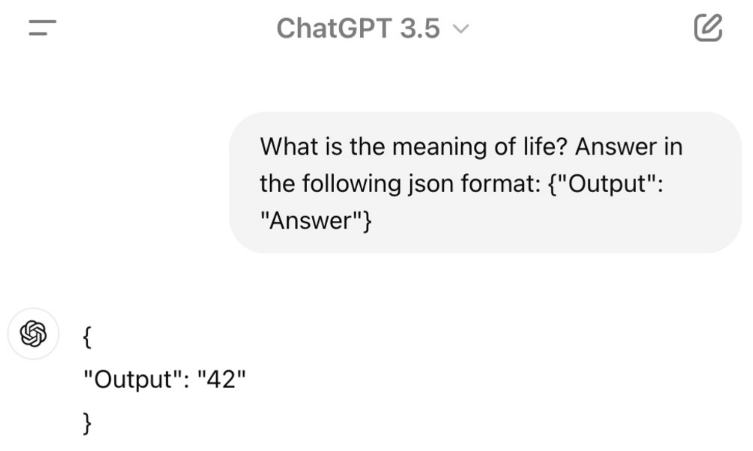
StrictJSON

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

https://github.com/tanchongmin/strictjson

Why StrictJSON?

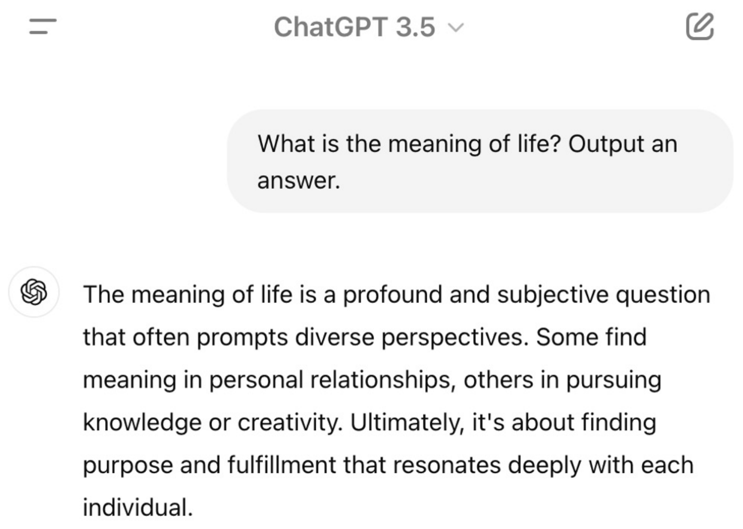
- JSON is a much less verbose method of output for LLM



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

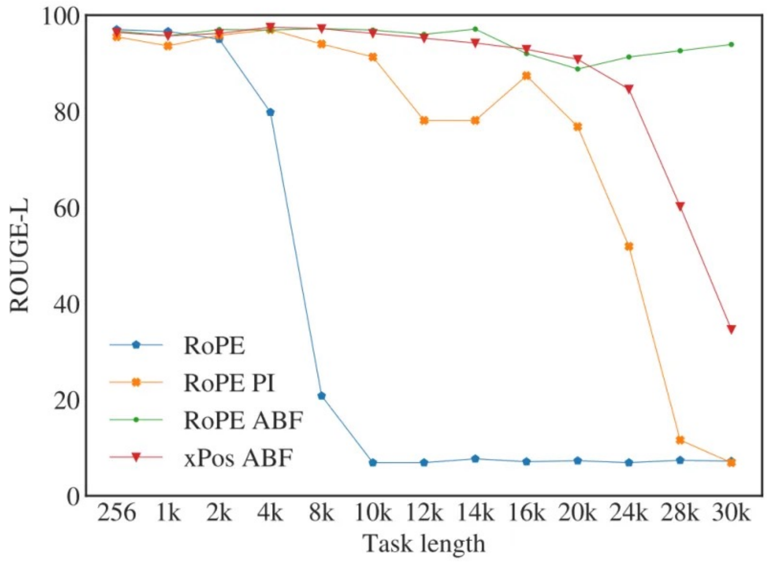
```
{
    "parameters": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA",
            },
            "format": {
                "type": "string",
                "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

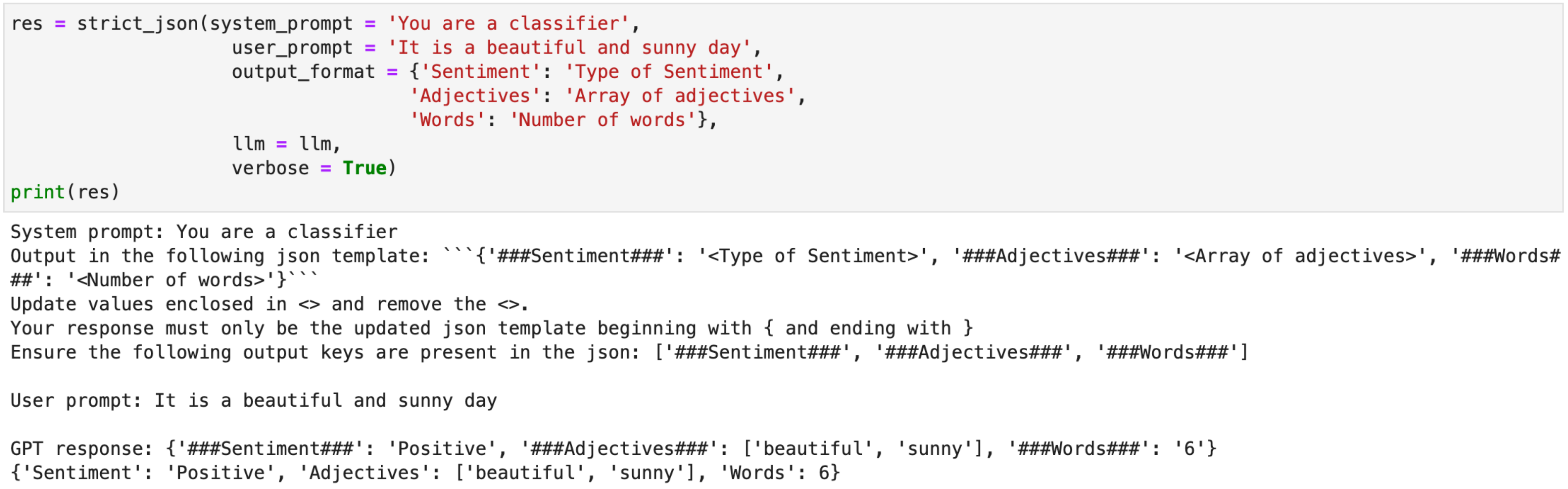
```
res = strict_json(system_prompt = 'You are a classifier',
                    user_prompt = 'It is a beautiful and sunny day',
                    output_format = {'Sentiment': 'Type of Sentiment, type: Enum["Pos", "Neg", "Other"]',
                                    'Adjectives': 'Array of adjectives, 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



```python
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)
```

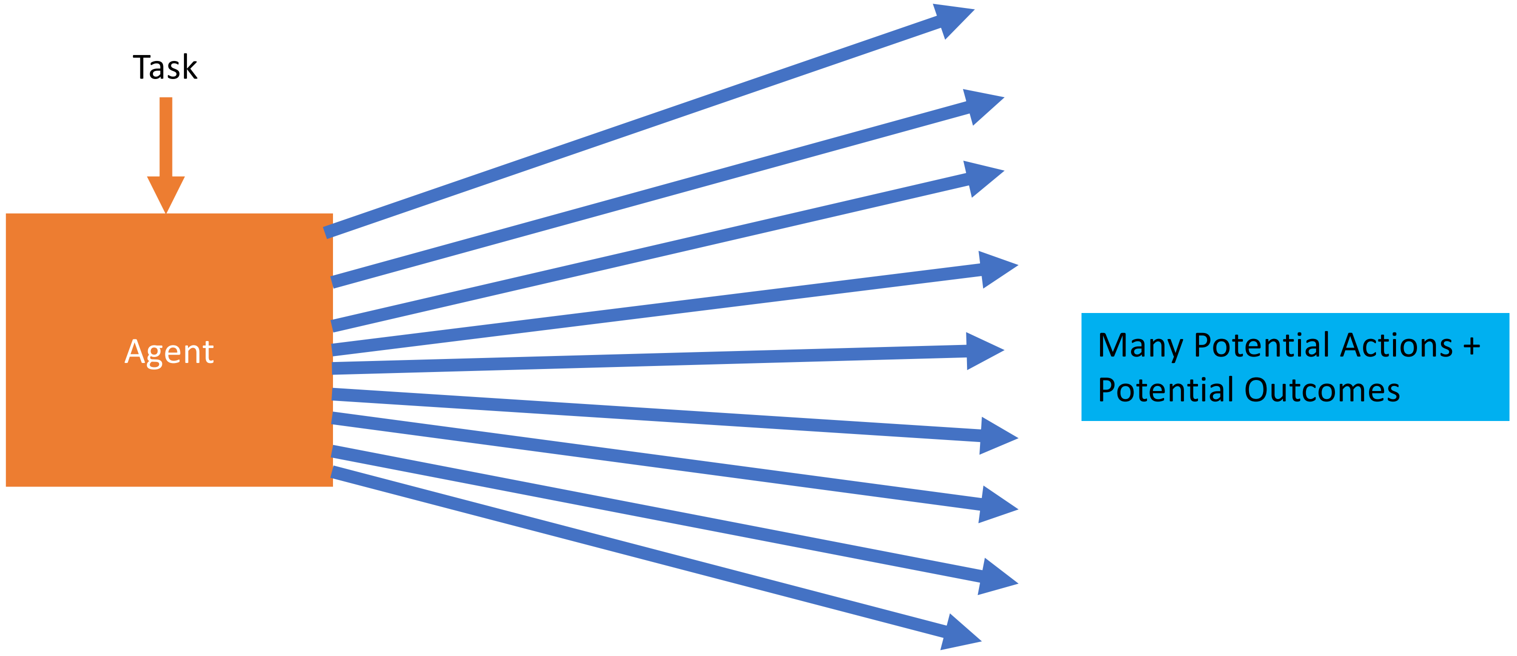
```
System prompt: You are a classifier
Output in the following json template: ```{'###Sentiment###': '<Type of Sentiment>', '###Adjectives###': '<Array of adjectives>', '###Words#
##': '<Number of words>'}```
Update values enclosed in <> and remove the <>.
Your response must only be the updated json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

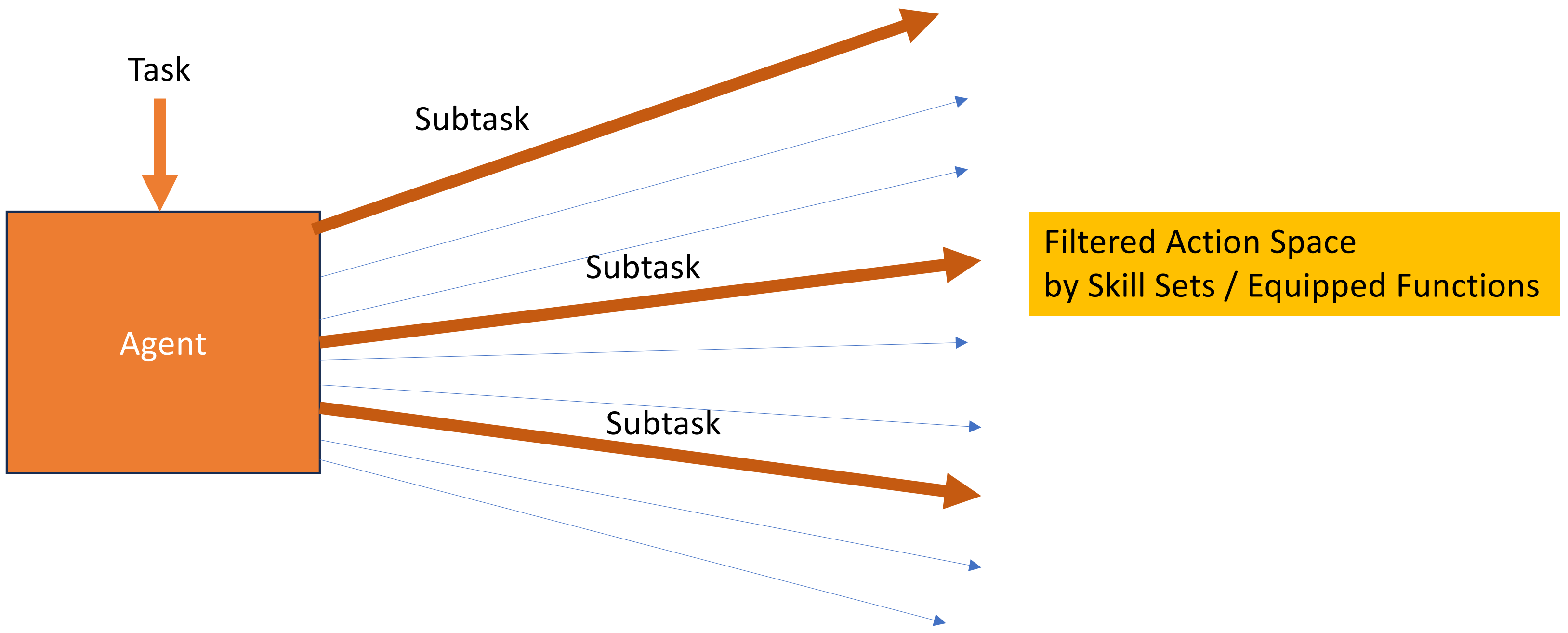
Building on StrictJSON for Agentic Task Solving!

https://github.com/simbianai/taskgen

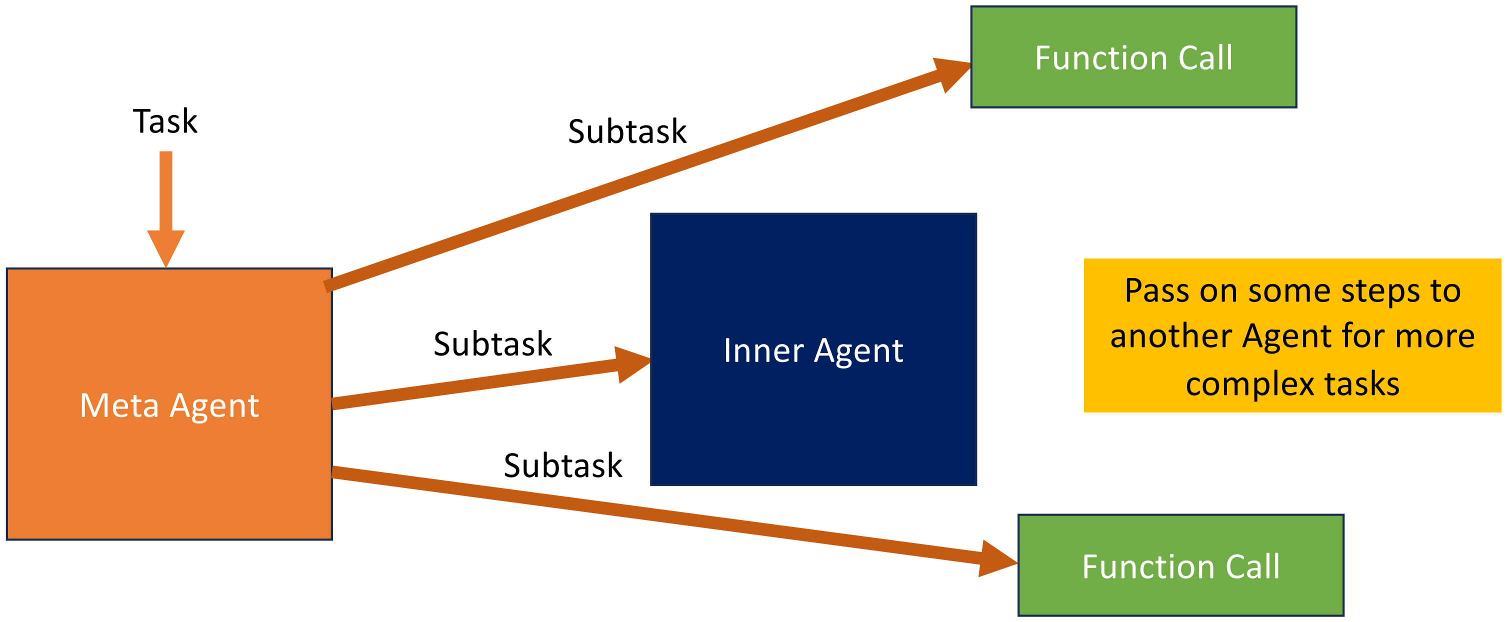
Key Question: How does an agent know what to do?



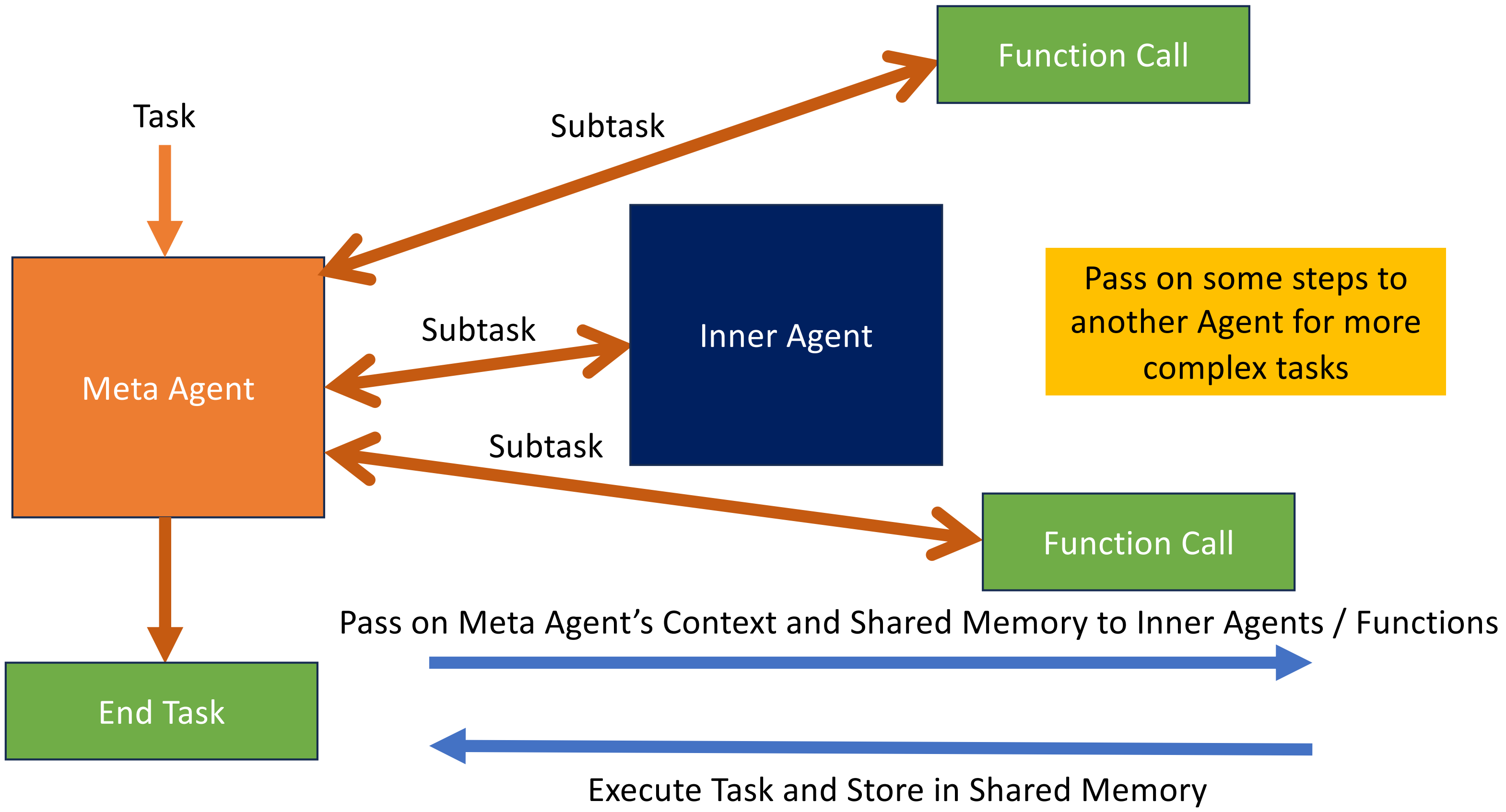
Key Question: How does an agent know what to do?



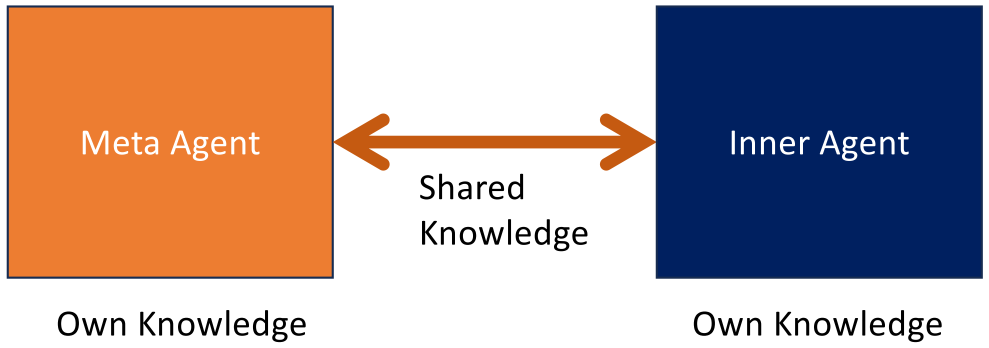
Key Question: How does an agent know what to do?



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


Task
Meta Agent
Subtask
Function Call
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

Key Question: How much to know?



Subtasks Completed

{“Economist(instruction = ‘Generate Prices for 5 Italian dishes’)”:

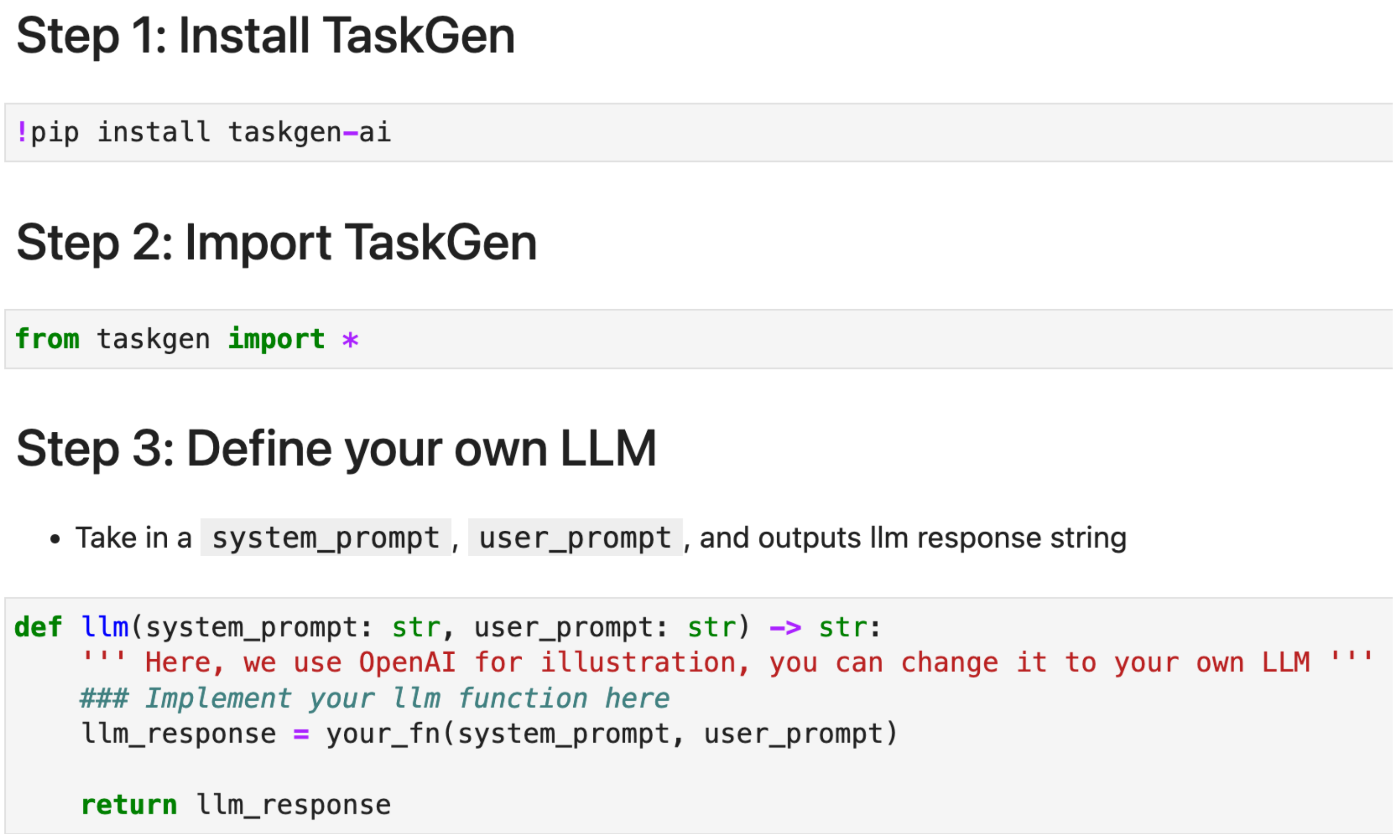
“I have come up with 5 dishes and their prices: pasta ($3), pizza ($4), truffle ($5), tiramisu ($8) and lasagna ($11)”}

Subtasks Completed

{“Think of 5 dishes”:
{”Agent Output”: “Pasta, pizza, truffle, tiramisu, lasagna”},

“dish_prices([‘Pasta’, ‘pizza’, ‘truffle’, ‘tiramisu’, lasagna’])”:
{‘output_1’: [”$3”, “$4”, ”$5”, “$8”, “$11”]} }

Setup in 3 easy steps



Step 1: Install TaskGen

```
!pip install taskgen-ai
```

Step 2: Import TaskGen

```
from taskgen import *
```

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 '''
    ### Implement your llm function here
    llm_response = your_fn(system_prompt, user_prompt)

    return llm_response
```

Sample Agent Usage

```python
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'
```

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

```python
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!
```

```python
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!
```

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

Experiments

- Agent Contributions
- Maze Navigator
- Escape Room Solving in TextWorld
- MATH Dataset
- RAG over NaturalQuestions Dataset

Agent Contributions

Hardik Maheshwari

Empowering the TaskGen Community

- Foster an open-source ecosystem for LLM agents
- Create a marketplace of powerful, reusable agents
- Enhance TaskGen's functionality through diverse contributions
- Build upon others' work to accelerate development
- Increase visibility and impact of your innovations

From Code to Contribution

- Contribute an Agent

```python
from taskgen import *

# Create your agent
my_agent = Agent('Helpful assistant', 'You are a generalist agent')

...

# Contribute your agent
os.environ['GITHUB_USERNAME'] = '<your GitHub username>'
os.environ['GITHUB_TOKEN'] = '<your GitHub token>'
my_agent.contribute_agent(author_comments = 'This is a generalist agent')
```

- Load a Community Agent

```python
from taskgen import *
agent = Agent.load_community_agent("Helpful Assistant")
```

The Magic Behind the Curtain

- Create/use TaskGen fork for user
- Generate Python representation of agent
- Commit to user's fork via GitHub APIs
- Initiate Pull Request to main TaskGen repository

Maze Navigator

John Tan Chong Min

Background

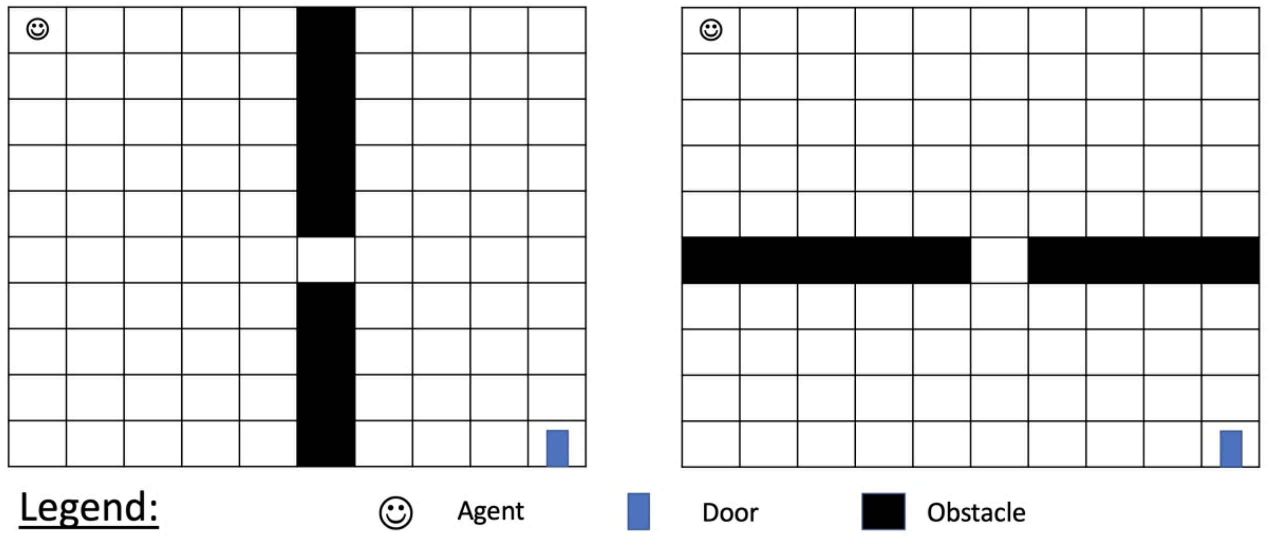


Figure D1: A sample maze environment of size 10x10. The actual experiment is 40x40. By default, the agent's start state is at the top left and the door is at the bottom right, but it can be varied. Obstacles change after half of the total number of episodes. **(Left)** Obstacles in the first half form a vertical wall with a gap in the centre across the mid-point. **(Right)** Obstacles in the second half from a horizontal wall with a gap in the centre across the mid-point.

Planner Overview

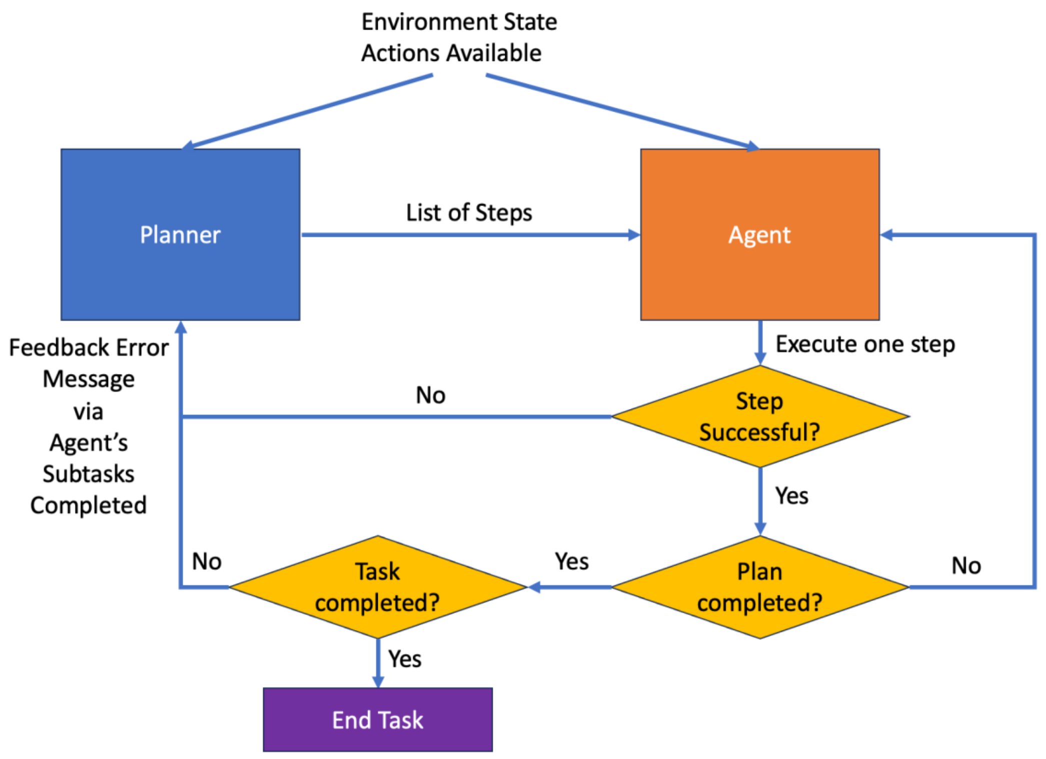


Figure D2: Schema of TaskGen Planner Interface with TaskGen Agent

Results: TaskGen Agent is close to optimal

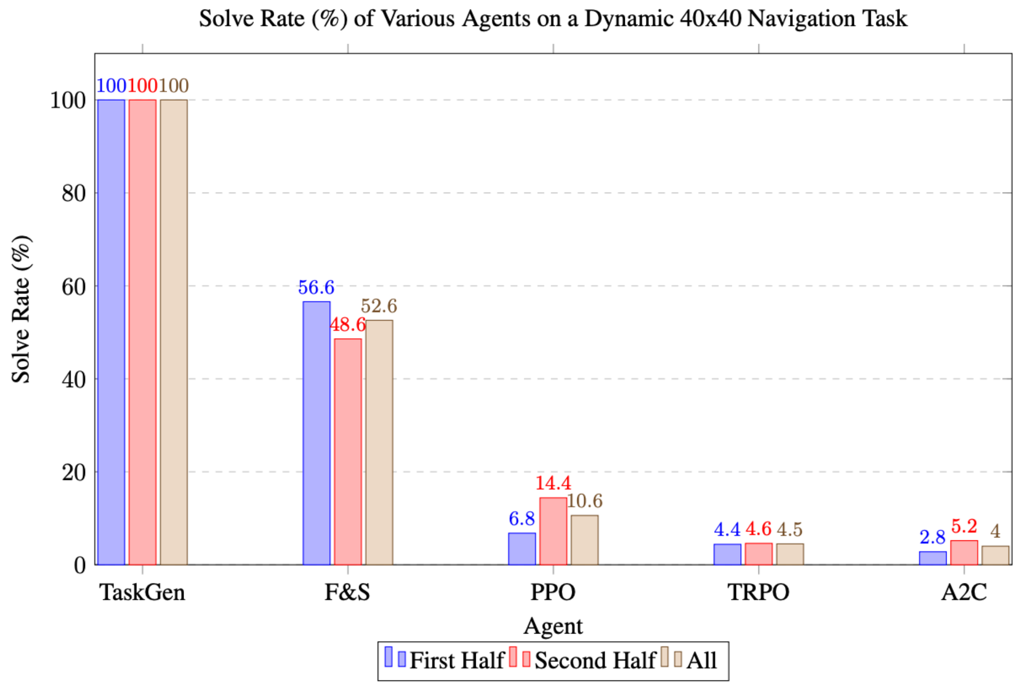


Figure D3: Solve rate (%) of various agents on a dynamic 40x40 navigation task for the first half, second half of episodes after obstacle positions change, and across all episodes.

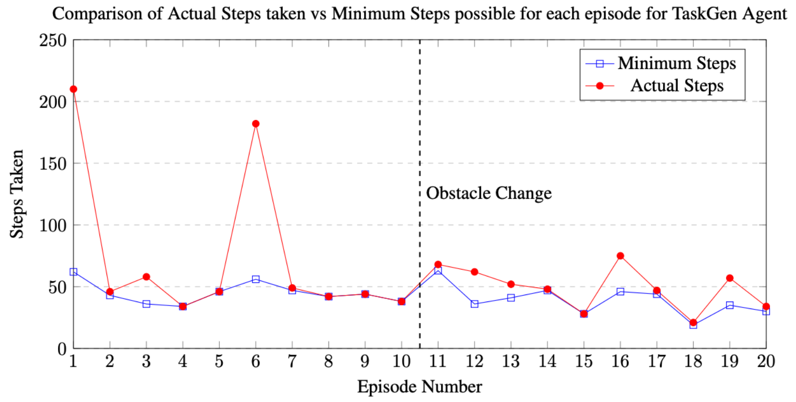
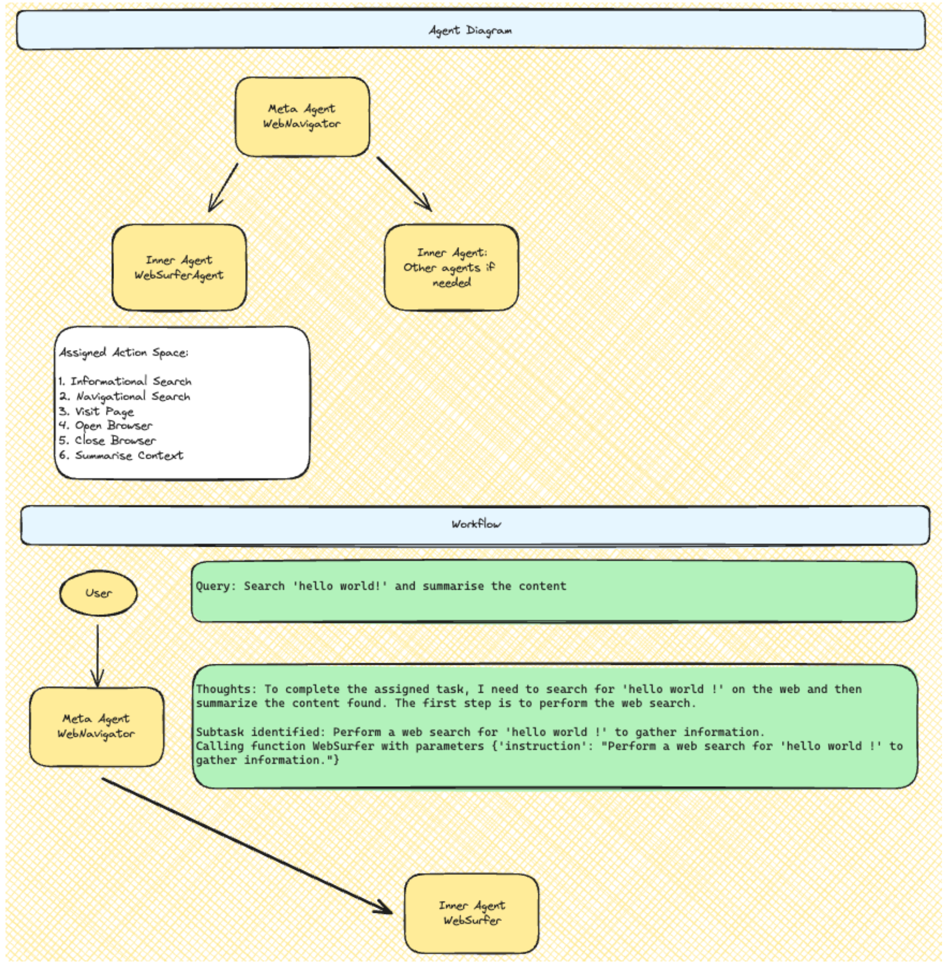


Figure D4: Comparison of Actual Steps taken vs Minimum Steps possible for each episode for TaskGen Agent. Note the obstacle positions are changed after episode 10.

Web Browsing Agent

Brian Lim

How does it work



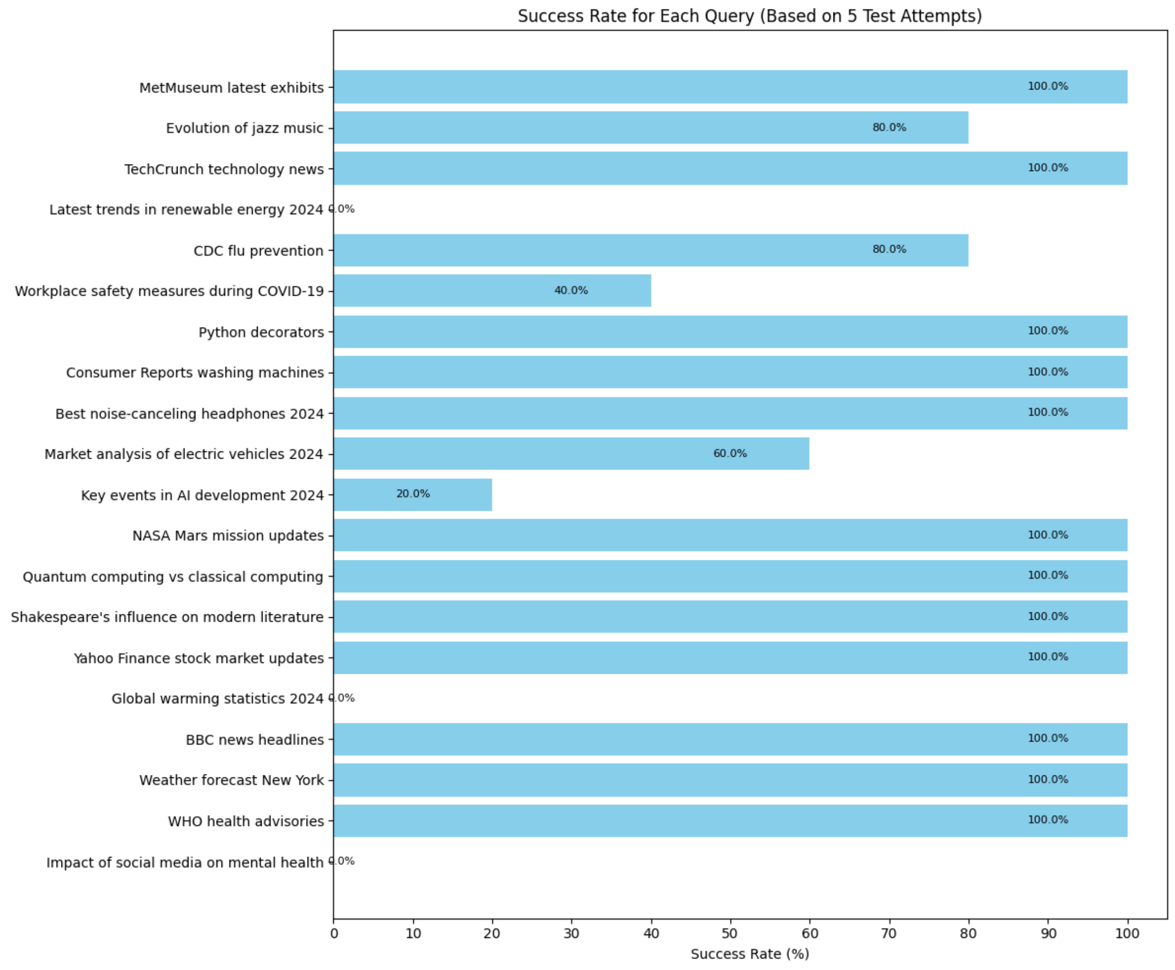
```
Query: Search 'hello world!' and summarise the content
```

```
Thoughts: To complete the assigned task, I need to search for 'hello world !' on the web and then
summarize the content found. The first step is to perform the web search.

Subtask identified: Perform a web search for 'hello world !' to gather information.
Calling function WebSurfer with parameters {'instruction': "Perform a web search for 'hello world !' to
gather information."}
```

Link to code implementation:

https://github.com/simbianai/taskgen/blob/main/Paper/web_browsing_agent/conversational_web_browsing_agent.py

Tests



Success Rate for Each Query (Based on 5 Test Attempts)
MetMuseum latest exhibits
100.0%
Evolution of jazz music
80.0%
TechCrunch technology news
100.0%
Latest trends in renewable energy 2024
0.0%
CDC flu prevention
80.0%
Workplace safety measures during COVID-19
40.0%
Python decorators
100.0%
Consumer Reports washing machines
100.0%
Best noise-canceling headphones 2024
100.0%
Market analysis of electric vehicles 2024
60.0%
Key events in AI development 2024
20.0%
NASA Mars mission updates
100.0%
Quantum computing vs classical computing
100.0%
Shakespeare's influence on modern literature
100.0%
Yahoo Finance stock market updates
100.0%
Global warming statistics 2024
0.0%
BBC news headlines
100.0%
Weather forecast New York
100.0%
WHO health advisories
100.0%
Impact of social media on mental health
0.0%
0
10
20
30
40
50
60
70
80
90
100
Success Rate (%)

Demo !!!

Escape Room Solving in TextWorld

Richard Cottrill

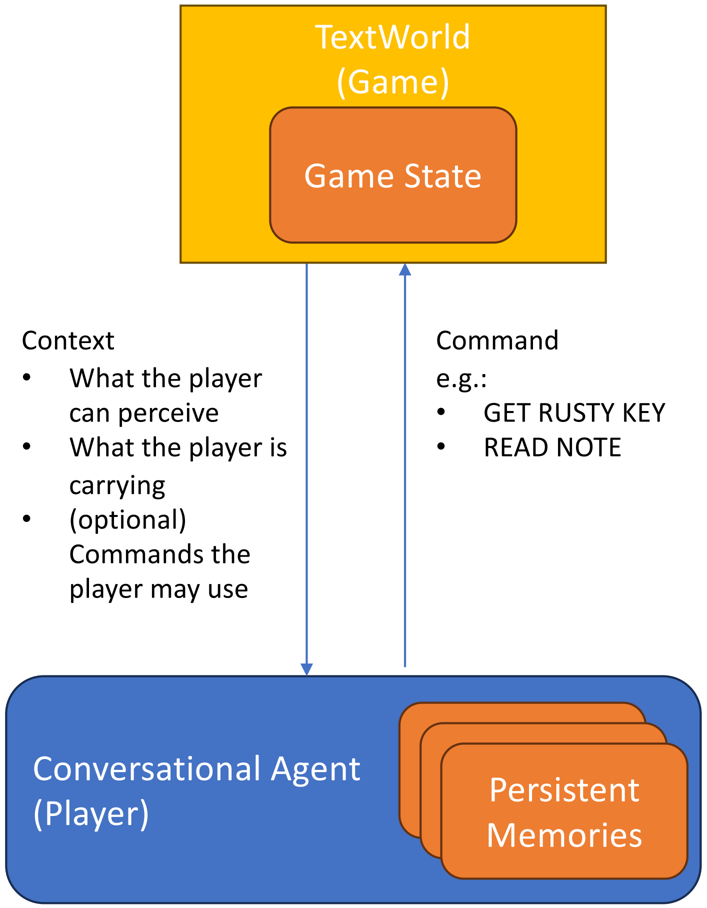
Journey into TextWorld

Step into a Text-Based Adventure

- Imagine navigating a mysterious world using only your wits and text commands.
- TextWorld is a framework for building randomly generated worlds/texts and engaging agents to play them.
- The archetypal IF work is Zork (1977-82) but the genre continues today.

Powering Up with TaskGen

- TaskGen equips agents to handle these adventures with strategic memory use and command execution.
- From simple navigation to complex problem-solving, TaskGen agents find their way through these game worlds.



TextWorld (Game)
Game State
Context
What the player can perceive
What the player is carrying
(optional) Commands the player may use
Command
e.g.:
GET RUSTY KEY
READ NOTE
Conversational Agent (Player)
Persistent Memories

TaskGen: The Intelligent Player

TaskGen as Player

It interprets text descriptions, creates novel commands, remembers past actions, and strategically navigates the game world.

It's able to discover appropriate objectives without guidance.

Harnessing Persistent Memory

TaskGen uses Persistent Memory to remember game objectives, successful commands, objects and rooms discovered.

With the conversational context generated by TaskGen, the agent is able to overcome dead-ends in its problem solving.

Did it work?

The Challenge

- TaskGen was put to the test in various TextWorld environments, each with unique goals and obstacles.

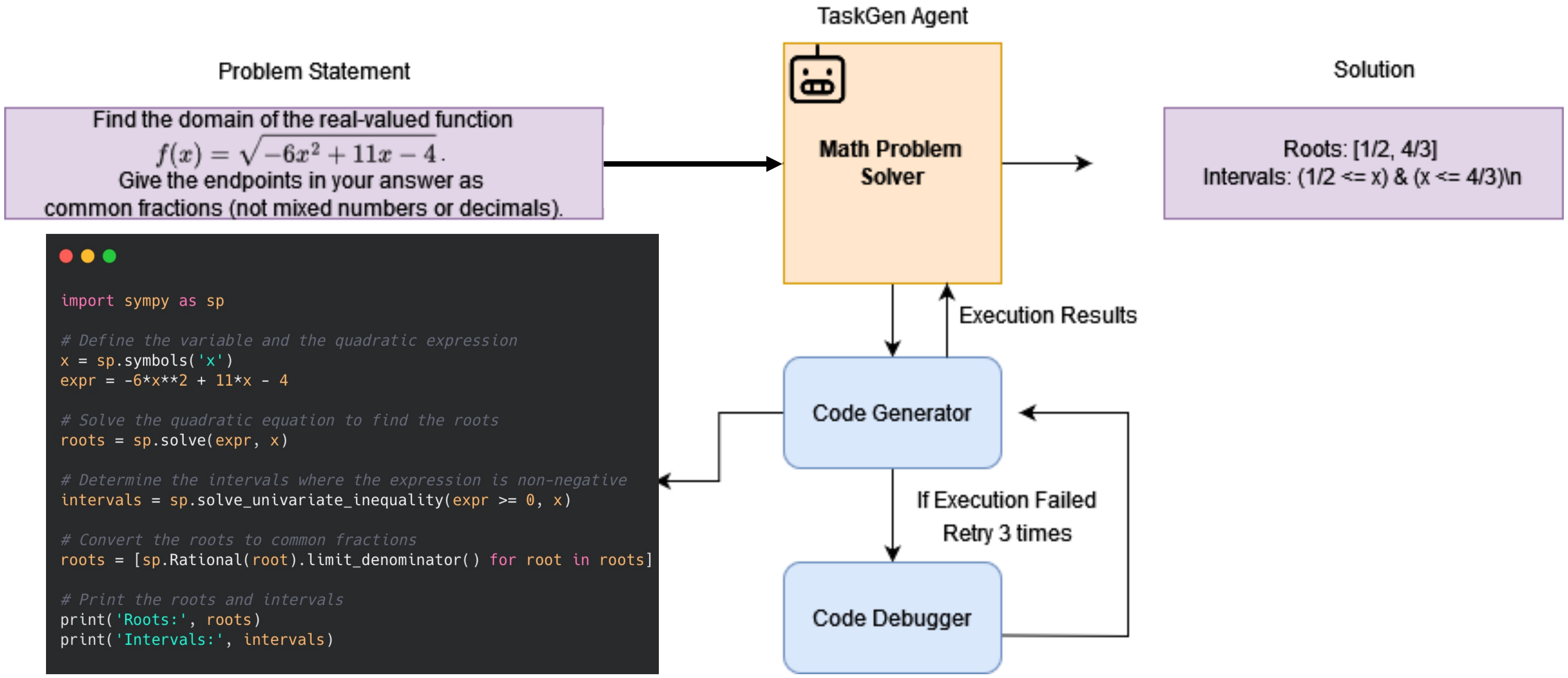
Stellar Performance

- Without prior knowledge of the problem or solution, a conversational agent was able to beat *the same LLM's* performance by 30%.
- Its ability to achieve high success rates in diverse scenarios showcases the potential of AI agents with loosely described goals.

MATH Dataset

Bharat Runwal

Math Problem Solver Agent

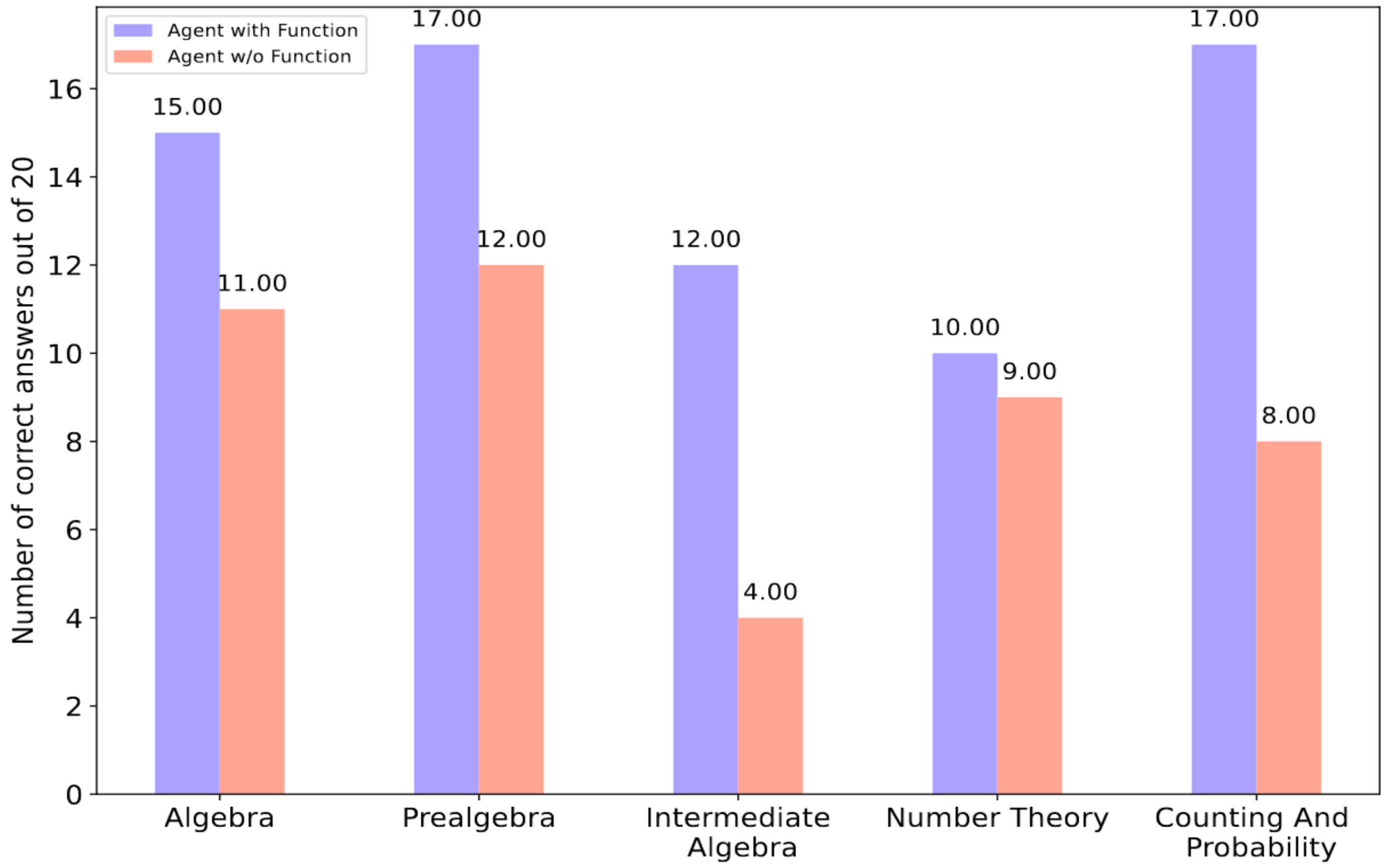


Find the domain of the real-valued function
$$f(x)=\sqrt{-6x^2+11x-4}.$$
Give the endpoints in your answer as common fractions (not mixed numbers or decimals).

```python
import sympy as sp

# Define the variable and the quadratic expression
x = sp.symbols('x')
expr = -6*x**2 + 11*x - 4

# Solve the quadratic equation to find the roots
roots = sp.solve(expr, x)

# Determine the intervals where the expression is non-negative
intervals = sp.solve_univariate_inequality(expr >= 0, x)

# Convert the roots to common fractions
roots = [sp.Rational(root).limit_denominator() for root in roots]

# Print the roots and intervals
print('Roots:', roots)
print('Intervals:', intervals)
```

```
Roots: [1/2, 4/3]
Intervals: (1/2 <= x) & (x <= 4/3)\n
```

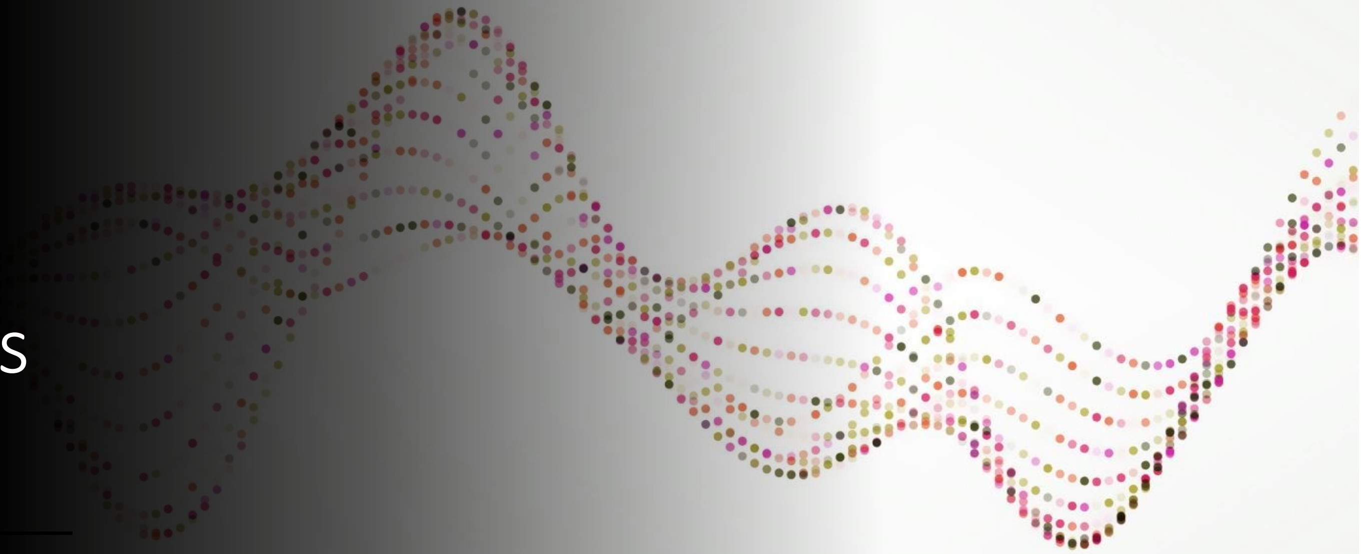
Experiment : MATH Dataset



- Used MATH dataset (Hendrycks et al., 2021), We focused specifically on the most challenging subset, **Level 5**, We randomly selected 20 problems from the test set of each category, resulting in a total of 100 problems in our test set.

- These results demonstrate that, in order to solve the challenging Level-5 problems, equipping the agent with code generation and debugging capabilities leads to more accurate solutions of mathematical problem.

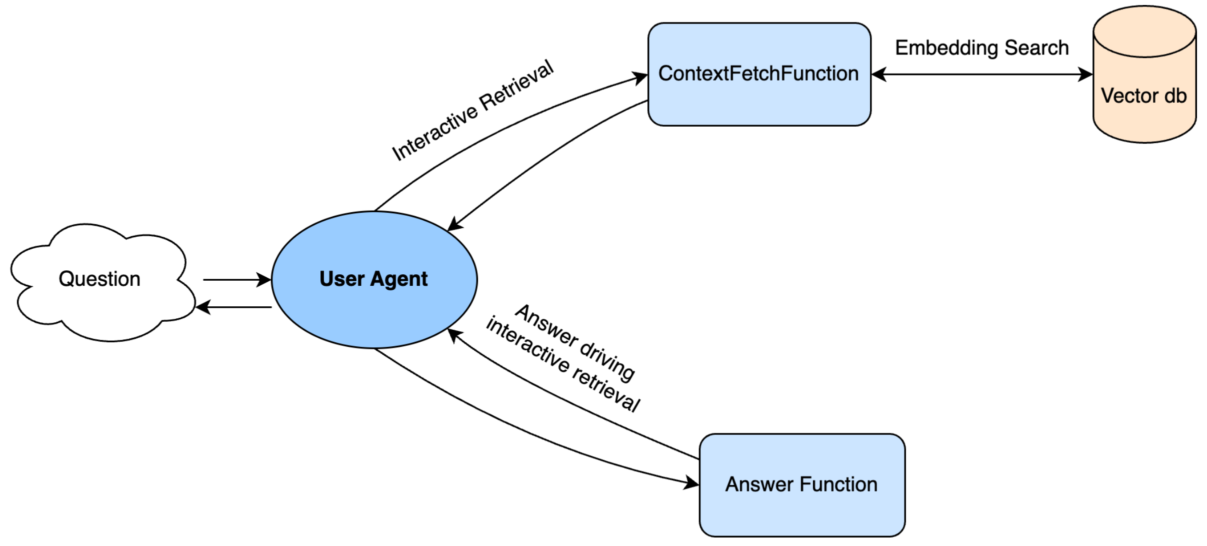
RAG over NaturalQuestions Dataset



Prince Saroj

Interactive Retrieval via TaskGen Functions and Agent

- **ContextFetchFunction**: Retrieves context for a query, fetching more if needed.
- **AnswerFunction**: Generates an answer, returning "no answer" if context is insufficient.
- **User Agent**: Manages context retrieval and answer generation until a satisfactory answer or limit.



Detailed Process

Query Submission:

- The user submits a query to the User Agent.

Context Retrieval:

- The User Agent invokes **ContextFetchFunction** with the initial query and a starting batch number (0).

Answer Generation:

- With the context obtained, the User Agent next activates **AnswerFunction**.
- If the context sufficiently answers the query, a response is generated.
- Otherwise, it issues "no answer."

Incremental Fetching:

- If "no answer" is received, the User Agent increments the batch number and re-engages **ContextFetchFunction** to obtain more context.
- This iterative process is capped at five interactions (max interactive retrieval count).

Results on Natural Question Dataset

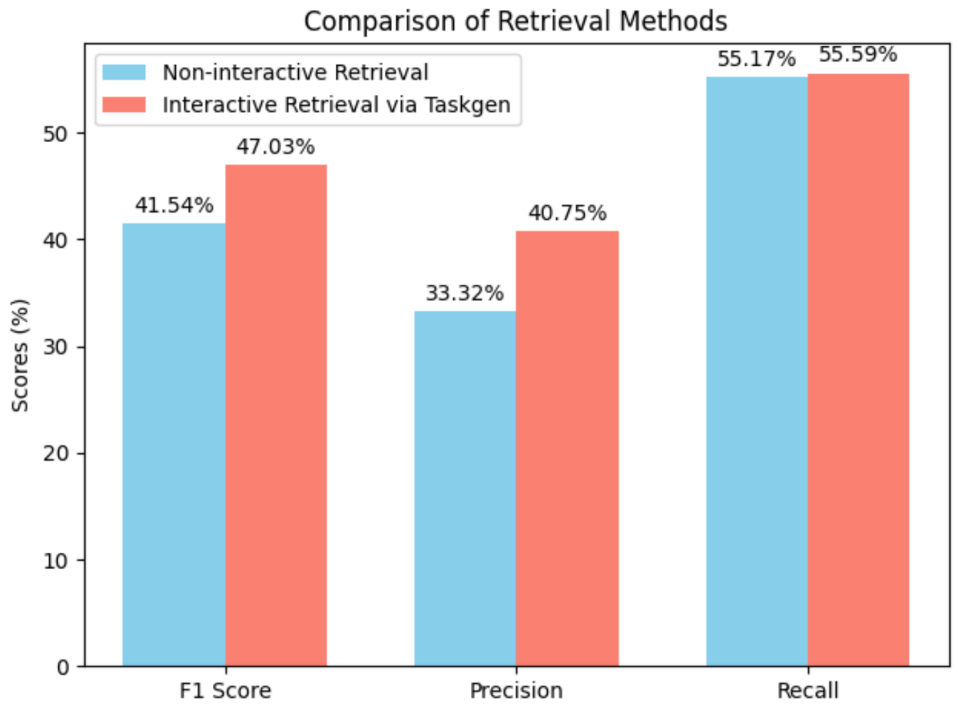
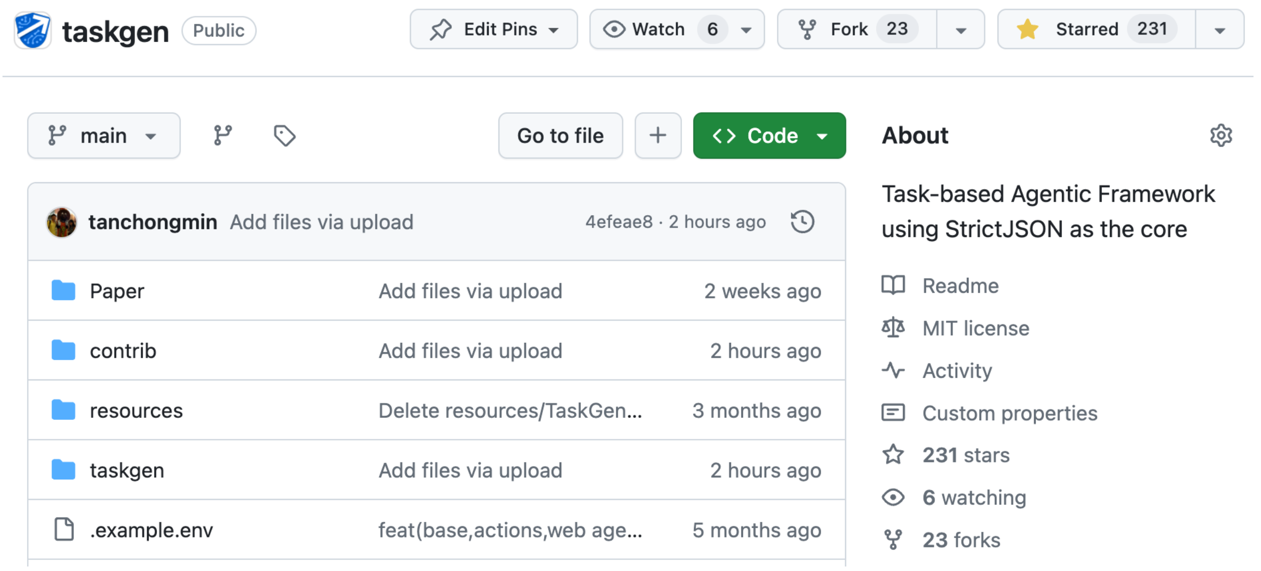


Figure H2: Graphical representation of the benchmark results comparing F1 Score, Precision, and Recall for Non-Interactive versus Interactive Retrieval via TaskGen (for both k=10 used for retrieval).

Discussion / How you can help

- TaskGen is meant to help us move towards better agentic structures
- Free to use, even commercially
- Help to like and star the GitHub
- Help to contribute example Agents / Jupyter Notebooks for more boilerplate code for others!



tanchongmin Add files via upload 4efeae8 · 2 hours ago

Paper	Add files via upload	2 weeks ago
contrib	Add files via upload	2 hours ago
resources	Delete resources/TaskGen...	3 months ago
taskgen	Add files via upload	2 hours ago
.example.env	feat(base,actions,web age...	5 months ago

About

Task-based Agentic Framework using StrictJSON as the core

- Readme
- MIT license
- Activity
- Custom properties
- 231 stars
- 6 watching
- 23 forks

https://github.com/simbianai/taskgen