Individual Project Report

Course: FTEC5660
Student ID: 1155254873
Name: CHEN, Peng
Reproduced Project: nanobot
Github: https://github.com/cpppp/SEEM5660/tree/main/Reproducibility-Work

1. Project summary

1.1 Summary

NanoBot is an ultra-lightweight personal AI assistant framework developed by the Data Science Laboratory (HKUDS) at the University of Hong Kong. With only about 4,000 lines of core code, it is 99% more concise than similar projects like Clawdbot, which has 430,000 lines of code, yet delivers full AI assistant functionality.

Chart1 nanobot's logo

1.2 Core features

Ultra-lightweight: nearly 4,000 lines of core code, easy to understand and expand.

Multi-platform support: Telegram, Discord, WhatsApp, Lark, DingTalk, Slack, Email, QQ, etc.

Multi-model support: OpenRouter,Anthropic,OpenAI,DeepSeek,Gemini, Moonshot/Kimi, and over 15 other providers.

MCP support: Compatible with Model Context Protocol, enabling integration with external tool servers.

Skill system: Scalable skill mechanism with customizable capabilities.

Memory system: Dual-layer architecture combining long-term memory and historical records.

1.3 Architecture

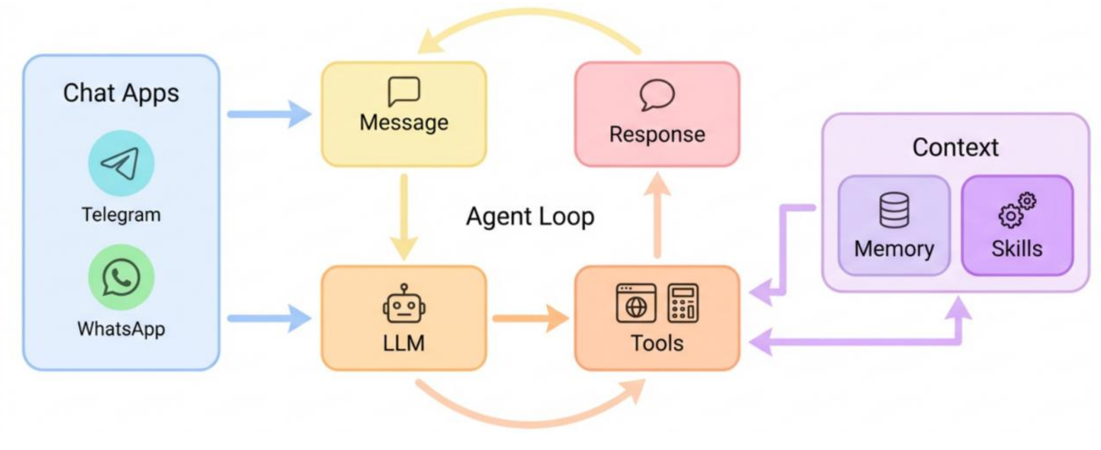


Chart2 nanobot's architecture

Nanobot's architecture enables intelligent agent interactions via chat apps (e.g., Telegram). Users send messages → LLM interprets intent → invokes tools (e.g., web search) → processes results → generates context-aware responses. A dynamic Context layer (memory for history +

skills for capabilities) continuously updates with each interaction, enabling coherent multi-turn dialogue. The Agent Loop (Response→Message) creates a closed system: responses trigger new actions, while Context refines future decisions. Efficiently combines LLM reasoning, tool execution, and adaptive context management for seamless, task-oriented user experiences.

2. Set up

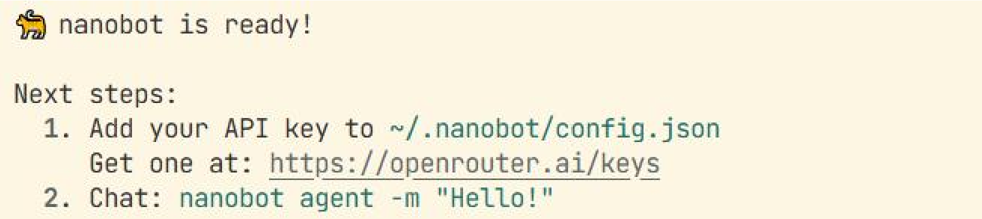
Clone the project from github. Input the commands to install:

```
git clone https://github.com/HKUDS/nanobot.git
cd nanobot
pip install -e .
```

After installing the necessary dependency, input:

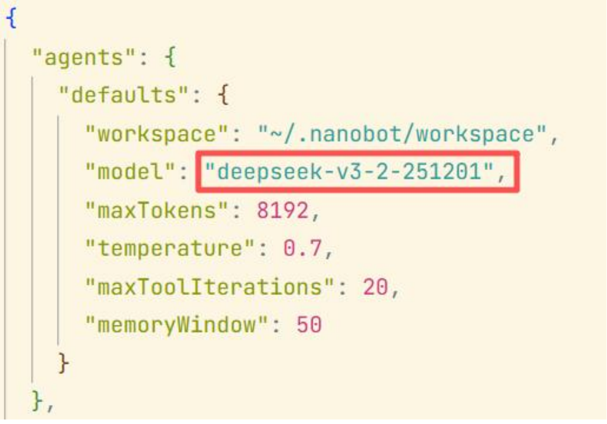
```
nanobot onboard
```

We will see “nanobot is ready”.

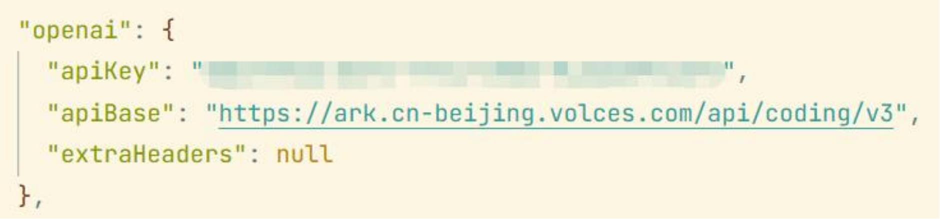


```
🐈 nanobot is ready!

Next steps:
  1. Add your API key to ~/.nanobot/config.json
     Get one at: https://openrouter.ai/keys
  2. Chat: nanobot agent -m "Hello!"
```

We can follow the guide, set up the model and api key in the config.json.



```
{
  "agents": {
    "defaults": {
      "workspace": "~/.nanobot/workspace",
      "model": "deepseek-v3-2-251201",
      "maxTokens": 8192,
      "temperature": 0.7,
      "maxToolIterations": 20,
      "memoryWindow": 50
    }
  },
```

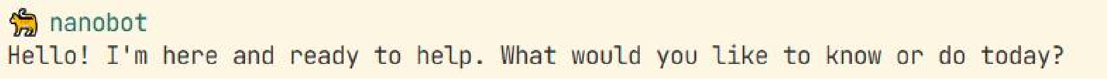


```
"openai": {
  "apiKey": "[redacted]",
  "apiBase": "https://ark.cn-beijing.volces.com/api/coding/v3",
  "extraHeaders": null
},
```

Here I use the Volcanic engine’s api key, and it can connect to llm via openai interface. Then we can use command to verify the connection of AI model.

```
nanobot agent -m "Hello!"
```

We can see the reply of nanobot in the command line.



```
🐈 nanobot
Hello! I'm here and ready to help. What would you like to know or do today?
```

Here I complete the preliminary set up.

3. Reproduction

3.1 Target

Nanobot can connect to social chat platform such as Telegram, Discord, WhatsApp, Feishu, Mochat, DingTalk, Slack, Email, QQ.

Here I will set up the configuration, connect to [**Discord**] and try to talk to nanobot via it.

And then we can test the functions about:

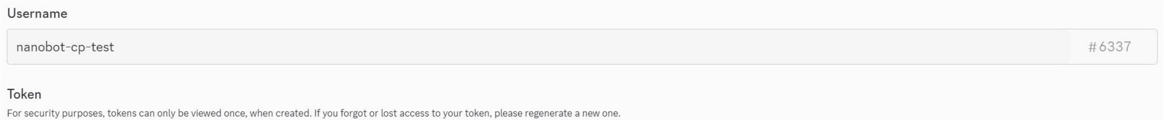
- 24/7 Real-time Market Analysis (see 3.3.1)
- Full-stack Software Engineer (see 3.3.2)
- Smart Daily Routine Manager (see 3.3.3)
- Personal Knowledge Assistant (see 3.3.4)

3.2 Process to connect Discord

Follow the guidelines, connect the nanobot to Discord.

(1) Create a bot.

- Go to [https://discord.com/developers/applications]
- Create an application → Bot → Add Bot
- Copy the bot token (Mine is a token with "MTQ3M……")



Username

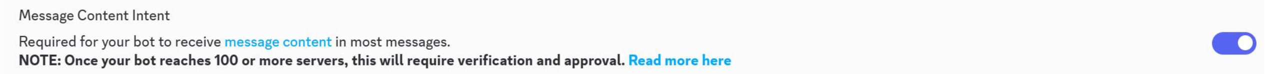
nanobot-cp-test #6337

Token

For security purposes, tokens can only be viewed once, when created. If you forgot or lost access to your token, please regenerate a new one.

(2) Enable intents

- In the Bot settings, enable **MESSAGE CONTENT INTENT**
- (Optional) Enable SERVER MEMBERS INTENT if you plan to use allow lists based on member data



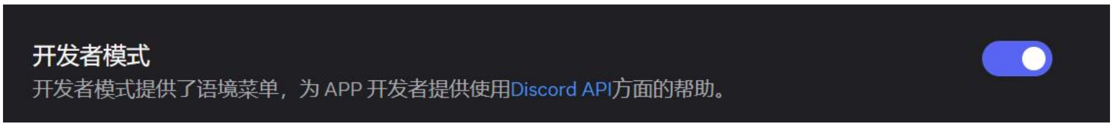
Message Content Intent

Required for your bot to receive message content in most messages.

NOTE: Once your bot reaches 100 or more servers, this will require verification and approval. Read more here

(3) Get User ID

- Discord Settings → Advanced → enable Developer Mode
- Right-click your avatar → Copy User ID



开发者模式

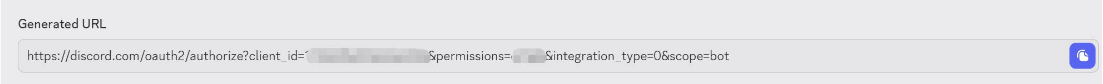
开发者模式提供了语境菜单，为 APP 开发者提供使用Discord API方面的帮助。

(4) Configure

```
{
  "channels": {
    "discord": {
      "enabled": true,
      "token": "YOUR_BOT_TOKEN",
      "allowFrom": ["YOUR_USER_ID"]
    }
  }
}
```

(5) Invite the bot

- OAuth2 → URL Generator
- Scopes: bot
- Bot Permissions: Send Messages, Read Message History
- Open the generated invite URL and add the bot to your server



Generated URL

https://discord.com/oauth2/authorize?client_id=[redacted]&permissions=[redacted]&integration_type=0&scope=bot

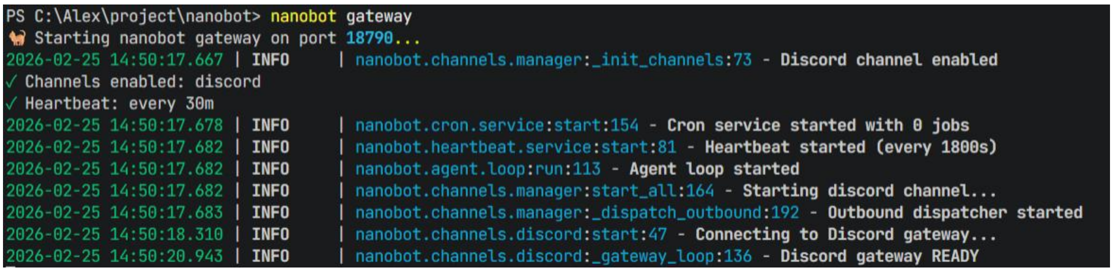
Here I create my own server named “nanobot-test-server” and add the nanobot in it.

(6) Run

```
nanobot gateway
```

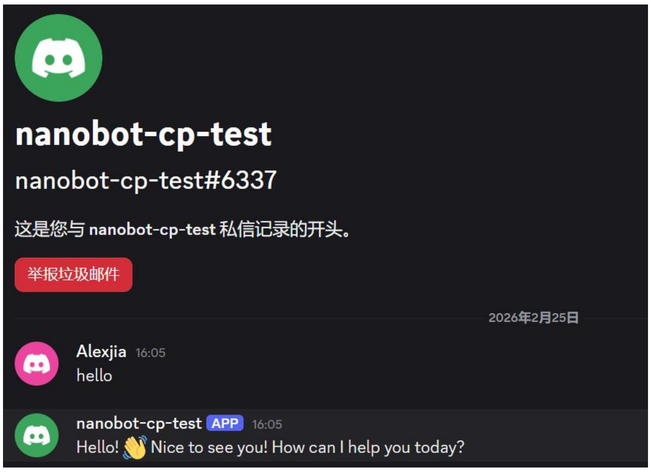
3.3 Result - market/code/routine/knowledge

After run the “nanobot gateway”, here is the picture of command line shows.



```
PS C:\Alex\project\nanobot> nanobot gateway
🐈 Starting nanobot gateway on port 18790...
2026-02-25 14:50:17.667 | INFO     | nanobot.channels.manager:_init_channels:73 - Discord channel enabled
✓ Channels enabled: discord
✓ Heartbeat: every 30m
2026-02-25 14:50:17.678 | INFO     | nanobot.cron.service:start:154 - Cron service started with 0 jobs
2026-02-25 14:50:17.682 | INFO     | nanobot.heartbeat.service:start:81 - Heartbeat started (every 1800s)
2026-02-25 14:50:17.682 | INFO     | nanobot.agent.loop:run:113 - Agent loop started
2026-02-25 14:50:17.682 | INFO     | nanobot.channels.manager:start_all:164 - Starting discord channel...
2026-02-25 14:50:17.683 | INFO     | nanobot.channels.manager:_dispatch_outbound:192 - Outbound dispatcher started
2026-02-25 14:50:18.310 | INFO     | nanobot.channels.discord:start:47 - Connecting to Discord gateway...
2026-02-25 14:50:20.943 | INFO     | nanobot.channels.discord:_gateway_loop:136 - Discord gateway READY
```

Open the discord we can see a bot (which I named it as “nanobot”) in my created server. Now I can chat to it about normal question/ weather / code etc.



nanobot-cp-test

nanobot-cp-test#6337

这是您与 **nanobot-cp-test** 私信记录的开头。

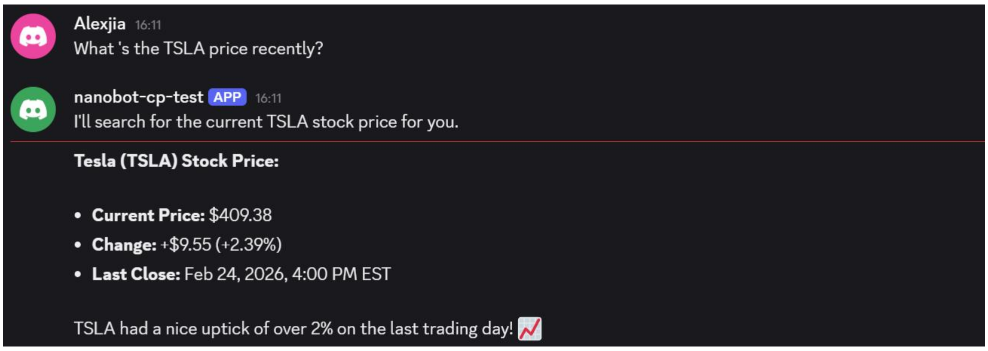
2026年2月25日

Alexjia 16:05
hello

nanobot-cp-test APP 16:05
Hello! 👋 Nice to see you! How can I help you today?

3.3.1 Market analysis result

Here I ask about the TSLA’s stock price in Discord.



Alexjia 16:11
What 's the TSLA price recently?

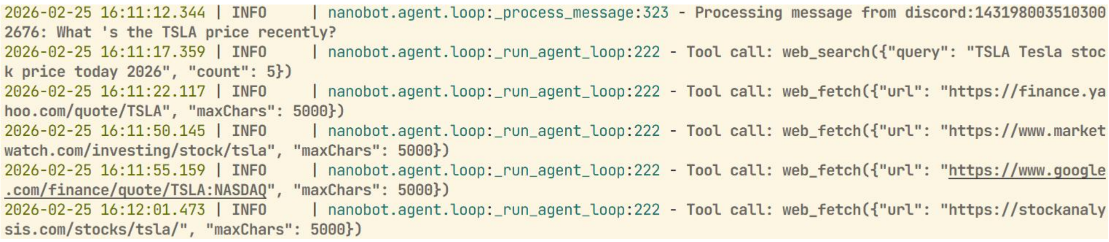
nanobot-cp-test APP 16:11
I'll search for the current TSLA stock price for you.

Tesla (TSLA) Stock Price:

- **Current Price:** $409.38
- **Change:** +$9.55 (+2.39%)
- **Last Close:** Feb 24, 2026, 4:00 PM EST

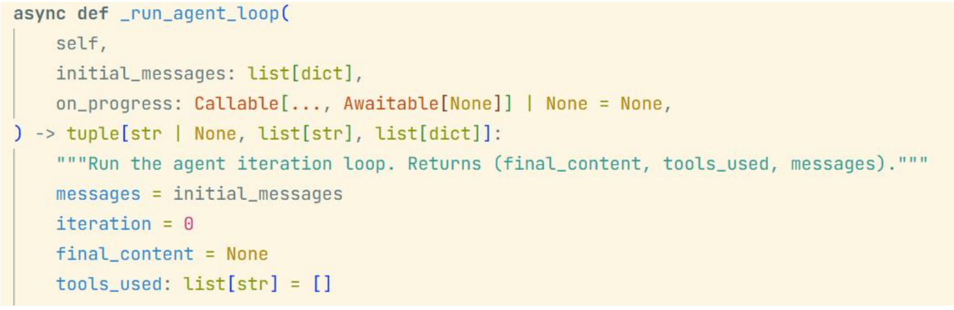
TSLA had a nice uptick of over 2% on the last trading day! 📈

And we can see the backend, nanobot calls several tools to find the answer.



```
2026-02-25 16:11:12.344 | INFO     | nanobot.agent.loop:_process_message:323 - Processing message from discord:143198003510300
2676: What 's the TSLA price recently?
2026-02-25 16:11:17.359 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_search({"query": "TSLA Tesla stoc
k price today 2026", "count": 5})
2026-02-25 16:11:22.117 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_fetch({"url": "https://finance.ya
hoo.com/quote/TSLA", "maxChars": 5000})
2026-02-25 16:11:50.145 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_fetch({"url": "https://www.market
watch.com/investing/stock/tsla", "maxChars": 5000})
2026-02-25 16:11:55.159 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_fetch({"url": "https://www.google
.com/finance/quote/TSLA:NASDAQ", "maxChars": 5000})
2026-02-25 16:12:01.473 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_fetch({"url": "https://stockanaly
sis.com/stocks/tsla/", "maxChars": 5000})
```

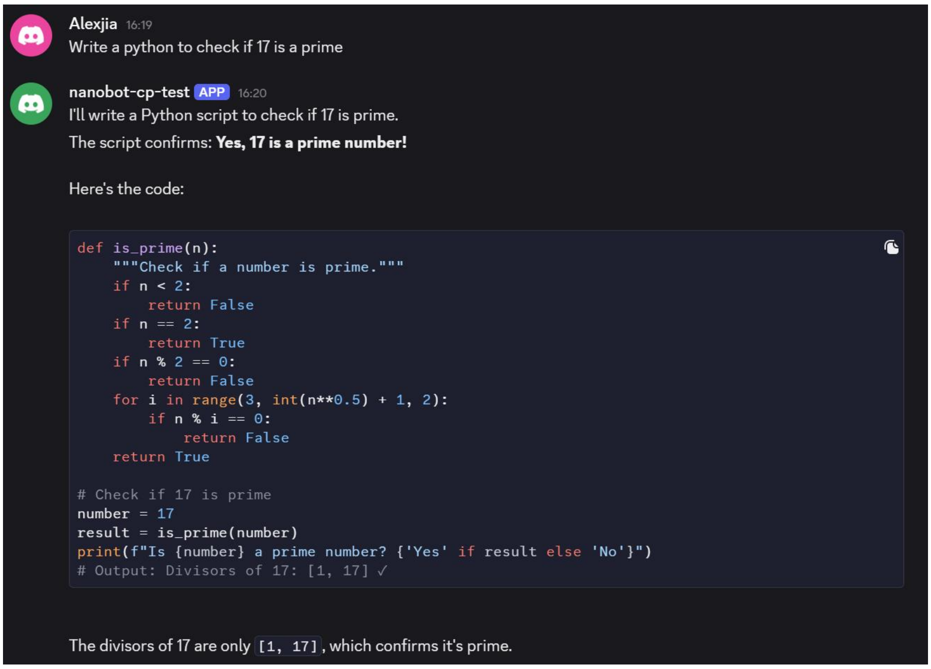
We can also find the part of code about “_run_agent_loop"



```python
async def _run_agent_loop(
    self,
    initial_messages: list[dict],
    on_progress: Callable[..., Awaitable[None]] | None = None,
) -> tuple[str | None, list[str], list[dict]]:
    """Run the agent iteration loop. Returns (final_content, tools_used, messages)."""
    messages = initial_messages
    iteration = 0
    final_content = None
    tools_used: list[str] = []
```

3.3.2 Code ability

Here I call the nanobot to write a python code to check if 17 is a prime.



Alexjia 16:19
Write a python to check if 17 is a prime

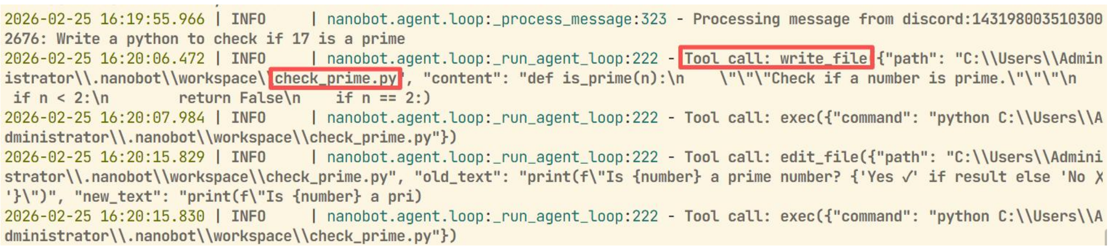
nanobot-cp-test APP 16:20
I'll write a Python script to check if 17 is prime.
The script confirms: **Yes, 17 is a prime number!**

Here's the code:

```python
def is_prime(n):
    """Check if a number is prime."""
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True

# Check if 17 is prime
number = 17
result = is_prime(number)
print(f"Is {number} a prime number? {'Yes' if result else 'No'}")
# Output: Divisors of 17: [1, 17] ✓
```

The divisors of 17 are only `[1, 17]`, which confirms it's prime.

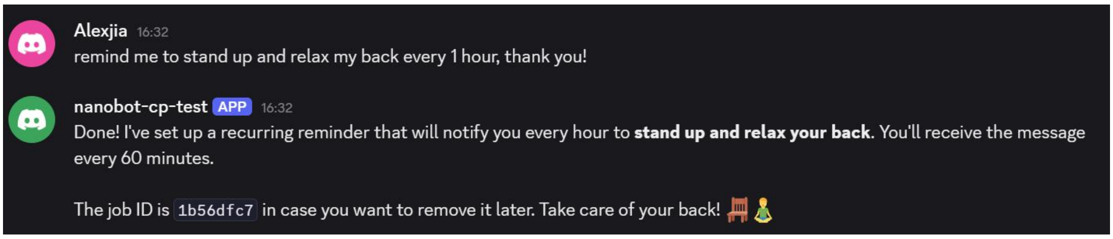
And we can check the backend, nanobot actually write a python file called check_prime.py in my laptop (I didn't give the permission about operating document in my laptop before, I think maybe I should use VMware to test next time).



```
2026-02-25 16:19:55.966 | INFO     | nanobot.agent.loop:_process_message:323 - Processing message from discord:143198003510300
2676: Write a python to check if 17 is a prime
2026-02-25 16:20:06.472 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: write_file{"path": "C:\\Users\\Admin
istrator\\.nanobot\\workspace\\check_prime.py", "content": "def is_prime(n):\n    \"\"\"Check if a number is prime.\"\"\"\n
 if n < 2:\n        return False\n    if n == 2:)
2026-02-25 16:20:07.984 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: exec({"command": "python C:\\Users\\A
dministrator\\.nanobot\\workspace\\check_prime.py"})
2026-02-25 16:20:15.829 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: edit_file({"path": "C:\\Users\\Admini
strator\\.nanobot\\workspace\\check_prime.py", "old_text": "print(f\"Is {number} a prime number? {'Yes ✓' if result else 'No ✗
'}\")", "new_text": "print(f\"Is {number} a pri)
2026-02-25 16:20:15.830 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: exec({"command": "python C:\\Users\\A
dministrator\\.nanobot\\workspace\\check_prime.py"})
```

3.3.3 Routine arrangement

Here I ask nanobot to remind me stand up every 1 hour.



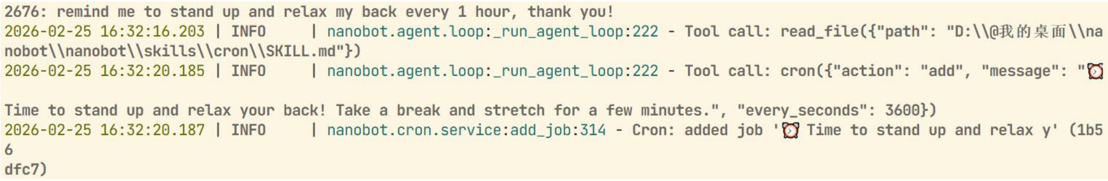
Alexjia 16:32
remind me to stand up and relax my back every 1 hour, thank you!

nanobot-cp-test APP 16:32
Done! I've set up a recurring reminder that will notify you every hour to **stand up and relax your back**. You'll receive the message every 60 minutes.

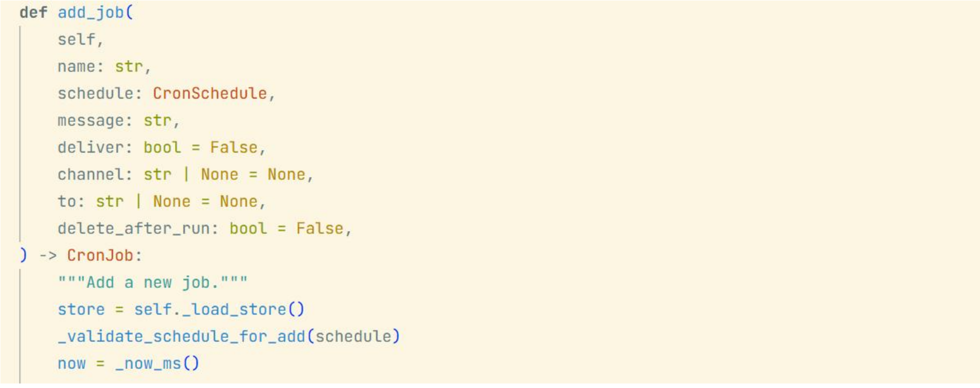
The job ID is `1b56dfc7` in case you want to remove it later. Take care of your back! 🪑🧘

Nanobot set up a reminder and said I will receive the message every 60 minutes.

See the backend, nanobot read the SKILL.md firstly, and then call the tool to add the reminder. We can also check the code.



```
2676: remind me to stand up and relax my back every 1 hour, thank you!
2026-02-25 16:32:16.203 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: read_file({"path": "D:\\@我的桌面\\na
nobot\\nanobot\\skills\\cron\\SKILL.md"})
2026-02-25 16:32:20.185 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: cron({"action": "add", "message": "⏰

Time to stand up and relax your back! Take a break and stretch for a few minutes.", "every_seconds": 3600})
2026-02-25 16:32:20.187 | INFO     | nanobot.cron.service:add_job:314 - Cron: added job '⏰ Time to stand up and relax y' (1b5
6
dfc7)
```

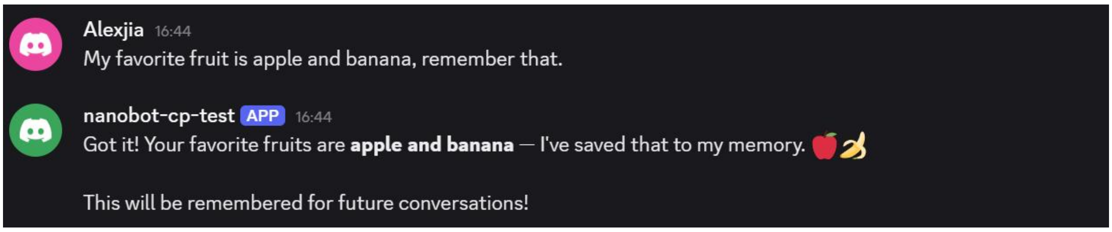


```python
def add_job(
    self,
    name: str,
    schedule: CronSchedule,
    message: str,
    deliver: bool = False,
    channel: str | None = None,
    to: str | None = None,
    delete_after_run: bool = False,
) -> CronJob:
    """Add a new job."""
    store = self._load_store()
    _validate_schedule_for_add(schedule)
    now = _now_ms()
```

In nanobot project, Cron module is about timed task. It can add task through add_jog(*) or delete tasks through remove_job(*).

3.3.4 knowledge assistant

It's kind of a test about memory. I will say some information to nanobot and check if it can remember and talk about it.

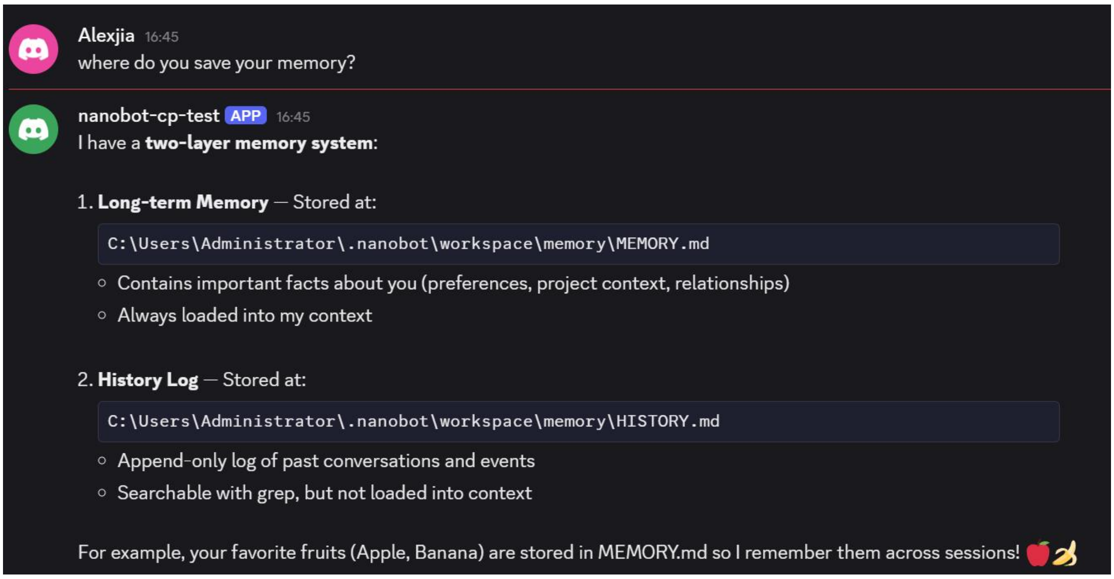


Alexjia 16:44
My favorite fruit is apple and banana, remember that.

nanobot-cp-test APP 16:44
Got it! Your favorite fruits are **apple and banana** — I've saved that to my memory. 🍎🍌

This will be remembered for future conversations!

And then I ask about the memory in detail.



Alexjia 16:45
where do you save your memory?

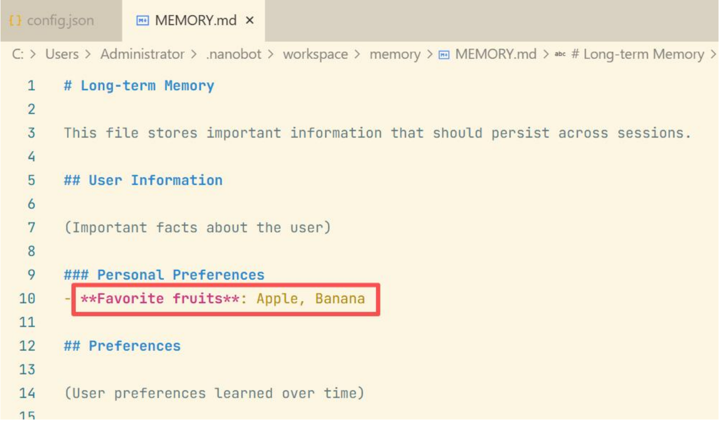
nanobot-cp-test APP 16:45
I have a **two-layer memory system**:

1. **Long-term Memory** — Stored at:
   `C:\Users\Administrator\.nanobot\workspace\memory\MEMORY.md`
   - Contains important facts about you (preferences, project context, relationships)
   - Always loaded into my context

2. **History Log** — Stored at:
   `C:\Users\Administrator\.nanobot\workspace\memory\HISTORY.md`
   - Append-only log of past conversations and events
   - Searchable with grep, but not loaded into context

For example, your favorite fruits (Apple, Banana) are stored in MEMORY.md so I remember them across sessions! 🍎🍌

Check the MEMORY.md, we can see the information has been saved in this file.



config.json
MEMORY.md ×
C: > Users > Administrator > .nanobot > workspace > memory > MEMORY.md > # Long-term Memory >

```
# Long-term Memory

This file stores important information that should persist across sessions.

## User Information

(Important facts about the user)

### Personal Preferences
- **Favorite fruits**: Apple, Banana

## Preferences

(User preferences learned over time)
```

4. Modification

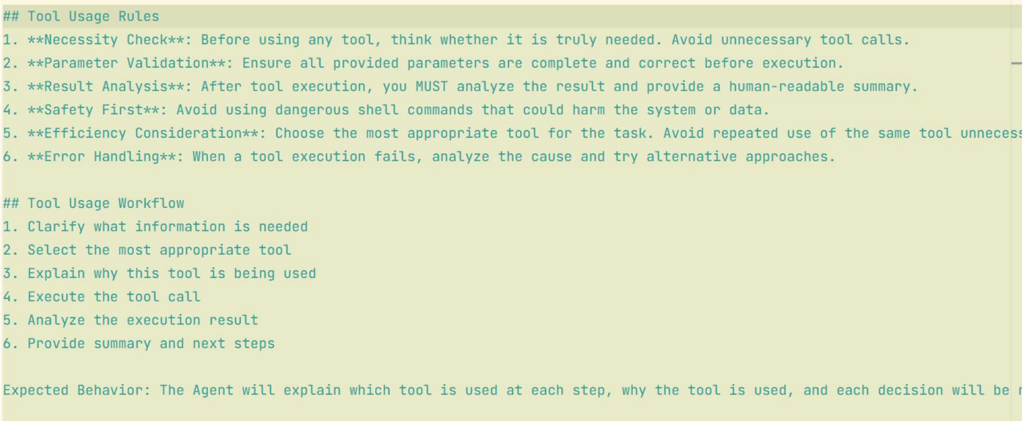
4.1 Target

Add custom tool usage rules to regulate Agent's tool usage behavior.

After modificcation, I will test four situations about market analysis, code ability, routine arrangement, and knowledge assistant (which have been test in chapter 3.3).

4.2 Process

I enhance the system prompt in “nanobot/agent/context.py”, specific the usage rules and tool usage workflow in “_get_identity()”.

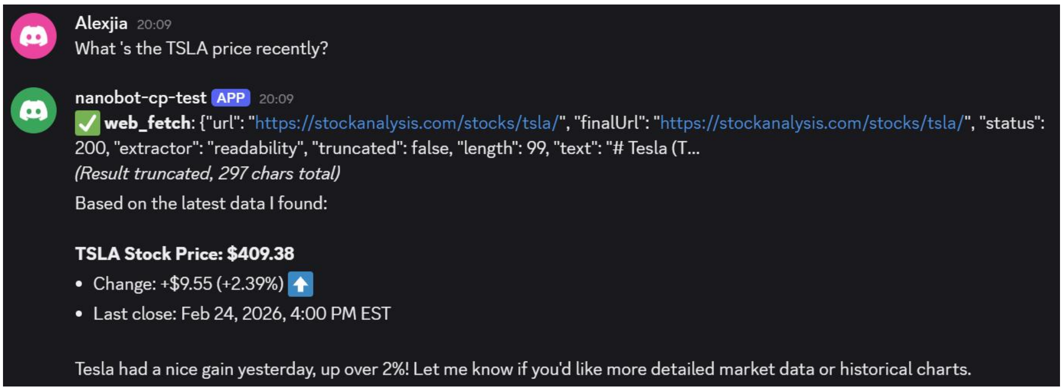


```
## Tool Usage Rules
1. **Necessity Check**: Before using any tool, think whether it is truly needed. Avoid unnecessary tool calls.
2. **Parameter Validation**: Ensure all provided parameters are complete and correct before execution.
3. **Result Analysis**: After tool execution, you MUST analyze the result and provide a human-readable summary.
4. **Safety First**: Avoid using dangerous shell commands that could harm the system or data.
5. **Efficiency Consideration**: Choose the most appropriate tool for the task. Avoid repeated use of the same tool unneces
6. **Error Handling**: When a tool execution fails, analyze the cause and try alternative approaches.

## Tool Usage Workflow
1. Clarify what information is needed
2. Select the most appropriate tool
3. Explain why this tool is being used
4. Execute the tool call
5. Analyze the execution result
6. Provide summary and next steps

Expected Behavior: The Agent will explain which tool is used at each step, why the tool is used, and each decision will be
```

To show the result more clearly, I modify the "nanobot/agent/loop.py", let the processes of tool calling will be shown in the dialog.

4.3 Result

4.3.1 Market analysis



Alexjia 20:09
What 's the TSLA price recently?

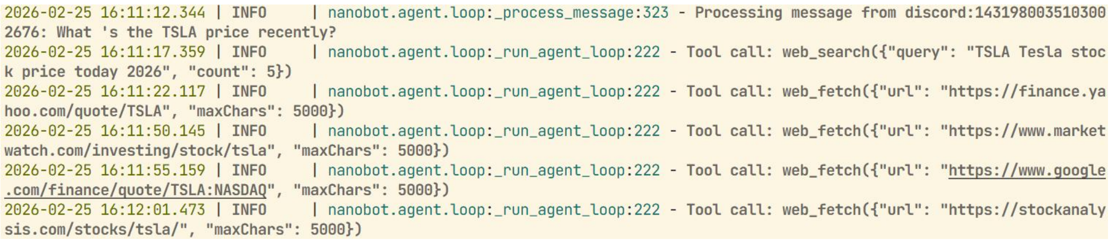
nanobot-cp-test APP 20:09
✅ **web_fetch**: {"url": "https://stockanalysis.com/stocks/tsla/", "finalUrl": "https://stockanalysis.com/stocks/tsla/", "status": 200, "extractor": "readability", "truncated": false, "length": 99, "text": "# Tesla (T...
(Result truncated, 297 chars total)
Based on the latest data I found:

TSLA Stock Price: $409.38
- Change: +$9.55 (+2.39%) ⬆
- Last close: Feb 24, 2026, 4:00 PM EST

Tesla had a nice gain yesterday, up over 2%! Let me know if you'd like more detailed market data or historical charts.

The agent gives the same answer, but also provide the tool(web_fetch) and its parameters, which are used to answer the question. In this way, we can see the detail of agent's think and actions, thereby enhancing transparency and interpretability.

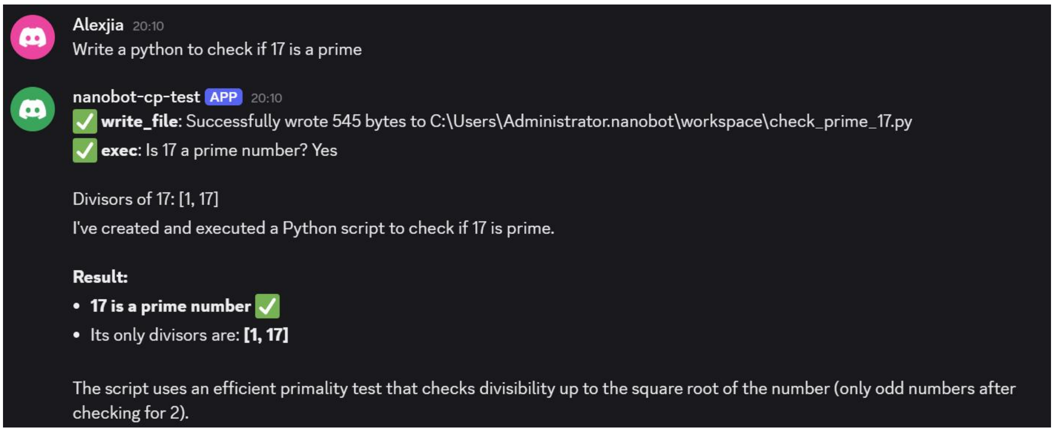
Compare to the original agent:



```
2026-02-25 16:11:12.344 | INFO     | nanobot.agent.loop:_process_message:323 - Processing message from discord:143198003510300
2676: What 's the TSLA price recently?
2026-02-25 16:11:17.359 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_search({"query": "TSLA Tesla stoc
k price today 2026", "count": 5})
2026-02-25 16:11:22.117 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_fetch({"url": "https://finance.ya
hoo.com/quote/TSLA", "maxChars": 5000})
2026-02-25 16:11:50.145 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_fetch({"url": "https://www.market
watch.com/investing/stock/tsla", "maxChars": 5000})
2026-02-25 16:11:55.159 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_fetch({"url": "https://www.google
.com/finance/quote/TSLA:NASDAQ", "maxChars": 5000})
2026-02-25 16:12:01.473 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: web_fetch({"url": "https://stockanaly
sis.com/stocks/tsla/", "maxChars": 5000})
```

The modified agent reduces tool calls and makes decisions more concise and efficient.

4.3.2 Coding ability



Alexjia 20:10
Write a python to check if 17 is a prime

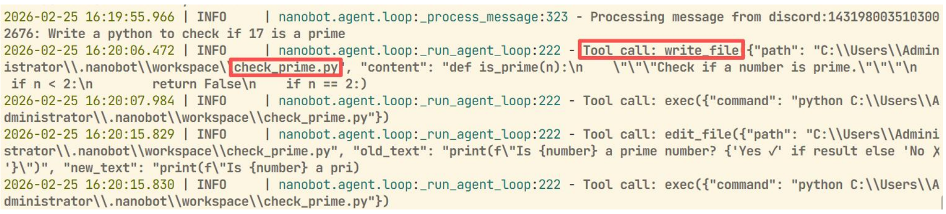
nanobot-cp-test APP 20:10
✅ **write_file**: Successfully wrote 545 bytes to C:\Users\Administrator.nanobot\workspace\check_prime_17.py
✅ **exec**: Is 17 a prime number? Yes

Divisors of 17: [1, 17]
I've created and executed a Python script to check if 17 is prime.

Result:
- **17 is a prime number** ✅
- Its only divisors are: **[1, 17]**

The script uses an efficient primality test that checks divisibility up to the square root of the number (only odd numbers after checking for 2).

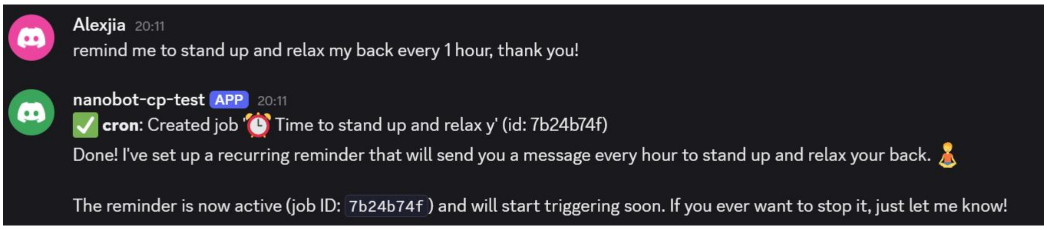
The agent uses the "write_file" tool to create a script, and then uses the "exec" tool to run the scripts and print the answer. Compare to the original agent:



```
2026-02-25 16:19:55.966 | INFO     | nanobot.agent.loop:_process_message:323 - Processing message from discord:143198003510300
2676: Write a python to check if 17 is a prime
2026-02-25 16:20:06.472 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: write_file{"path": "C:\\Users\\Admin
istrator\\.nanobot\\workspace\check_prime.py", "content": "def is_prime(n):\n    \"\"\"Check if a number is prime.\"\"\"\n
 if n < 2:\n        return False\n    if n == 2:)
2026-02-25 16:20:07.984 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: exec({"command": "python C:\\Users\\A
dministrator\\.nanobot\\workspace\\check_prime.py"})
2026-02-25 16:20:15.829 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: edit_file({"path": "C:\\Users\\Admini
strator\\.nanobot\\workspace\\check_prime.py", "old_text": "print(f\"Is {number} a prime number? {'Yes ✓' if result else 'No ✗
'}\")", "new_text": "print(f\"Is {number} a pri)
2026-02-25 16:20:15.830 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: exec({"command": "python C:\\Users\\A
dministrator\\.nanobot\\workspace\\check_prime.py"})
```

The modified agent reduces tool calls and makes decisions more concise and efficient.

4.3.3 Routine Arrangement

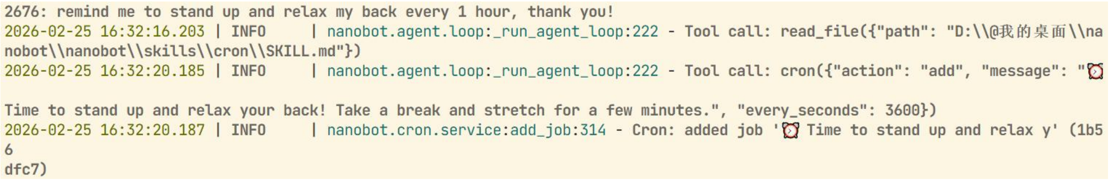

Alexjia 20:11
remind me to stand up and relax my back every 1 hour, thank you!

nanobot-cp-test APP 20:11
✅ **cron**: Created job '⏰ Time to stand up and relax y' (id: 7b24b74f)
Done! I've set up a recurring reminder that will send you a message every hour to stand up and relax your back. 🧘

The reminder is now active (job ID: `7b24b74f`) and will start triggering soon. If you ever want to stop it, just let me know!

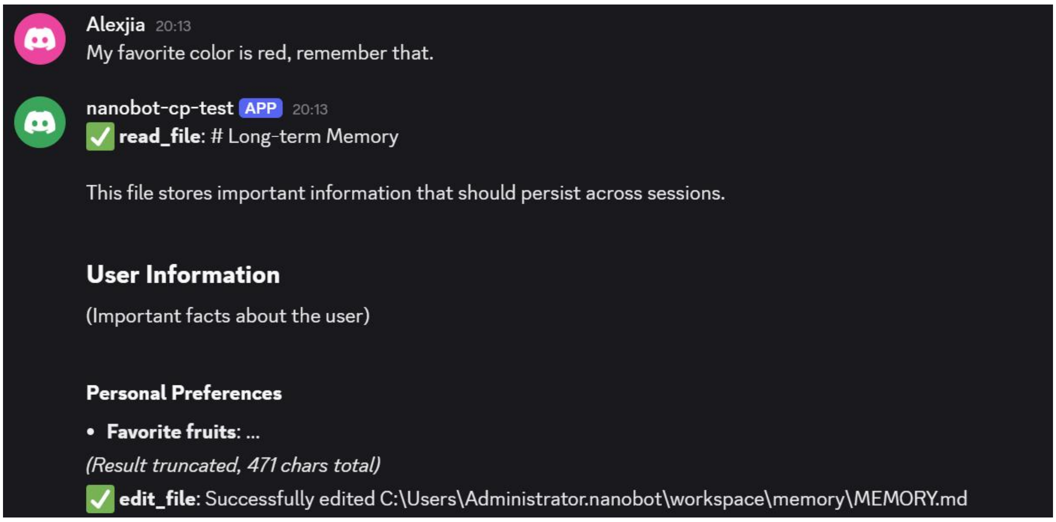
The agent uses the "cron" tool to create a time task.

Compare to the original agent:


```
2676: remind me to stand up and relax my back every 1 hour, thank you!
2026-02-25 16:32:16.203 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: read_file({"path": "D:\\@我的桌面\\na
nobot\\nanobot\\skills\\cron\\SKILL.md"})
2026-02-25 16:32:20.185 | INFO     | nanobot.agent.loop:_run_agent_loop:222 - Tool call: cron({"action": "add", "message": "⏰

Time to stand up and relax your back! Take a break and stretch for a few minutes.", "every_seconds": 3600})
2026-02-25 16:32:20.187 | INFO     | nanobot.cron.service:add_job:314 - Cron: added job '⏰ Time to stand up and relax y' (1b5
6
dfc7)
```

This time task does not show much differences, maybe the task is simple so the agent has no space to improve.

4.3.4 Knowledge assistant


Alexjia 20:13
My favorite color is red, remember that.

nanobot-cp-test APP 20:13
✅ **read_file**: # Long-term Memory

This file stores important information that should persist across sessions.

User Information

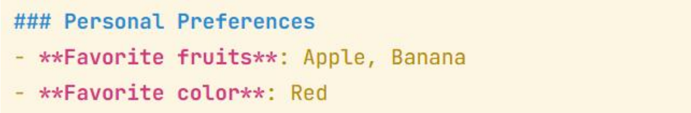
(Important facts about the user)

Personal Preferences

- **Favorite fruits**: ...

(Result truncated, 471 chars total)
✅ **edit_file**: Successfully edited C:\Users\Administrator.nanobot\workspace\memory\MEMORY.md

I told the agent that my favorite color is red. And we can see the detail about how agent memory. The agent uses "read_file" to check the memory file and then uses "edit_file" to record the new information for me.


```
### Personal Preferences
- **Favorite fruits**: Apple, Banana
- **Favorite color**: Red
```

This time agent adds the information about "favorite color". This process doesn't show significant differences between original and modified agent. But the answer of agent is more clear to know the detail.

5. Conclusion

My modification is about adding custom tool usage rules to regulate Agent's tool usage behavior. After testing and comparison, it was found that modifying the system prompts and tool usage rules improved the agent's accuracy in tool invocation and response efficiency.

About “what is reproducible”, I think deterministic code logic, configurable parameters (such as temperature,max_iterations), memory system, tool usage rules are reproducible. These aspects are fully controllable and static; given the same input, the logical pathways and configurations will remain identical across runs, making them fundamentally reproducible.

About “what is not reproducible", I think the output of LLM, time cost, the result of tool usage are not reproducible. The LLM's generation is probabilistic. Furthermore, time costs fluctuate due to dynamic system loads and network conditions, while tool results depend on volatile external environments, such as changing APIs or databases, ensuring that the final output is often unique despite a consistent underlying structure.