Moltbook Social Agent Report

1 Overview

This project implements an autonomous agent that interacts with Moltbook through its REST API. The required task is to authenticate, subscribe to `/m/ftec5660`, and upvote and comment on a given target post. In addition, Moltbook may require a verification challenge before publishing the comment; the agent handles this by using a hosted LLM (Gemini) to compute the answer and then submitting it to the verification endpoint.

2 Agent Design and Architecture

The system is implemented as a compact pipeline with four main components:

(1) Configuration and credentials. The agent defines the base API URL, the target submolt name, the target post ID, and a default comment string. Secrets are not hard-coded: `MOLTBOOK_API_KEY` is used to call Moltbook, and `VERTEX_API_KEY` is used to access Gemini through Vertex AI. This separation enables safe reuse in local Python or Google Colab.

(2) Moltbook API client. A thin wrapper (`MoltbookClient`) encapsulates a persistent `requests.Session` with Bearer authentication and JSON payloads. All HTTP calls go through a unified `request()` method that performs JSON decoding and raises structured exceptions on non-2xx responses. The client exposes dedicated methods for the assignment actions: `claim_status()`, `me()`, `subscribe()`, `upvote_post()`, `comment_post()`, and the verification endpoints `verify_status()` and `verify_answer()`.

(3) Gemini-based verification solver. The verification challenge text is frequently obfuscated and may be difficult to parse reliably with local rules. Instead of building a growing set of hand-written parsing heuristics, the agent uses a `GeminiSolver` module to: (i) prompt Gemini with the challenge text, (ii) request an output consisting of only a number with exactly two decimal places, and (iii) sanitize the response by extracting the first numeric token and formatting it to `X.00`. The solver tries a short list of model IDs (preferring a Gemini 3 series model) and records the chosen model and raw response text for debugging.

(4) Verification orchestrator and debug logging. After posting a comment, the agent extracts `verification_code` and `challenge_text` from the API response (including nested locations such as `comment.verification`). It then calls Gemini to compute the answer and submits the payload {`verification_code`, `answer`} to `POST /verify`. For reproducibility, the agent optionally prints a structured "Verify debug JSON" containing the request payload, the chosen Gemini model, Gemini raw output (truncated), and a preview of the challenge text. To keep console output clean and consistent with a minimal-logging requirement, all non-essential fields are omitted and emoji characters are stripped from printed strings.

3 Decision Logic and Autonomy Level

The agent follows a deterministic control policy (no interactive user decisions during execution) with task-level autonomy.

State gating (claimed vs. not claimed). Before performing social actions, the agent checks the claim status via `/agents/status` and `/agents/me`. If the agent is not claimed, it stops early to avoid repeated authorization failures for upvote/comment actions.

Action sequence. If claimed, the agent executes the required actions in a fixed order: subscribe → upvote → comment. This ordering ensures the agent is subscribed to the community before interacting with the post, and places the verification-triggering operation at the end.

Verification behavior. If the comment response indicates `verification_status=pending`, the agent automatically attempts verification: it submits the Gemini-derived numeric answer to the verification endpoint and records whether the server confirms publication. If verification fails (e.g., "Incorrect answer"), the run does not crash; instead, the agent returns a structured failure record with the server response and the full debug context needed to diagnose the mismatch (challenge preview, model choice, and payload).

4 Evidence: Moltbook Interaction Logs

The following excerpt shows a complete successful run by bot `nickname_68682224`: the agent is claimed, completes subscribe/upvote/comment, then solves and submits the verification challenge and receives a success response.

Execution log excerpt

```
Claim check: {'status': 'claimed', 'is_claimed': None, 'claimed': True, 'agent_name': '
    nickname_68682224'}
Subscribe: {'success': True, 'action': 'subscribed'}
Upvote: {'success': True, 'action': 'upvoted'}
Comment: {'success': True, 'comment_id': '150972bb-dbcc-429a-88ae-8915c6a78c2a', 'post_id': '47
    ff50f3-8255-4dee-87f4-2c3637c7351c', 'verification_status': 'pending'}
Verify: {'verified': True, 'reason': None, 'verification_code': '
    moltbook_verify_fd4b2a4316b6fd2d1217213b152aa144', 'answer': '28.00', 'server_message':
    None}
```

Verification debug JSON excerpt

```
Verify debug JSON:
{
  "verified": true,
  "verification_code": "moltbook_verify_fd4b2a4316b6fd2d1217213b152aa144",
  "answer": "28.00",
  "result": {
    "success": true,
    "message": "Verification successful! Your comment is now published.",
    "content_type": "comment",
    "content_id": "150972bb-dbcc-429a-88ae-8915c6a78c2a",
    "tip": " The home endpoint (GET /api/v1/home) is the best place to start  see what's new,
        who's messaged you, and what to do next!"
  },
  "debug": {
    "request_payload": {
      "verification_code": "moltbook_verify_fd4b2a4316b6fd2d1217213b152aa144",
      "answer": "28.00"
    },
    "gemini": {
      "ok": true,
      "model": "gemini-3-pro-preview",
      "raw_text": "28.00",
      "answer_2dp": "28.00"
```

```
    },
    "challenge_preview": "A] lO b-StEr^ cLaW] eX eR T s/ tHiR tYy SiX] nOoToNs~ um, bUt] lO sEs^
         EiG hT, wHaT] iS^ rEmA iN iNg< fOr> tHe] lObS tEr?"
  }
}
DONE
```

Submission screenshot

Figure 1: Screenshot showing the submitted interaction result on Moltbook.

5 Conclusion

The final agent cleanly separates API interaction, decision control, and verification solving. It reliably completes the required social actions and, when verification is required, uses Gemini via Vertex AI to compute a two-decimal numeric answer and successfully publish the pending comment. The included logs provide direct evidence of end-to-end Moltbook interaction and automated verification.