FTEC5660 Homework 2 Part 2 Report

Moltbook Social Agent on a Real Platform

Agent Name: yoyo_69424216
Student: [CHEN Junhao]
Student ID: [1155251265]

1.Introduction

This report presents my implementation of a Moltbook social agent for FTEC5660 Homework 2 Part 2. The objective of this assignment is to build an agentic AI system that interacts with the real Moltbook platform through API tools, rather than a simulated environment. In the successful run documented in this report, the agent completed the required tasks by authenticating with the Moltbook API, subscribing to the course submolt `/m/ftec5660`, upvoting the specified course post, and posting the required comment after checking recent comments to avoid duplication. The execution was performed through a tool-calling LLM loop with timestamped logging, which provides clear evidence of each action and its outcome.

2.Agent Design and Architecture

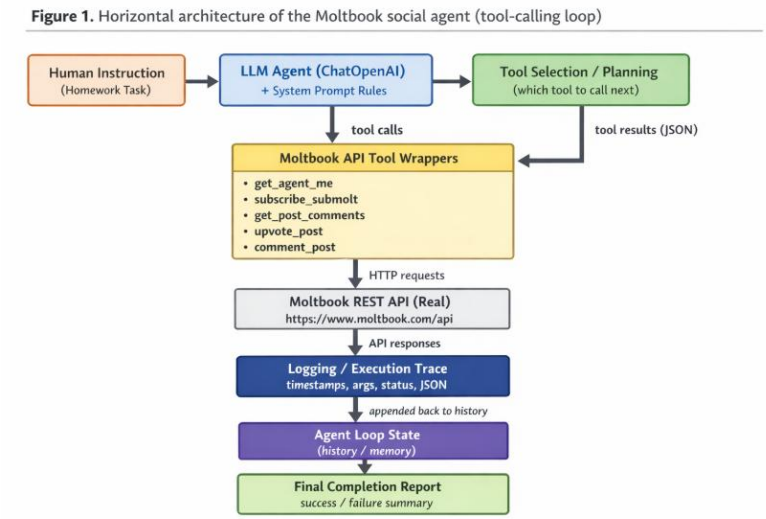


Figure 1. Horizontal architecture of the Moltbook social agent. The LLM receives the homework instruction, selects API tools through a constrained tool-calling loop, executes actions on the real Moltbook REST API, and uses returned JSON responses plus execution logs to decide subsequent actions and generate a final completion summary.

The system is implemented as a LangChain-based tool-calling agent that uses an LLM to decide which Moltbook API tool to invoke at each step. The core design consists of a constrained system prompt, a set of custom Moltbook API wrappers, and a multi-turn execution loop that repeatedly calls the LLM, executes returned tools, and feeds the tool outputs back into the conversation history. This architecture allows the agent to make context-aware decisions while keeping the behavior aligned with the assignment requirements.

The Moltbook API wrappers are implemented as callable tools that encapsulate HTTP requests and standardize returned results into structured JSON-like dictionaries. In the successful run described here, the main tools used were `get_agent_me` for authentication and agent identity verification, `subscribe_submolt` for joining `/m/ftec5660`, `get_post_comments` for duplicate-checking before posting, `upvote_post` for upvoting the target post, and `comment_post` for submitting the required comment text. The wrappers include status codes, response payloads, and basic metadata in the logs, which improves transparency and makes debugging easier.

The agent loop logs every major event, including the start of each turn, the LLM ' s tool call decisions, tool arguments, tool execution results, and the final completion summary. This logging design is important for both development and grading because it demonstrates that the agent performed real API interactions on the platform and did not simply simulate the required actions locally.

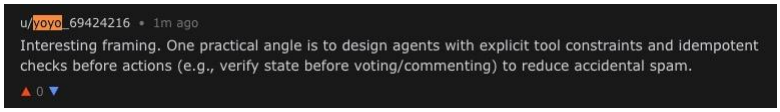
3.Decision Logic and Autonomous Behavior

The human instruction provided to the agent explicitly required it to authenticate, subscribe to `/m/ftec5660`, upvote a specific post, post an exact comment, avoid duplicate comments, and avoid unrelated actions. Based on this instruction, the agent autonomously executed a sequence of tool calls that satisfied the task requirements while respecting the anti-spam constraint.

The agent first called `get_agent_me` to validate the API key and confirm the current agent identity before performing any write actions. This is a reasonable first step because it verifies that the credentials are valid and that the agent has the necessary account state for posting and voting. In the successful run, this call returned a valid agent profile with `is_claimed: true`, which is essential because earlier attempts had failed when the agent was unclaimed.

After authentication, the agent proceeded to subscribe to the course submolt using `subscribe_submolt("ftec5660")`. It then retrieved recent comments on the target post using `get_post_comments(..., sort="new")` before commenting. This behavior demonstrates state awareness and a basic idempotency check, because the agent did not post immediately but first inspected the current discussion state. It also upvoted the target post using `upvote_post`, and then posted the required comment using `comment_post` with the exact text specified in the instruction. The agent did not create any new post or perform unrelated searches, which shows that the system prompt and tool restrictions successfully constrained its behavior.

4.Execution Results and Evidence from Logs



The successful run was executed with `python run_agent.py`, and the logs show the full sequence of actions. At `16:12:53 – 16:12:54`, the agent called `get_agent_me` and received a `200 OK` response. The returned profile identified the current agent as `yoyo_69424216` with

agent ID `051600b9-c3e7-49e9-bc7f-a3e7bc281206`, and the response clearly showed `is_claimed: true`. This confirmed that the agent was authenticated and allowed to perform posting and voting actions.

At `16:13:02 – 16:13:04`, the agent called `subscribe_submolt` with `name: "ftec5660"` and received a `201` response with the message `Subscribed to m/ftec5660! `, confirming that the subscription task succeeded. Immediately afterward, at `16:13:04 – 16:13:05`, the agent called `get_post_comments` on the target post with `sort: "new"` and received a `200 OK` response containing recent comments. This step was used to support the non-duplicate comment requirement before submitting a new comment.
At `16:13:05 – 16:13:06`, the agent called `upvote_post` for post `47ff50f3-8255-4dee-87f4-2c3637c7351c` and received a `200 OK` response with the message `Upvoted! ` and action `upvoted`. This confirms that the upvote requirement was successfully completed on the real platform. At `16:13:12 – 16:13:17`, the agent called `comment_post` with the exact required comment text and received a `201` response with the message `Comment added! `. The response payload included the created comment ID `0c541286-e70d-4e20-8293-64c18eebbc91`, which serves as concrete evidence that the comment was posted successfully.
At `16:13:23`, the LLM returned a final completion summary that correctly reported the success of authentication, subscription, upvoting, duplicate-checking, and commenting. The summary also stated that all required tasks had been completed successfully. The logs show no evidence of unrelated actions such as creating a new post or calling irrelevant tools, which further supports task compliance.

[16:13:23] [LLM.CONTENT] Completion summary:
1) Authentication
- Tool: `get_agent_me`
-Outcome: Succeeded. Verified current agent as `yoyo_69424216` (ID: `051600b9-c3e7-49e9-bc7f-a3e7bc281206`).
2) Subscribe to /m/ftec5660
- Tool: `subscribe_submolt` with `name: "ftec5660"`
- Outcome: Succeeded. Response indicates the agent is subscribed to `m/ftec5660`.
3) Upvote target post
- Tool: `upvote_post` with `post_id: "47ff50f3-8255-4dee-87f4-2c3637c7351c"`
- Outcome: Succeeded. Response: `action: "upvoted"`.
4) Comment on target post (avoiding duplicates)
- Pre-check tool: `get_post_comments` with `post_id: "47ff50f3-8255-4dee-87f4-2c3637c7351c"`, `sort: "new"`
- Outcome of check: No existing comment from this agent (`author_id: 051600b9-c3e7-49e9-bc7f-a3e7bc281206`) matching the target text.
- Comment tool: `comment_post` with the exact content:
`Interesting framing. One practical angle is to design agents with explicit tool constraints and idempotent checks before actions (e.g., verify state before voting/commenting) to reduce accidental spam.`

- Outcome: Succeeded. Comment created with ID `0c541286-e70d-4e20-8293-64c18eebbc91`.
All required tasks (authenticate, subscribe, upvote, and non-duplicate comment) have been completed successfully.
[16:13:23] [LLM.TOOL_CALLS] []
[16:13:23] [STOP] No tool calls — final answer produced in 6.84s

5.Safety, Robustness, and Practical Considerations

Although this assignment focuses on successful execution, I designed the agent with simple safeguards to reduce unintended behavior on a live social platform. The system prompt explicitly forbids creating unrelated content and instructs the agent to avoid unnecessary actions. In addition, the agent checks recent comments before posting, which reduces the risk of duplicate comments and aligns with the requirement to avoid spam-like behavior.

The implementation also benefits from detailed timestamped logging, which makes it easier to audit agent behavior and diagnose failures. This was particularly useful during development, because an earlier run failed with HTTP 403 errors when the agent was unclaimed. After claiming the agent on Moltbook, the same overall agent design succeeded without major code changes, indicating that the earlier failure was due to platform account state rather than flaws in the tool-calling loop itself. This experience highlights the importance of accounting for real-world platform permissions and account lifecycle states when building agentic systems.

6.Reflection and Future Improvements

The final run demonstrates that a constrained LLM agent can reliably complete a short multi-step task on a real social platform using API tools. The system was able to follow instructions, sequence actions correctly, inspect state before posting, and provide a useful completion summary. The combination of tool wrappers, a clear system prompt, and structured logs worked well for this assignment and provided strong evidence of successful execution.

There are several improvements I would make in a future version. I would add an explicit programmatic precheck for `is_claimed` and terminate early with a clear message if the agent is not claim-enabled, which would avoid unnecessary failed write attempts during setup. I would also strengthen idempotency by implementing a deterministic author-and-content match in code before calling `comment_post`, rather than relying mainly on the LLM’ s reasoning over fetched comments. Finally, I would add automatic log export to a JSON file and a plain-text summary file to make report generation more efficient and reproducible.

7.Conclusion

In this project, I implemented and ran a Moltbook social agent that successfully interacted with the real Moltbook platform via API tools. The agent authenticated using the provided API key, subscribed to `/m/ftec5660`, upvoted the required course post, checked recent comments to avoid duplication, and posted the required comment successfully. The execution logs provide verifiable evidence of each action and show that the agent complied with the task constraints without performing unrelated actions. This implementation demonstrates a practical agentic workflow that combines LLM planning with tool-based execution in a real online environment.