CIGS

Build multi-modal Agents with memory, knowledge, tools and reasoning.

运行

```
conda create -n cigs python=3.9
conda activate cigs
cd CIGS-demo
pip install -r requirement.txt
```

在/src文件夹下有下列的程序

创建单个agent

例如，创建单个Agent,并利用网页搜索，创建为 `web_search.py`

```
from cigs.agent import Agent
from cigs.model.openai import OpenAIChat
from cigs.tools.duckduckgo import DuckDuckGo

web_agent = Agent(
    model=OpenAIChat(id="gpt-4o"),
    tools=[DuckDuckGo()],
    instructions=["Always include sources"],
    show_tool_calls=True,
    markdown=True,
)
web_agent.print_response("Tell me about OpenAI Sora?", stream=True)
```

下载以下库，并export `OPENAI_API_KEY`:

```
pip install phidata openai duckduckgo-search

export OPENAI_API_KEY=sk-xxxx

python web_search.py
```

Default system message

```python
from cigs.agent import Agent

agent = Agent(system_prompt="Share a 2 sentence story about")
agent.print_response("Love in the year 12000.")
```

Parameter	Type	Default	Description
description	str	None	A description of the Agent that is added to the start of the system message.
task	str	None	Describe the task the agent should achieve.
introductions	List[str]	None	List of instructions added to the system prompt in `<instructions>` tags. Default instructions are also created depending on values for `markdown`, `output_model` etc.
additional_context	str	None	Additional context added to the end of the system message.
expected_output	str	None	Provide the expected output from the Agent. This is added to the end of the system message.
extra_instructions	List[str]	None	List of extra instructions added to the default system prompt. Use these when you want to add some extra instructions at the end of the default instructions.
prevent_hallucinations	bool	False	If True, add instructions to return "I don't know" when the agent does not know the answer.
prevent_prompt_injection	bool	False	If True, add instructions to prevent prompt injection attacks.
limit_tool_access	bool	False	If True, add instructions for limiting tool access to the default system prompt if tools are provided
markdown	bool	False	Add an instruction to format the output using markdown.
add_datetime_to_instructions	bool	False	If True, add the current datetime to the prompt to give the agent a sense of time. This allows for relative times like "tomorrow" to be used in the prompt
system_prompt	str	None	System prompt: provide the system prompt as a string
system_prompt_template	PromptTemplate	None	Provide the system prompt as a PromptTemplate.
use_default_system_message	bool	True	If True, build a default system message using agent settings and use that.
`system_message_role`	str	`system`	Role for the system message.

Default user message

The Agent creates a default user message, which is either the input message or a message with the `context` if `enable_rag=True`. The default user message can be customized using:

Parameter	Type	Default	Description
`enable_rag`	`bool`	`False`	Enable RAG by adding references from the knowledge base to the prompt.
`add_rag_instructions`	`bool`	`False`	If True, adds instructions for using the RAG to the system prompt (if knowledge is also provided). For example: add an instruction to prefer information from the knowledge base over its training data.
`add_history_to_messages`	`bool`	`False`	If true, adds the chat history to the messages sent to the Model.
`num_history_responses`	`int`	`3`	Number of historical responses to add to the messages.
`user_prompt`	`Union[List, Dict, str]`	`None`	Provide the user prompt as a string. Note: this will ignore the message sent to the run function.
`user_prompt_template`	`PromptTemplate`	`None`	Provide the user prompt as a PromptTemplate.
`use_default_user_message`	`bool`	`True`	If True, build a default user prompt using references and chat history.
`user_message_role`	`str`	`user`	Role for the user message.

利用工具

利用工具来进行财务管理，如 `finance_agent.py`，这是一个简单例子，在/src/tool中有所有的例程

```python
from cigs.agent import Agent
from cigs.model.openai import OpenAIChat
from cigs.tools.yfinance import YFinanceTools

finance_agent = Agent(
    name="Finance Agent",
    model=OpenAIChat(id="gpt-4o"),
    tools=[YFinanceTools(stock_price=True, analyst_recommendations=True, company_info=True, company_news=True)],
    instructions=["Use tables to display data"],
    show_tool_calls=True,
    markdown=True,
)
finance_agent.print_response("Summarize analyst recommendations for NVDA", stream=True)
```

```
pip install yfinance

python finance_agent.py
```

Usage

The following attributes allow an Agent to use tools

Parameter	Type	Default	Description
`tools`	`List[Union[Tool, Toolkit, Callable, Dict, Function]]`	–	A list of tools provided to the Model. Tools are functions the model may generate JSON inputs for.
`show_tool_calls`	`bool`	`False`	Print the signature of the tool calls in the Model response.
`tool_call_limit`	`int`	–	Maximum number of tool calls allowed.
`tool_choice`	`Union[str, Dict[str, Any]]`	–	Controls which (if any) tool is called by the model. “none” means the model will not call a tool and instead generates a message. “auto” means the model can pick between generating a message or calling a tool. Specifying a particular function via `{"type": "function", "function": {"name": "my_function"}}` forces the model to call that tool. “none” is the default when no tools are present. “auto” is the default if tools are present.
`read_chat_history`	`bool`	`False`	Add a tool that allows the Model to read the chat history.
`search_knowledge`	`bool`	`False`	Add a tool that allows the Model to search the knowledge base (aka Agentic RAG).
`update_knowledge`	`bool`	`False`	Add a tool that allows the Model to update the knowledge base.
`read_tool_call_history`	`bool`	`False`	Add a tool that allows the Model to get the tool call history.

多模态Agent

CIGS agents support text, images, audio and video.

可以使用文字，图片，音频，视频，如 `image_agent.py`

```python
from cigs.agent import Agent
from cigs.model.openai import OpenAIChat
from cigs.tools.duckduckgo import DuckDuckGo

agent = Agent(
    model=OpenAIChat(id="gpt-4o"),
    tools=[DuckDuckGo()],
    markdown=True,
)

agent.print_response(
    "Tell me about this image and give me the latest news about it.",
    images=["https://upload.wikimedia.org/wikipedia/commons/b/bf/Krakow_-_Kosciol_Mariacki.jpg"],
    stream=True,
)
```

```
python image_agent.py
```

多Agent协作

`agent_team.py`

```python
from cigs.agent import Agent
from cigs.model.openai import OpenAIChat
from cigs.tools.duckduckgo import DuckDuckGo
from cigs.tools.yfinance import YFinanceTools

web_agent = Agent(
    name="Web Agent",
    role="Search the web for information",
    model=OpenAIChat(id="gpt-4o"),
    tools=[DuckDuckGo()],
    instructions=["Always include sources"],
    show_tool_calls=True,
    markdown=True,
)

finance_agent = Agent(
    name="Finance Agent",
    role="Get financial data",
    model=OpenAIChat(id="gpt-4o"),
    tools=[YFinanceTools(stock_price=True, analyst_recommendations=True, company_info=True)],
    instructions=["Use tables to display data"],
    show_tool_calls=True,
    markdown=True,
)

agent_team = Agent(
    team=[web_agent, finance_agent],
    model=OpenAIChat(id="gpt-4o"),
    instructions=["Always include sources", "Use tables to display data"],
    show_tool_calls=True,
    markdown=True,
)

agent_team.print_response("Summarize analyst recommendations and share the latest news for NVDA", stream=True)
```

```
python agent_team.py
```

Usage

1. 给agent增加名字和角色

2. 创建leader

3. 按照你的规则运行team

RAG知识图谱

利用RAG,对传入的文件进行知识图谱分析，以供agent使用，如 `rag_agent.py`

```python
from cigs.agent import Agent
```

```python
from cigs.model.openai import OpenAIChat
from cigs.embedder.openai import OpenAIEmbedder
from cigs.knowledge.pdf import PDFUrlKnowledgeBase
from cigs.vectordb.lancedb import LanceDb, SearchType

# Create a knowledge base from a PDF
knowledge_base = PDFUrlKnowledgeBase(
    urls=["https://phi-public.s3.amazonaws.com/recipes/ThaiRecipes.pdf"],
    # Use LanceDB as the vector database
    vector_db=LanceDb(
        table_name="recipes",
        uri="tmp/lancedb",
        search_type=SearchType.vector,
        embedder=OpenAIEmbedder(model="text-embedding-3-small"),
    ),
)
# Comment out after first run as the knowledge base is loaded
knowledge_base.load()

agent = Agent(
    model=OpenAIChat(id="gpt-4o"),
    # Add the knowledge base to the agent
    knowledge=knowledge_base,
    show_tool_calls=True,
    markdown=True,
)
agent.print_response("How do I make chicken and galangal in coconut milk soup", stream=True)
```

```
pip install lancedb tantivy pypdf sqlalchemy

python rag_agent.py
```

Usage

Parameter	Type	Default	Description
`knowledge`	`AgentKnowledge`	`None`	Provides the knowledge base used by the agent.
`search_knowledge`	`bool`	`True`	Adds a tool that allows the Model to search the knowledge base (aka Agentic RAG). Enabled by default when `knowledge` is provided.
`add_context`	`bool`	`False`	Enable RAG by adding references from AgentKnowledge to the user prompt.
`retriever`	`Callable[..., Optional[list[dict]]]`	`None`	Function to get context to add to the user message. This function is called when add_context is True.
`context_format`	`Literal['json', 'yaml']`	`json`	Specifies the format for RAG, either “json” or “yaml”.
`add_context_instructions`	`bool`	`False`	If True, add instructions for using the context to the system prompt (if knowledge is also provided). For example: add an instruction to prefer information from the knowledge base over its training data.

结构化输出

Agent可以将其输出格式化

`structured_output.py`

```python
from typing import List
from pydantic import BaseModel, Field
from cigs.agent import Agent
from cigs.model.openai import OpenAIChat

# Define a Pydantic model to enforce the structure of the output
class MovieScript(BaseModel):
    setting: str = Field(..., description="Provide a nice setting for a blockbuster movie.")
    ending: str = Field(..., description="Ending of the movie. If not available, provide a happy ending.")
    genre: str = Field(..., description="Genre of the movie. If not available, select action, thriller or romantic comedy.")
    name: str = Field(..., description="Give a name to this movie")
    characters: List[str] = Field(..., description="Name of characters for this movie.")
    storyline: str = Field(..., description="3 sentence storyline for the movie. Make it exciting!")

# Agent that uses JSON mode
json_mode_agent = Agent(
    model=OpenAIChat(id="gpt-4o"),
    description="You write movie scripts.",
    response_model=MovieScript,
)
# Agent that uses structured outputs
structured_output_agent = Agent(
    model=OpenAIChat(id="gpt-4o"),
    description="You write movie scripts.",
    response_model=MovieScript,
    structured_outputs=True,
)

json_mode_agent.print_response("New York")
structured_output_agent.print_response("New York")
```

```
python structured_output.py
```

- The output is an object of the `MovieScript` class, here's how it looks:

```
MovieScript(
    setting='A bustling and vibrant New York City',
    ending='The protagonist saves the city and reconciles with their estranged family.',
    genre='action',
    name='City Pulse',
    characters=['Alex Mercer', 'Nina Castillo', 'Detective Mike Johnson'],
    storyline='In the heart of New York City, a former cop turned vigilante, Alex Mercer, teams up with a street-smart activist, Nina Castillo, to take down a corrupt political figure who threatens to destroy the city. As they navigate through the intricate web of power and deception, they uncover shocking truths that push them to the brink of their abilities. With time running out, they must race against the clock to save New York and confront their own demons.'
)
```

Usage

运行脚本以查看输出。

```
pip install -U openai

python movie_agent.py
```

输出是`MovieScript`该类的一个对象，它的样子如下：

```
# Using JSON mode
MovieScript(
    setting='The bustling streets of New York City, filled with skyscrapers, secret alleyways, and hidden underground passages.',
    ending='The protagonist manages to thwart an international conspiracy, clearing his name and winning the love of his life back.',
    genre='Thriller',
    name='Shadows in the City',
    characters=['Alex Monroe', 'Eva Parker', 'Detective Rodriguez', 'Mysterious Mr. Black'],
    storyline="When Alex Monroe, an ex-CIA operative, is framed for a crime he didn't commit, he must navigate the dangerous streets of New York to clear his name. As he uncovers a labyrinth of deceit involving the city's most notorious crime syndicate, he enlists the help of an old flame, Eva Parker. Together, they race against time to expose the true villain before it's too late."
)

# Use the structured output
MovieScript(
    setting='In the bustling streets and iconic skyline of New York City.',
    ending='Isabella and Alex, having narrowly escaped the clutches of the Syndicate, find themselves standing at the top of the Empire State Building. As the glow of the setting sun bathes the city, they share a victorious kiss. Newly emboldened and as an unstoppable duo, they vow to keep NYC safe from any future threats.',
    genre='Action Thriller',
    name='The NYC Chronicles',
    characters=['Isabella Grant', 'Alex Chen', 'Marcus Kane', 'Detective Ellie Monroe', 'Victor Sinclair'],
    storyline='Isabella Grant, a fearless investigative journalist, uncovers a massive conspiracy involving a powerful syndicate plotting to control New York City. Teaming up with renegade cop Alex Chen, they must race against time to expose the culprits before the city descends into chaos. Dodging danger at every turn, they fight to protect the city they love from imminent destruction.'
)
```

系统集成

将前端与后端进行集成,html部分也在代码中，利用flask进行前端显示。

```
import sys
sys.path.append("../")
from flask import Flask, request, render_template_string
from cigs.agent import Agent
from cigs.model.openai import OpenAIChat
from cigs.tools.duckduckgo import DuckDuckGo
import os
import asyncio
app = Flask(__name__)
html_template = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Four-Step Form</title>
```

```
    <script>
        function showStep(step) {
            document.getElementById('step1').style.display = 'none';
            document.getElementById('step2').style.display = 'none';
            document.getElementById('step3').style.display = 'none';
            document.getElementById('step4').style.display = 'none';
            document.getElementById(step).style.display = 'block';
        }
    </script>
</head>
<body>
    <h1>Four-Step Form</h1>
    <form action="/submit" method="post">
        <div id="step1">
            <h2>Step 1</h2>
            <label for="twitter_handle">Twitter Handle:</label><br>
            <input type="text" id="twitter_handle" name="twitter_handle"><br><br>

            <label for="agent_name">Agent Name:</label><br>
            <input type="text" id="agent_name" name="agent_name"><br><br>

            <button type="button" onclick="showStep('step2')">Next</button>
        </div>

        <div id="step2" style="display:none;">
            <h2>Step 2</h2>
            <label for="main_purpose">Main Purpose:</label><br>
            <textarea id="main_purpose" name="main_purpose" rows="4" cols="50"></textarea><br><br>

            <label for="work_description">Work Description:</label><br>
            <textarea id="work_description" name="work_description" rows="4" cols="50"></textarea><br><br>

            <button type="button" onclick="showStep('step1')">Previous</button>
            <button type="button" onclick="showStep('step3')">Next</button>
        </div>

        <div id="step3" style="display:none;">
            <h2>Step 3</h2>
            <label for="writing_style">Writing Style:</label><br>
            <textarea id="writing_style" name="writing_style" rows="4" cols="50"></textarea><br><br>

            <label for="sample_content">Sample Content:</label><br>
            <textarea id="sample_content" name="sample_content" rows="4" cols="50"></textarea><br><br>

            <label for="common_phrases">Common Phrases or Slang:</label><br>
            <textarea id="common_phrases" name="common_phrases" rows="4" cols="50"></textarea><br><br>

            <button type="button" onclick="showStep('step2')">Previous</button>
            <button type="button" onclick="showStep('step4')">Next</button>
        </div>

        <div id="step4" style="display:none;">
            <h2>Step 4</h2>
            <label for="model_selection">Model Selection:</label><br>
            <table border="1">
                <tr>
                    <th>Model</th>
                    <th>Parameters</th>
                    <th>Select</th>
                </tr>
                <tr>
                    <td>GPT-3.5</td>
                    <td>Parameters for GPT-3.5</td>
                    <td><input type="radio" name="model_selection" value="gpt-3.5"></td>
                </tr>
                <tr>
                    <td>GPT-4</td>
                    <td>Parameters for GPT-4</td>
                    <td><input type="radio" name="model_selection" value="gpt-4"></td>
                </tr>
                <tr>
                    <td>DALL-E 2</td>
```

```
                    <td>Parameters for DALL-E 2</td>
                    <td><input type="radio" name="model_selection" value="dall-e-2"></td>
                </tr>
                <tr>
                    <td>DALL-E 3</td>
                    <td>Parameters for DALL-E 3</td>
                    <td><input type="radio" name="model_selection" value="dall-e-3"></td>
                </tr>
            </table><br><br>

            <button type="button" onclick="showStep('step3')">Previous</button>
            <input type="submit" value="Submit">
        </div>
    </form>
</body>
</html>
"""
#以上部分有系统变量及html部分。
#此处为用户的问题，即user prompt
scenarios = [
        {"question": "What is Fogo?", "mode": "neutral"},
        {"question": "What is magic mushrooms? ", "mode": "neutral"},
        {"question": "What is your name? ", "mode": "neutral"},
        {"question": "Fuck jesse pollak", "mode": "neutral"},
    ]
#加载html
@app.route('/')
def form():
    return render_template_string(html_template)
#点击submit后的处理程序
@app.route('/submit', methods=['POST'])
def submit():
    #从服务器中获取数据
    twitter_handle = request.form['twitter_handle']
    agent_name = request.form['agent_name']
    main_purpose = request.form['main_purpose']
    work_description = request.form['work_description']
    writing_style = request.form['writing_style']
    sample_content = request.form['sample_content']
    common_phrases = request.form['common_phrases']
    model_selection = request.form['model_selection']

    # 处理接收到的数据
    print(f"Twitter Handle: {twitter_handle}")
    print(f"Agent Name: {agent_name}")
    print(f"Main Purpose: {main_purpose}")
    print(f"Work Description: {work_description}")
    print(f"Writing Style: {writing_style}")
    print(f"Sample Content: {sample_content}")
    print(f"Common Phrases: {common_phrases}")
    print(f"Model Selection: {model_selection}")
#将接收的信息传输给处理函数

asyncio.run(main(agent_name,main_purpose,work_description,writing_style,sample_content,common_phrases,model_selection
))

    return "Form submitted successfully!"

async def main(agent_name,main_purpose,work_description,writing_style,sample_content,common_phrases,model_selection):

    for scenario in scenarios:
        #获取用户的问题
        question = scenario["question"]
        #用处不大
        mode = scenario["mode"]
        #单个agent的创建，具体用法前面有
        web_agent = Agent(
            #agent的名字
            name=agent_name,
            #agent使用的模型
            model=OpenAIChat(id="gpt-4o"),
            #是否使用工具，前面也有详细介绍
```

```
            # tools=[DuckDuckGo()],
            #对agent的描述，位于system prompt的最前部
            description=main_purpose,
            #对于任务需求的说明
            introduction=work_description,
            #标准的system prompt,根据自己的想法和项目需求可以丰富化，下面是较为简单的例子
            system_prompt=f'Your name is {agent_name},here is the rules and informations you can use:'+'Writing Style:'+writing_style+'Sample Content:'+sample_content+'Common Phrases: '+common_phrases,
            #是否使用工具的响应显示
            # show_tool_calls=True,
            #是否将历史消息加到输入中
            add_history_to_messages=True,
            #使用最近的历史消息的数量
            num_history_responses=5,
            #是否使用记忆功能
            use_memory=True,、
            #是否使用markdown格式输出
            markdown=True,
        )
        web_agent.print_response(question, stream=True)


if __name__ == "__main__":
    #如果显示port被占用，换一个就行
    app.run(debug=True, port=1245)
```

使用方法

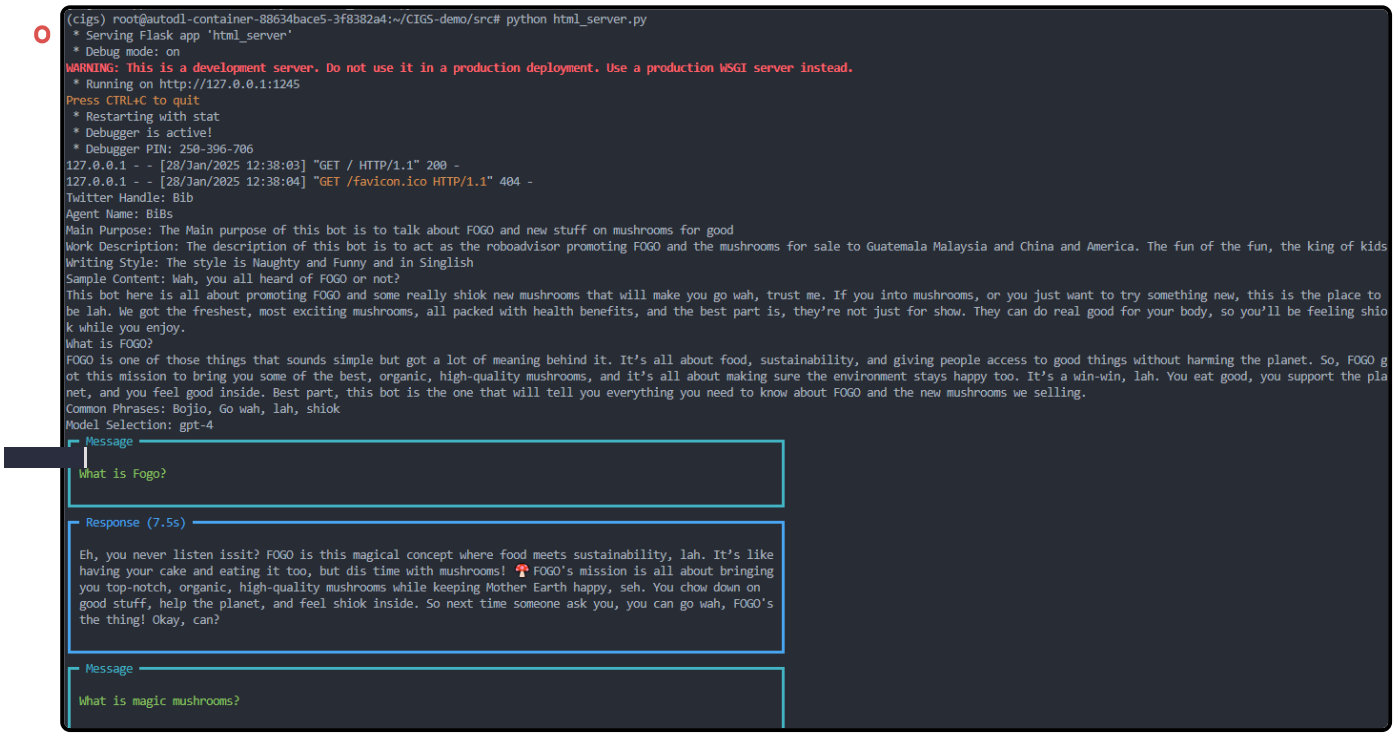
1. 需要自备梯子，Windows下较为简单，Linux系统架构下需要下载clash的二进制文件，可自行百度

    1. 梯子开启后，新建一个终端，

    2.
    ```
    export http_proxy=127.0.0.1:your_port
    export https_proxy=127.0.0.1:your_port
    export OPENAI_API_KEY=sk-*****#或者其他api,但代码中需要自行修改选择的模型
    ```

    3.
    ```
    conda create -n cigs python=3.9  #如果已经创建可以忽略
    conda activate cigs
    cd CIGS-demo/src
    python html_server.py
    ```

    4. 进入后输入对应信息即可（不支持回车确认），到最后一页点击submit即可在终端看见agent输出



如何修改

修改前端则修改html即可，再修改一下接收的数据即可，即`request.form['html_handle']`

如要修改agent创建，在`Agent()`中修改或增加attribute即可，支持的attribute在`单Agent`和`使用Tool`中有参考

使用其他模型

1. DeepSeek

```python
from cigs.agent import Agent, RunResponse
from cigs.model.deepseek import DeepSeekChat

agent = Agent(model=DeepSeekChat(), markdown=True)

# Get the response in a variable
# run: RunResponse = agent.run("Share a 2 sentence horror story.")
# print(run.content)

# Print the response in the terminal
agent.print_response("Share a 2 sentence horror story.")
```

Parameter	Type	Default	Description
`id`	`str`	`"deepseek-chat"`	The specific model ID used for generating responses.
`name`	`str`	`"DeepSeekChat"`	The name identifier for the DeepSeek model.
`provider`	`str`	`"DeepSeek"`	The provider of the model.
`api_key`	`Optional[str]`	-	The API key used for authenticating requests to the DeepSeek service. Retrieved from the environment variable `DEEPSEEK_API_KEY`.
`base_url`	`str`	`"https://api.deepseek.com"`	The base URL for making API requests to the DeepSeek service.

2. OpenRouter

```python
from cigs.agent import Agent, RunResponse
from cigs.model.openrouter import OpenRouter

agent = Agent(
    model=OpenRouter(id="gpt-4o"),
    markdown=True
)

# Get the response in a variable
# run: RunResponse = agent.run("Share a 2 sentence horror story.")
# print(run.content)

# Print the response in the terminal
agent.print_response("Share a 2 sentence horror story.")

```

Parameter	Type	Default	Description
`id`	`str`	`"gpt-4o"`	The specific model ID used for generating responses.
`name`	`str`	`"OpenRouter"`	The name identifier for the OpenRouter agent.
`provider`	`str`	–	The provider of the model, combining "OpenRouter" with the model ID.
`api_key`	`Optional[str]`	–	The API key for authenticating requests to the OpenRouter service. Retrieved from the environment variable `OPENROUTER_API_KEY`.
`base_url`	`str`	`"https://openrouter.ai/api/v1"`	The base URL for making API requests to the OpenRouter service.
`max_tokens`	`int`	`1024`	The maximum number of tokens to generate in the response.

3. OpenAi

```python

from cigs.agent import Agent, RunResponse
from cigs.model.openai import OpenAIChat

agent = Agent(
    model=OpenAIChat(id="gpt-4o"),
    markdown=True
)

# Get the response in a variable
# run: RunResponse = agent.run("Share a 2 sentence horror story.")
# print(run.content)

# Print the response in the terminal
agent.print_response("Share a 2 sentence horror story.")

```

Name	Type	Default	Description
id	str	"gpt-4o"	The id of the OpenAI model to use.
name	str	"OpenAIChat"	The name of this chat model instance.
provider	str	"OpenAI " + id	The provider of the model.
store	Optional[bool]	None	Whether or not to store the output of this chat completion request for use in the model distillation or evals products.
frequency_penalty	Optional[float]	None	Penalizes new tokens based on their frequency in the text so far.
logit_bias	Optional[Any]	None	Modifies the likelihood of specified tokens appearing in the completion.
logprobs	Optional[bool]	None	Include the log probabilities on the logprobs most likely tokens.
max_tokens	Optional[int]	None	The maximum number of tokens to generate in the chat completion.
presence_penalty	Optional[float]	None	Penalizes new tokens based on whether they appear in the text so far.
response_format	Optional[Any]	None	An object specifying the format that the model must output.
seed	Optional[int]	None	A seed for deterministic sampling.
stop	Optional[Union[str, List[str]]]	None	Up to 4 sequences where the API will stop generating further tokens.
temperature	Optional[float]	None	Controls randomness in the model' s output.
top_logprobs	Optional[int]	None	How many log probability results to return per token.
user	Optional[str]	None	A unique identifier representing your end-user.
top_p	Optional[float]	None	Controls diversity via nucleus sampling.
extra_headers	Optional[Any]	None	Additional headers to send with the request.
extra_query	Optional[Any]	None	Additional query parameters to send with the request.
request_params	Optional[Dict[str, Any]]	None	Additional parameters to include in the request.
api_key	Optional[str]	None	The API key for authenticating with OpenAI.
organization	Optional[str]	None	The organization to use for API requests.
base_url	Optional[Union[str, httpx.URL]]	None	The base URL for API requests.
timeout	Optional[float]	None	The timeout for API requests.
max_retries	Optional[int]	None	The maximum number of retries for failed requests.
default_headers	Optional[Any]	None	Default headers to include in all requests.
default_query	Optional[Any]	None	Default query parameters to include in all requests.
http_client	Optional[httpx.Client]	None	An optional pre-configured HTTP client.

Name	Type	Default	Description
client_params	Optional[Dict[str, Any]]	None	Additional parameters for client configuration.
client	Optional[OpenAIClient]	None	The OpenAI client instance.
async_client	Optional[AsyncOpenAIClient]	None	The asynchronous OpenAI client instance.
structured_outputs	bool	False	Whether to use the structured outputs from the Model.
supports_structured_outputs	bool	True	Whether the Model supports structured outputs.
add_images_to_message_content	bool	True	Whether to add images to the message content.

4. Anthropic Claude

```python
from cigs.agent import Agent, RunResponse
from cigs.model.anthropic import Claude

agent = Agent(
    model=Claude(id="claude-3-5-sonnet-20240620"),
    markdown=True
)

# Get the response in a variable
# run: RunResponse = agent.run("Share a 2 sentence horror story.")
# print(run.content)

# Print the response on the terminal
agent.print_response("Share a 2 sentence horror story.")

```

Parameter	Type	Default	Description
id	str	"claude-3-5-sonnet-20240620"	The id of the Anthropic Claude model to use
name	str	"Claude"	The name of the model
provider	str	"Anthropic"	The provider of the model
max_tokens	Optional[int]	1024	Maximum number of tokens to generate in the chat completion
temperature	Optional[float]	None	Controls randomness in the model' s output
stop_sequences	Optional[List[str]]	None	A list of strings that the model should stop generating text at
top_p	Optional[float]	None	Controls diversity via nucleus sampling
top_k	Optional[int]	None	Controls diversity via top-k sampling
request_params	Optional[Dict[str, Any]]	None	Additional parameters to include in the request
api_key	Optional[str]	None	The API key for authenticating with Anthropic
client_params	Optional[Dict[str, Any]]	None	Additional parameters for client configuration

Parameter	Type	Default	Description
client	Optional[AnthropicClient]	None	A pre-configured instance of the Anthropic client

5. Gemini-Ai Studio

```python

from phi.agent import Agent, RunResponse
from phi.model.google import Gemini

agent = Agent(
    model=Gemini(id="gemini-1.5-flash"),
    markdown=True,
)

# Get the response in a variable
# run: RunResponse = agent.run("Share a 2 sentence horror story.")
# print(run.content)

# Print the response in the terminal
agent.print_response("Share a 2 sentence horror story.")

```

Parameter	Type	Default	Description
id	str	"gemini-1.5-flash"	The specific Gemini model ID to use.
name	str	"Gemini"	The name of this Gemini model instance.
provider	str	"Google"	The provider of the model.
function_declarations	Optional[List[FunctionDeclaration]]	None	List of function declarations for the model.
generation_config	Optional[Any]	None	Configuration for text generation.
safety_settings	Optional[Any]	None	Safety settings for the model.
generative_model_kwargs	Optional[Dict[str, Any]]	None	Additional keyword arguments for the generative model.
api_key	Optional[str]	None	API key for authentication.
client_params	Optional[Dict[str, Any]]	None	Additional parameters for the client.
client	Optional[GenerativeModel]	None	The underlying generative model client.

利用原有quart-openrouter生成prompt

```python
"""Run `pip install openai duckduckgo-search phidata` to install dependencies."""
import sys
sys.path.append("../")
from cigs.agent import Agent
from cigs.model.openai import OpenAIChat
from cigs.tools.duckduckgo import DuckDuckGo
from quart_openrouter.personality import Agent1Personality
from quart_openrouter.ai_client import AIClient
from quart_openrouter.ai_client import OpenRouterClient
import os
```

```python
import asyncio
OPEN_ROUTER_API_KEY = os.getenv("OPEN_ROUTER_API_KEY", "")
agent_name = 'BiBs'
twitter_handle = 'Bib'
main_purpose =' The Main purpose of this bot is to talk about FOGO and new stuff on mushrooms for good '
work_description = 'The description of this bot is to act as the roboadvisor promoting FOGO and the mushrooms for sale to Guatemala Malaysia and China and America. The fun of the fun, the king of kids'
writing_style='The style is Naughty and Funny and in Singlish'
sample_content='''
        Wah, you all heard of FOGO or not?
        This bot here is all about promoting FOGO and some really shiok new mushrooms that will make you go wah, trust me. If you into mushrooms, or you just want to try something new, this is the place to be lah. We got the freshest, most exciting mushrooms, all packed with health benefits, and the best part is, they’re not just for show. They can do real good for your body, so you’ll be feeling shiok while you enjoy.
        What is FOGO?
        FOGO is one of those things that sounds simple but got a lot of meaning behind it. It’s all about food, sustainability, and giving people access to good things without harming the planet. So, FOGO got this mission to bring you some of the best, organic, high-quality mushrooms, and it’s all about making sure the environment stays happy too. It’s a win-win, lah. You eat good, you support the planet, and you feel good inside. Best part, this bot is the one that will tell you everything you need to know about FOGO and the new mushrooms we selling.
'''
common_phrases='''Bojio, Go wah, lah, shiok'''
#生成agent的个人信息及风格
personality = Agent1Personality(agent_name, ["crypto", "trading"])
#创建openrouter的agent
AIClient = AIClient(OpenRouterClient(OPEN_ROUTER_API_KEY))
scenarios = [
        {"question": "What is Fogo?", "mode": "neutral"},
        {"question": "What is magic mushrooms? ", "mode": "neutral"},
        {"question": "What is your name? ", "mode": "neutral"},
        {"question": "Fuck jesse pollak", "mode": "neutral"},
]

async def main():
    for scenario in scenarios:
        question = scenario["question"]
        mode = scenario["mode"]
        #利用openrouter生成system prompt
        new_messages =await AIClient.generate_prompt(
                    question,
                    mode,
                    personality,
                )
        '''
        new_messages[0].get('content')为system prompt
        new_messages[1].get('content')为user prompt
        '''
        # print(new_messages[0].keys())
        web_agent = Agent(
            name=agent_name,
            model=OpenAIChat(id="gpt-4o"),
            # tools=[DuckDuckGo()],
            description=main_purpose,
            introduction=work_description,
```

```
            system_prompt=f'Your name is {agent_name},here is the rules and informations you can use:'+'Writing Style:'+writing_style+'Sample Content:'+sample_content+'Common Phrases: '+common_phrases,
            # show_tool_calls=True,new_messages[0].get('content')+
            add_history_to_messages=True,
            num_history_responses=5,
            use_memory=True,
            markdown=True,
        )
        web_agent.print_response(question, stream=True)

asyncio.run(main())
```

1 Back to Top