

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.

```
...
2026-02-26 18:28:33.892 | WARNING | metagpt.utils.cost_manager:update_cost:49 - Model gemini-2.5-flash not found in TOKEN_COSTS.
2026-02-26 18:28:33.906 | INFO  | metagpt.utils.git_repository:rename_root:219 - Rename directory /content/workspace/202602261828558 to /content/workspace/password_generator_cli
2026-02-26 18:28:33.909 | INFO  | metagpt.utils.file_repository:save:57 - save to: /content/workspace/password_generator_cli/docs/prd/20260226182833.json
...
2026-02-26 18:28:33.911 | WARNING | metagpt.utils.npm:npm:run:36 - RUN npm install -g mermaid-js/mermaid-clc to install mdc, or consider changing engine to 'playwright', 'puppeteer', or 'ink'.
2026-02-26 18:28:33.912 | INFO  | metagpt.utils.file_repository:save:57 - save to: /content/workspace/password_generator_cli/resources/prd/20260226182833.md
2026-02-26 18:28:33.915 | INFO  | metagpt.roles.role:.act:403 - Bob(Architect): to do WriteDesign(WriteDesign)
```
json
{
 "Implementation approach": "We will develop a simple Python command-line interface (CLI) application. The core logic for password generation will be encapsulated within a dedicated 'PasswordGenerator' class, enclosed in a module named 'password_generator'. The module will contain the following files: 'main.py' and 'password_generator.py'. The 'main.py' file will serve as the entry point for the CLI, while 'password_generator.py' will contain the logic for generating passwords based on user input and configuration parameters. The 'password_generator' module will be imported into 'main.py' to facilitate the creation of PasswordGenerator instances.",

 "File list": [
 "main.py",
 "password_generator.py"
],
 "Data structures and interfaces": "The application will use a class diagram to define the data structures and interfaces. The 'PasswordGenerator' class will have attributes for lowercase_chars, uppercase_chars, digit_chars, symbol_chars, and pyperclip. The 'PasswordGenerator' class will have methods for generating passwords and interacting with the user via the CLI. The 'main.py' file will call the 'PasswordGenerator' class to generate a password and print it to the console.",

 "Program call flow": "The program flow will start with the user running the 'main.py' script. This script will initialize a 'PasswordGenerator' object and prompt the user for input. The user will provide the number of characters desired and any other specific requirements. The 'PasswordGenerator' object will use this information to generate a password and return it to the 'main.py' script. Finally, the 'main.py' script will print the generated password to the terminal.",

 "Anything UNCLEAR": "There is no specific set of symbols defined in the code. For simplicity and broad compatibility, the symbol_chars will be defined as a commonly used set: '!@#$%^&()_+=+'. This avoids overly complex or problematic characters in the generated password."}
...
2026-02-26 18:29:53.312 | WARNING | metagpt.utils.cost_manager:update_cost:49 - Model gemini-2.5-flash not found in TOKEN_COSTS.
2026-02-26 18:29:53.318 | WARNING | metagpt.utils.repair_llm_raw_output:extract_content_from_output:320 - extract content try another pattern: \[CONTENT\]((\s|S*)+)\[CONTENT\]
2026-02-26 18:29:53.318 | WARNING | metagpt.utils.repair_llm_raw_output:run_and_passon:268 - parse json from content inside [CONTENT]\[CONTENT] failed at retry 1. exp: Expecting value: line 1 column 1 (char 0)
2026-02-26 18:29:53.319 | ERROR | metagpt.utils.repair_llm_raw_output:repair_invalid_json:237 - repair_invalid_json, raw_error: Expecting value: line 1 column 1 (char 0)
2026-02-26 18:29:53.319 | ERROR | metagpt.utils.common:.log:1554 - Finished call to metagpt.actions.action_node.ActionNode._ask_v1 after 79.393(s), this was the 1st time calling it. exp: RetryError[Put
2026-02-26 18:30:46.506 | INFO | metagpt.utils.token_counter:count_input_tokens:296 - Warning: model gemini-2.5-flash not found in tiktoken. Using cl100k_base encoding.
2026-02-26 18:30:47.754 | WARNING | metagpt.provider.openai_api:.calc_usage:268 - usage calculation failed: num_tokens_from_messages() is not implemented for model gemini-2.5-flash. See https://cookbook.openai.com/en/stable/tiktoken.html#usage-calculation
2026-02-26 18:30:47.758 | WARNING | metagpt.utils.repair_llm_raw_output:extract_content_from_output:320 - extract content try another pattern: \[CONTENT\]((\s|S*)+)\[CONTENT\]
2026-02-26 18:30:47.759 | WARNING | metagpt.utils.repair_llm_raw_output:run_and_passon:268 - parse json from content inside [CONTENT]\[CONTENT] failed at retry 1. exp: Expecting value: line 1 column 2 (char 1)
2026-02-26 18:30:47.759 | INFO | metagpt.utils.repair_llm_raw_output:repair_invalid_json:237 - repair_invalid_json, raw_error: Expecting value: line 1 column 2 (char 1)
2026-02-26 18:30:47.759 | ERROR | metagpt.utils.common:.log:1554 - Finished call to metagpt.actions.action_node.ActionNode._ask_v1 after 133.833(s), this was the 2nd time calling it. exp: RetryError[Put
...

```

正在执行 (已持续 5 分 24 秒) Python 3

1.gemini-2.5-flash's result

## 2. gemini-2.5-pro's result

```

print("Generated password: [password]")

if args.copy:
 try:
 ClipboardUtils.copy_to_clipboard(password)
 print("Password copied to clipboard.")
 except RuntimeError as exc:
 print(f"Warning: {exc}")
 sys.exit(2)

if __name__ == "__main__":
 Main.run()

2026-02-26 18:49:41.963 | INFO | metagpt.utils.cost_manager:update_cost|57 - Total running cost: $0.220 | Max budget: $3.000 | Current cost: $0.037, prompt_tokens: 1890, completion_tokens: 596
2026-02-26 18:49:41.966 | INFO | metagpt.actions.write_code_review:run|185 - Code review and rewrite main.py: 1/2 | len(iterative_code)-3047, len(self.i_context.code_doc_content)-3047
Code Review Result
1. Yes, the code is implemented as per the requirements. It provides a command-line interface for password generation, uses argparse for argument parsing, and integrates PasswordGenerator and ClipboardUtils as specified.
2. Yes, the code logic is correct. It properly handles user input, error cases, and program flow, including clipboard operations and password generation.
3. Yes, the code follows the "Data structures and interfaces" described in the system design. The Main class encapsulates CLI handling and program flow, and uses the other modules as intended.
4. Yes, all functions are implemented. The parse_args and run methods are present and fully functional.
5. Yes, all necessary pre-dependencies are imported. argparse, sys, password_generator, and clipboard_utils are imported as required.
6. Yes, methods from other files are reused correctly. PasswordGenerator and ClipboardUtils are used as per their interfaces.

Actions
pass

Code Review Result
LGM

2026-02-26 18:49:44.119 | INFO | metagpt.utils.cost_manager:update_cost|57 - Total running cost: $0.255 | Max budget: $3.000 | Current cost: $0.034, prompt_tokens: 2835, completion_tokens: 200
2026-02-26 18:49:44.122 | INFO | metagpt.utils.file_repository:save|57 - save to: /content/workspace/simple_password_generator/simple_password_generator/main.py
2026-02-26 18:49:44.125 | INFO | metagpt.utils.file_repository:save|62 - update dependency: /content/workspace/simple_password_generator/simple_password_generator/main.py|`docs/system_design/20260226184921.json`>
2026-02-26 18:49:44.139 | INFO | metagpt.roles.engineer:act_summarize|190 --max-auto-summarize-code=0
2026-02-26 18:49:44.139 | WARNING | metagpt.environment.base_env.publish_message|192 - Message no recipients: ["id":2274136681604ebc94bd769f1837b0d63,"content":"","role":"Engineer","cause_by":"metagpt.action"
2026-02-26 18:49:44.145 | INFO | metagpt.utils.git_repository:archive|168 - Archive: `['dependencies.json', 'docs/prd/20260226184921.json', 'docs/requirement.txt', 'docs/system_design/20260226184921.json']`>
☒ 任务完成：耗时: 33.88 秒

```

### 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.