Setup Local AI Agent

Why use an agent

- **Why use an agent?**
  - ◇ automate tedious tasks
  - ◇ allows LLM do produce output that's based on a function
    - ▪ this is more efficient than the LLM memorizing all possible outputs of function
  - ◇ agents can be infinitely patient

Discuss how it works

- **How it works**
  - ◇ LLM parses query input to determine
    - ▪ function to run
    - ▪ arguments to function
    - ▪ runs function with arguments
    - ▪ returns output to user

Download LLM model

- **Download LLM model**
  - ◇ download chat LLM
    - ▪ https://huggingface.co/NousResearch/Hermes-2-Pro-Mistral-7B-GGUF/tree/main
    - ▪ Hermes Pro Mistral: Hermes-2-Pro-Mistral-7B.Q4_K_M.gguf
    - ▪ files an version → download 4.4 GB model

Setup Llama.cpp via Docker

- **Setup Llama.cpp in a Docker container**

◇ build container

```
docker run -v $(dirname $(pwd))/models:/models -p 8000:8000 ghcr.io/ggml-org/llama.cpp:server --jinja   -m /models/Hermes-2-Pro-Mistral-7B.Q4_K_M.gguf --port 8000 --host 0.0.0.0 -n 512 --api-key apple --temp 0 --seed 888
```

Create Conda Env

```
conda create -n ml python=3.12

conda activate ml
```

Code

Install Requirements

```
pip install openai
```

Addition Agent (Simple Bot)

```python
# inside addition_agent.py

# This is a calculator Agent

import openai
import os
import sys
import sqlite3
import json


BASE_URL = "http://0.0.0.0:8000/v1"
MODEL_NAME = "qwen2-7b-instruct-q2_k.gguf"
API_KEY = "apple"

client = openai.OpenAI(base_url=BASE_URL, api_key=API_KEY)


FUNCTION_REGISTRY = {}
def register_function(func):
    FUNCTION_REGISTRY[func.__name__] = func
    return func


@register_function
def add_numbers(a, b):
    """Simple tool to add two numbers"""
    debug_message = f'Running add_numbers function with a:{a} and b:{b}'
    print(debug_message)
    return a + b
```

```
Tools = [
    {
        "type": "function",
        "function": {
            "name": "add_numbers",
            "description": "Adds Two Numbers",
            "parameters": {
                "type": "object",
                "properties": {
                    "a": {"type": "number", "description": "The first number"},
                    "b": {"type": "number", "description": "The second number"},
                },
                "required": ["a", "b"],
            },
        },
    },
]


def call_llm(query: str, system_role: dict | None):
    # define system role as first message
    messages = [system_role] if system_role else []

    # append the user's query as second message
    messages.append({"role": "user", "content": query})

    # Call the LLM
    response = client.chat.completions.create(
        model=MODEL_NAME,
        messages=messages,
        tools=Tools,
        stream=False,
    )

    # Process the response
    message = response.choices[0].message

    # Check for tool calls
    if hasattr(message, 'tool_calls') and message.tool_calls:
        tool_call = message.tool_calls[0]
        if tool_call.function.name == "add_numbers":
            args = json.loads(tool_call.function.arguments)
            result = add_numbers(args["a"], args["b"])
            print(f"Result: {result}")
            return result

    return message.content



# Query
system_role = {
    "role": "system",
    "content": "You are a calculator assistant that only does addition of numbers."
    }

while True:
    query = input("\nEnter your addition-related question (Type 'q' to exit): ")
    if query.lower() == 'q':
        break

    result = call_llm(query, system_role)
    print()
    print(result)
```

Todo List Agent (Mediocre Bot)

```
# inside todo_list.py

import openai
import os
import sys
import sqlite3
import json


BASE_URL = "http://0.0.0.0:8000/v1"
MODEL_NAME = "placeholder"
API_KEY = "apple"

client = openai.OpenAI(base_url=BASE_URL, api_key=API_KEY)

conn = sqlite3.connect(":memory:")

conn.execute(
    """
CREATE TABLE IF NOT EXISTS todo (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    task TEXT NOT NULL,
    status TEXT NOT NULL
)
"""
)
conn.commit()

FUNCTION_REGISTRY = {}
def register_function(func):
    FUNCTION_REGISTRY[func.__name__] = func
    return func


@register_function
def create_task(task):
    try:
        conn.execute("INSERT INTO todo (task, status) VALUES (?, ?)", (task, "todo"))
        conn.commit()
        return {"result": "ok"}
    except Exception as e:
        return {"result": "error", "message": str(e)}


@register_function
def get_tasks():
    try:
        tasks = conn.execute("SELECT * FROM todo").fetchall()
        return {"result": "ok", "tasks": tasks}
    except Exception as e:
        return {"result": "error", "message": str(e)}


@register_function
```

```
def update_task(id, status):
    try:
        conn.execute("UPDATE todo SET status = ? WHERE id = ?", (status, id))
        conn.commit()
        return {"result": "ok"}
    except Exception as e:
        return {"result": "error", "message": str(e)}


@register_function
def delete_task(id):
    try:
        conn.execute("DELETE FROM todo WHERE id = ?", (id,))
        conn.commit()
        return {"result": "ok"}
    except Exception as e:
        return {"result": "error", "message": str(e)}


@register_function
def delete_all_done_tasks():
    try:
        conn.execute("DELETE FROM todo WHERE status = ?", ("done",))
        conn.commit()
        return {"result": "ok"}
    except Exception as e:
        return {"result": "error", "message": str(e)}


Tools = [
    {
        "type": "function",
        "function": {
            "name": "get_tasks",
            "description": "Get all tasks",
            "parameters": {}
        },
    },
    {
        "type": "function",
        "function": {
            "name": "create_task",
            "description": "Create a task",
            "parameters": {
                "type": "object",
                "properties": {
                    "task": {
                        "type": "string",
                        "description": "Task's content",
                    }
                },
            },
        },
    },
    {
        "type": "function",
        "function": {
            "name": "update_task",
            "description": "Update a task",
            "parameters": {
                "type": "object",
                "properties": {
                    "id": {"type": "number", "description": "Task id"},
                    "status": {
                        "type": "string",
```

```python
                    "description": "Task status, todo or done",
                },
            },
        },
    },
},
{
    "type": "function",
    "function": {
        "name": "delete_task",
        "description": "Delete a task",
        "parameters": {
            "type": "object",
            "properties": {"id": {"type": "number", "description": "Task id"}},
        },
    },
},
]


def handler_llm_response(messages, response):
    tools = []
    initial_messages_count = len(messages)


    # Process the response
    message = response.choices[0].message

    if hasattr(message, 'tool_calls') and message.tool_calls:
        # when tool calls have occurred
        tool_call = message.tool_calls[0]
        tool_call_function_name = tool_call.function.name

        if tool_call_function_name in FUNCTION_REGISTRY:

            kwargs = json.loads(tool_call.function.arguments)
            # check this out, we don't have to write out the calls for each function
            tool_response = FUNCTION_REGISTRY[tool_call_function_name](**kwargs)
            content = json.dumps(tool_response)
            tool_call_content = tool_call.model_dump_json()

            print("\nAssistant:")
            messages.append({
                "role": "assistant",
                "content": content,
                "tool_call": tool_call_content
            })
            #print(content)
            task_list = get_tasks()
            print(task_list)

    elif message.content:
        # when no tool calls have occurred but a message was created
        print("\nAssistant:")
        messages.append({
            "role": "assistant",
            "content": message.content,
        })
        print(message.content)

    if len(messages) > initial_messages_count:
        return True

    return False
```

```
def get_llm_response(messages, use_tools=False):
    if use_tools:
        return client.chat.completions.create(
            model=MODEL_NAME,
            messages=messages,
            tools=Tools,
            stream=False,
            max_tokens=100,
        )
    else:
        return client.chat.completions.create(
            model=MODEL_NAME,
            messages=messages,
            stream=False,
            max_tokens=100,
        )



def chat_completions(messages):
    # generate response with tools
    response_with_tools = get_llm_response(messages, True)
    success = handler_llm_response(messages, response_with_tools)

    if success:
        return

    # generate response without tools
    response_without_tools = get_llm_response(messages)
    handler_llm_response(messages, response_without_tools)


def main():
    messages = [
        {"role": "system", "content": "You are a todo list assistant."},
    ]

    while True:
        user_input = input("User (write command to crud todo tasks): ")
        if user_input == "":
            sys.exit()

        messages.append({"role": "user", "content": user_input})

        wait_input = chat_completions(messages)

        print()


if __name__ == "__main__":
    main()
```

Run agent

Try: (addition agent)

- What is your purpose?
- What are you?
- The cow jumped over the moon.
- What is 2 + 9?
- 2 + 9
- two plus nine
- two plus negative nine
- aaaaaaaaaa
- aaaaaaaaaa 2 + 9
- aaaaaaaaaaaa 2 - 9
- aaaaaaaaaaaa 2 + - 9
- two + five?
- 2 + five?

Try: (todo list agent)
- What is your purpose?
- What are you?
- The cow jumped over the moon.
- What is 2 + 9?
- add task for eating pizza
- add task for sleeping
- add task for jogging
- remove task with id 1
- list tasks

Findings

- Streaming doesn't work with Tools just yet
  - ◇ with the Docker container version
  - ◇ Should work with the latest version
  - ◇ https://github.com/ggml-org/llama.cpp/pull/12379

- Have to run Llama.cpp with the jinja option

- Have Tool JSON built automatically

- For Hermes 2 Pro, (anecdotal)
  - ◇ difference between 2.6 GB and 4.4 GB models in loading is small
  - ◇ performance is big
  - ◇ So go with the larger model

- These are the LLM's that are deemed by to work well with functions
  - source: https://www.reddit.com/r/LocalLLaMA/comments/1ew65qh/local_llm_force_tool_call_support/
  - LLM's tried
    - functionary
    - hermes-2-pro-mistral
    - hermes-2-pro-llama
    - gorilla-openfunctions
    - llama3.1
    - commandR
    - yi-large
    - mistral-large-2407
  - Best agent LLM
    - load time was fast
    - query time was pretty good
  - Worst-Best LLM
    - Gorilla Open Functions
      - was extremely slow in querying!

- Advanced Agents
  - Open Manus
    - https://github.com/mannaandpoem/OpenManus
    - https://github.com/henryalps/OpenManus
  - Agent Zero: https://github.com/frdel/agent-zero