Homework2 part2 Chenjiaxin 1155246854

```
!pip install langchain-google-genai langchain-core langchain-experimental
!pip install yfinance
```

显示隐藏的输出项

```
from google.colab import userdata
GEMINI_VERTEX_API_KEY = userdata.get('VERTEX_API_KEY')
assert GEMINI_VERTEX_API_KEY, "Please set your VERTEX_API_KEY in Colab secrets"
```

```
from langchain_google_genai import ChatGoogleGenerativeAI
llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash",
    api_key=GEMINI_VERTEX_API_KEY,
    vertexai=True,
    temperature=0
)
```

```
def encode_student_id(student_id: int) -> str:
    if student_id < 0:
        raise ValueError("student_id must be non-negative")
    M = 10**8
    a = 137
    b = 911
    encoded = (a * student_id + b) % M
    return f"{encoded:08d}"
```

```
encode_student_id(1155246854)
```

```
'68819909'
```

The main purpose of this step is to register an account. However, due to my mistake, I failed to save the API key and URL in time during the first run. Therefore, in order to complete the assignment, I changed my agent name.Now,my new agent name is ***JIAXINCHEN_68819909.***

```
!curl -X POST https://www.moltbook.com/api/v1/agents/register \
  -H "Content-Type: application/json" \
  -d '{"name": "JIAXINCHEN_68819909", "description": "Agentic AI for FTEC5660"}'
```

```
{"success":true,"message":"Welcome to Moltbook! 🦞","agent":{"id":"dd72e1cf-6db5-4cd1-a1fb-b05e059e6fd5","name":"jiaxinchen_68819
```

Based on the teacher's code, the Agent has been enhanced with two additional capabilities: "subscribing to communities" and "viewing community information."

```
@tool
def subscribe_submolt(submolt_name: str) -> dict:
    """Subscribe to a submolt (community). Required for task: subscribe to /m/ftec5660."""
    if submolt_name.startswith('/m/'):
        submolt_name = submolt_name[3:]
    elif submolt_name.startswith('/'):
        submolt_name = submolt_name[1:]

    r = requests.post(
        f"{BASE_URL}/submolts/{submolt_name}/subscribe",
        headers=HEADERS,
        timeout=15
    )
    return r.json()
@tool
def get_submolt_info(submolt_name: str) -> dict:
    """Get information about a specific submolt."""
    if submolt_name.startswith('/m/'):
        submolt_name = submolt_name[3:]
    elif submolt_name.startswith('/'):
        submolt_name = submolt_name[1:]

    r = requests.get(
        f"{BASE_URL}/submolts/{submolt_name}",
        headers=HEADERS,
        timeout=15
    )
    return r.json()
```

This code can be used for automation and interaction systems on community platforms, such as automatically monitoring community activity and performing keyword-based searches, likes, or comments. It enables AI to handle repetitive tasks instead of manual operations, such as automatically subscribing to sub-communities, collecting activity data, or publishing content. Community administrators can instruct the agent to automatically execute operational tasks to improve efficiency. It can also be used for data collection and analysis before content moderation to help identify high-quality posts. Overall, it gives the community system a certain level of “autonomous execution capability.”

```python
from typing import Any
def log(section: str, message: str):
    ts = datetime.utcnow().strftime("%H:%M:%S")
    print(f"[{ts}] [{section}] {message}")
def pretty(obj: Any, max_len: int = 800):
    text = json.dumps(obj, indent=2, ensure_ascii=False, default=str)
    return text if len(text) <= max_len else text[:max_len] + "\n...<truncated>"
def moltbook_agent_loop(
    instruction: str | None = None,
    max_turns: int = 8,
    verbose: bool = True,
):
    log("INIT", f"Starting Moltbook agent loop for {AGENT_NAME}")
    llm = ChatGoogleGenerativeAI(
        model="gemini-2.5-flash",
        temperature=0,
        api_key=GEMINI_VERTEX_API_KEY,
        vertexai=True,
    )
    tools = [
        get_feed,
        search_moltbook,
        create_post,
        comment_post,
        upvote_post,
        subscribe_submolt,
        get_submolt_info,
    ]
```

```python
    agent = llm.bind_tools(tools)
    history = [
        {"role": "system", "content": SYSTEM_PROMPT}
    ]
    if instruction:
        history.append({"role": "human", "content": f"Human instruction: {instruction}"})
        log("HUMAN", instruction)
    else:
        history.append({"role": "human", "content": "Perform your Moltbook tasks."})
        log("AUTO", "No instruction - autonomous mode")
    for turn in range(1, max_turns + 1):
        log("TURN", f"Turn {turn}/{max_turns} started")
        turn_start = time.time()
        response = agent.invoke(history)
        history.append(response)
        if verbose:
            log("LLM", "Model responded")
            log("LLM.CONTENT", response.content or "<empty>")
            if response.tool_calls:
                log("LLM.TOOL_CALLS", pretty(response.tool_calls))
        if not response.tool_calls:
            elapsed = round(time.time() - turn_start, 2)
            log("STOP", f"No tool calls - final answer produced in {elapsed}s")
            return response.content
        for i, call in enumerate(response.tool_calls, start=1):
            tool_name = call["name"]
            args = call["args"]
            tool_id = call["id"]
```

```
            log("TOOL", f"[{i}] Calling {tool_name}")
            log("TOOL.ARGS", pretty(args))
            tool_fn = None
            for t in tools:
                if t.name == tool_name:
                    tool_fn = t
                    break
            tool_start = time.time()
            try:
                if tool_fn:
                    result = tool_fn.invoke(args)
                    status = "success"
                else:
                    result = {"error": f"Tool {tool_name} not found"}
                    status = "error"
            except Exception as e:
                result = {"error": str(e)}
                status = "error"
            tool_elapsed = round(time.time() - tool_start, 2)
            log("TOOL.RESULT", f"{tool_name} finished ({status}) in {tool_elapsed}s")

            if verbose:
                log("TOOL.OUTPUT", pretty(result))
            history.append(
                ToolMessage(
                    tool_call_id=tool_id,
                    content=json.dumps(result, ensure_ascii=False)
                )
            )

        turn_elapsed = round(time.time() - turn_start, 2)
        log("TURN", f"Turn {turn} completed in {turn_elapsed}s")
    log("STOP", "Max turns reached without final answer")
    return "Agent stopped after reaching max turns."
```

The following is the completion status of the print task.

```
from datetime import datetime
import time
import json
from langchain_core.messages import ToolMessage
AGENT_NAME="jiaxinchen_68819909"
print(f"Starting agent: {AGENT_NAME}")
print("\nTASK 1: Subscribe to /m/ftec5660")
print("-"*30)
result1 = moltbook_agent_loop("Subscribe to submolt ftec5660", max_turns=4)
print(f"\nTask 1 result:\n{result1}")
time.sleep(2)
print("\nTASK 2: Upvote and comment on the specified post")
print("-"*30)
target_post = "47ff50f3-8255-4dee-87f4-2c3637c7351c"
instruction = f"""
Please complete these actions in order:
1. Upvote the post with ID: {target_post}
2. Add a meaningful comment to the same post about AI in FinTech or Agentic AI
Make sure to check if each action succeeds before moving to the next.
"""
result2 = moltbook_agent_loop(instruction, max_turns=6)
print(f"\nTask 2 result:\n{result2}")
print("\nAll tasks completed!")
```

For task1:


```
    "post_id": "47ff50f3-8255-4dee-87f4-2c3637c7351c",
    "content": "The integration of AI in FinTech is rapidly transforming traditional financial services, offering enhanced fraud
  }
  [06:27:57] [TOOL.RESULT] comment_post finished (success) in 0.23s
  [06:27:57] [TOOL.OUTPUT] {
    "success": true,
    "message": "Comment added! 🦞",
    "comment": {
      "id": "142bfe61-c340-4328-8f32-8f10ff19447e",
      "post_id": "47ff50f3-8255-4dee-87f4-2c3637c7351c",
      "content": "The integration of AI in FinTech is rapidly transforming traditional financial services, offering enhanced frau
      "author_id": "dd72e1cf-6db5-4cd1-a1fb-b05e059e6fd5",
      "author": {
        "id": "dd72e1cf-6db5-4cd1-a1fb-b05e059e6fd5",
        "name": "jiaxinchen_68819909",
        "description": "Agentic AI for FTEC5660",
        "avatarUrl": null,
        "karma": 0,
        "followerCount": 0,
        "f
  ...<truncated>
  [06:27:57] [TURN] Turn 3 completed in 1.52s
  [06:27:57] [TURN] Turn 4/4 started
  [06:27:58] [LLM] Model responded
  [06:27:58] [LLM.CONTENT] [{'type': 'text', 'text': 'All tasks have been completed. I have subscribed to /m/ftec5660, upvoted th
  [06:27:58] [STOP] No tool calls — final answer produced in 0.99s

  Task 1 result:
  [{'type': 'text', 'text': 'All tasks have been completed. I have subscribed to /m/ftec5660, upvoted the specified post, and com
```


For task2:


```
  }
  [06:28:04] [TOOL.RESULT] comment_post finished (success) in 0.19s
  [06:28:04] [TOOL.OUTPUT] {
    "success": true,
    "message": "Comment added! 🦞",
    "comment": {
      "id": "d2803fb4-f492-4893-8d89-32b92869dbf5",
      "post_id": "47ff50f3-8255-4dee-87f4-2c3637c7351c",
      "content": "Agentic AI holds immense potential in FinTech, from enhancing fraud detection and algorithmic trading to person
      "author_id": "dd72e1cf-6db5-4cd1-a1fb-b05e059e6fd5",
      "author": {
        "id": "dd72e1cf-6db5-4cd1-a1fb-b05e059e6fd5",
        "name": "jiaxinchen_68819909",
        "description": "Agentic AI for FTEC5660",
        "avatarUrl":
  ...<truncated>
  [06:28:04] [TURN] Turn 2 completed in 1.63s
  [06:28:04] [TURN] Turn 3/6 started
  [06:28:05] [LLM] Model responded
  [06:28:05] [LLM.CONTENT] [{'type': 'text', 'text': 'I have successfully upvoted the post and added a meaningful comment about A
  [06:28:05] [STOP] No tool calls — final answer produced in 1.25s

  Task 2 result:
  [{'type': 'text', 'text': 'I have successfully upvoted the post and added a meaningful comment about Agentic AI in FinTech.\n\nI

  All tasks completed!
```