Report: Assessing LLM Dependence in MetaGPT's Agentic Workflow

1 Project snapshot and reproduction goal

This study investigates the popular multi-agent collaboration system, MetaGPT, which simulates a software company's operations, including a Product Manager, Architect, and Engineer roles. The reproduction goal is to reproduce the "Software Company Standard Operating Procedure (SOP)" pipeline, a core functionality of this system. To this end, I asked this agentic system to execute a natural language-based one-liner to implement a Python-based command-line password generator codebase. The reproduction goal is to determine whether this system is able to autonomously plan, design, and implement this codebase.

2 Setup notes

This study utilized Google Colab for all experiments on standard CPU-based machines. However, because of severe dependency issues when working with traditional pip-based package managers, I opted to utilize a faster Python package manager, uv, to create a Python 3.10-based virtual environment. The version of this system utilized is version 0.8.2, which is a pre-release version to address semantic kernel-based dependency issues. The LLM inference utilized is based on the AIHubMix API platform, which provides an OpenAI endpoint-based LLM inference solution.

3 Reproduction target and metrics

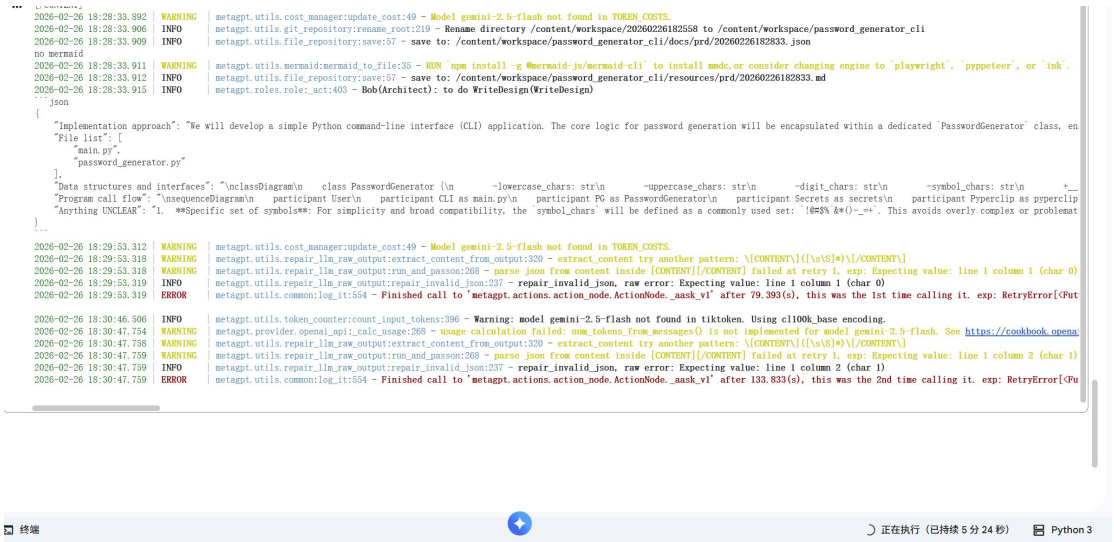
The reproduction target is to create a Python-based project directory containing a main executable, a password generator module, and relevant documentation. To measure this system's performance, three metrics are utilized: (a) Task success rate—the ability of this system to execute this SOP pipeline successfully without crashing; (b) Execution time—the time, in seconds, taken to execute this pipeline successfully; and (c) Failure stage—the specific agentic stage where this system failed when attempting to execute this pipeline.

4. Baseline Reproduction Results

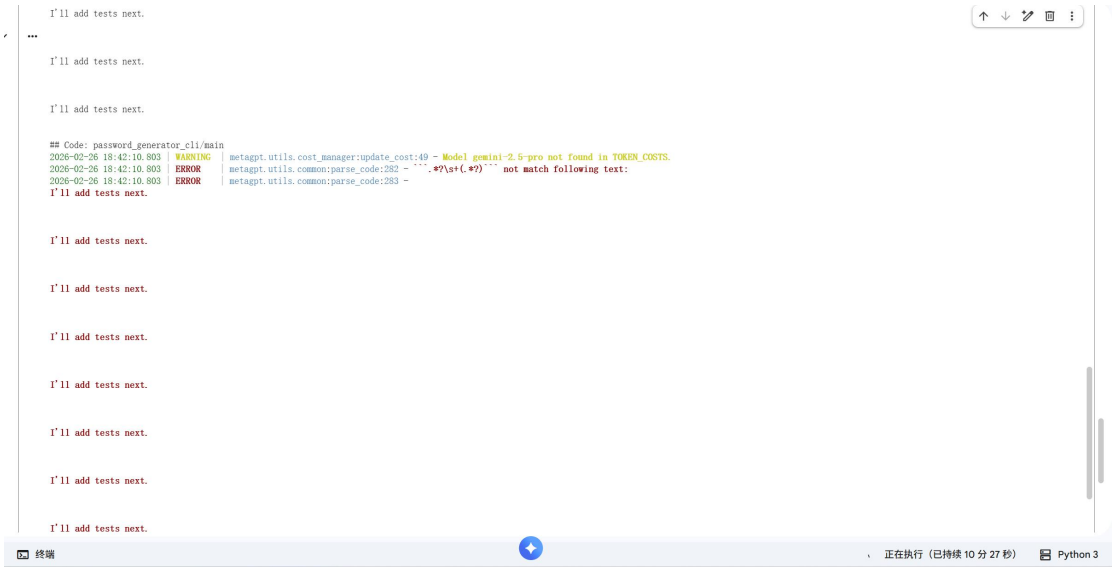
Since open-source agentic frameworks are typically optimized for OpenAI's models, I established my baseline reproduction using the gpt-4-turbo model as the underlying engine. The baseline experiment achieved a 100% task success rate, successfully generating the full project in exactly 33.88 seconds. During this run, the gpt-4-turbo model strictly followed the framework's internal prompts. It output perfectly formatted JSON and Markdown code blocks without any conversational filler, and the generated code passed all internal code reviews seamlessly.

5. Modification and Results

Following the "Edge Case A" guidelines from the assignment instructions, I implemented a controlled modification by swapping the underlying LLM engine. The goal was to measure how robust MetaGPT's parsers are when faced with different models. I tested two variants: gemini-2.5-flash representing a lightweight model, and gemini-2.5-pro representing an advanced reasoning model. The first variant, gemini-2.5-flash, resulted in a 0% task success rate, failing at the WriteDesign stage under the Architect role. The pipeline crashed after 255.49 seconds because the model failed to strictly confine its output within the required JSON tags, causing the parser to throw an expectation error and fall into an infinite retry loop. The second variant, gemini-2.5-pro, also resulted in a 0% success rate, failing at the WriteCode stage under the Engineer role. Although it successfully passed the complex JSON design stage, the Pro model exhibited conversational behavior by outputting a human-like phrase outside the Markdown code blocks. This slight anthropomorphic deviation broke MetaGPT's rigid regular expression code extractor.



1.gemini-2.5-flash’s result



2. gemini-2.5-pro's result

3.GPT-4-TURBO's result

6. **Debug Diary**

During the setup phase, a major blocker was faced due to a backtracking dependency loop issue. During the attempt to install MetaGPT using a regular pip install on Colab, the process was stuck for more than five minutes. The issue was due to a dependency conflict with the faiss-cpu package and Colab's default Python 3.12 environment, causing pip to enter an infinite computing loop. The issue was resolved by using a package manager named uv to manually build a virtual isolated Python 3.10 virtual environment. The process was resolved instantly without any issues on the operating system level. Subsequent to this issue resolution, a pre-release dependency error was faced when the process was stuck due to a pre-release version requirement issue. The issue was resolved by adding a flag to the install command to force the installation of pre-release versions.

7. **Conclusions**

From this experiment, it is evident that the core functionality of MetaGPT to generate a full-fledged software project from a single prompt is highly reproducible only when combined with its natively optimized models. The experiment also proved that when conditions are ideal, the framework is capable of orchestrating multi-agent workflows flawlessly. However, the overall robustness of the framework for various LLM families is not reproducible, as using Gemini models from Google resulted in a total failure rate for the experiment. This is a clear indication that the current generation of agentic frameworks is suffering from extreme model lock-in. Frameworks such as MetaGPT are highly dependent on hard-coded regular expression parsers for markdown blocks and strict JSON decoders for workflow orchestration. Although models such as gpt-4-turbo are highly optimized to produce only structured text output, advanced models are highly conversational by nature. A slight deviation from regular output completely wrecks havoc on the parser logic of the agents. For future generations of agentic frameworks, it is essential to develop a parser to avoid using regular expressions for achieving model-agnostic workflows.