



# Code Review for Sprint 2



Review Standard



Number of Reviews



The Participants of Reviews



Review Examples

Example 1 - Crawler function

Example 2 - Rag framework

Example 3 - Test cases



Learn From the AI Code Review



## Review Standard [↗](#)

- **Review Scope:** The focus of this review will primarily be on the main functional components of Sprint 2, including the crawler function and RAG preliminary framework. Additionally, we will review code related to test cases to ensure comprehensive test coverage and code quality.
- **Review Method:** We will utilize automated code review and AI techniques for code review. After receiving feedback from the AI, we will use a combination of human evaluations and peer reviews within the development team to determine the level of adoption of AI reviews. Our review standard will be based on the following criteria:
  - **Readability:** whether the code is clearly commented, whether it is descriptive of variables and functions, and whether the code structure is clear and reasonable.
  - **Consistency:** Check that the code follows the team's agreed code style guidelines, including indentation, use of parentheses, naming conventions, etc. Follow the same data structure and processing logic for the way data is processed and stored.
  - **Efficiency:** Evaluate the algorithmic complexity in the code. Check whether the code effectively uses system resources, avoids unnecessary loops, reduces double calculations, etc., to improve the execution efficiency of the code.
  - **Robustness:** Check that the code contains appropriate exception handling mechanisms to deal with possible error situations, to ensure that the code can catch exceptions and give appropriate feedback or handling methods.



## Number of Reviews [↗](#)

We have chosen to conduct code reviews during each merge of the development branch into the main branch to ensure that code quality issues are detected and resolved in a timely manner, to prevent problems from expanding in subsequent development phases, and to minimize potential technical issues and risks. In addition, the frequency of code reviews can be affected by project progress and changes in requirements, so we maintain some flexibility to adjust the number and content of reviews as needed.

In total, we plan to conduct **three code reviews**:

1. The first review will focus on our crawler implementation, with a primary focus on crawler correctness, reliability, and efficiency.
2. The second review focused on the core functionality of our main framework to ensure the functional stability, performance and scalability of the core framework.
3. The third review will focus on our test cases, mainly to ensure that the test code is reasonable and does not affect the core framework due to the wrong conclusions of the test cases.




## The Participants of Reviews [↗](#)

We are divided into three code reviews based on the development function, and each review will be composed of technical lead, developer, and tester. At each review session, developers are required to be involved in the review process. Each review meeting will involve at least two reviewers. The responsibilities of the participants are divided as follows:

- **Technical Lead**

- Responsible for reviewing overall code quality and architectural design.
- Provide technical guidance and advice.
- Ensure code meets the team's technical standards
- **Developer**
  - Responsible for developing the code for the corresponding function.
  - Participate in the review process, explain and discuss your code implementation.
  - Provide feedback and suggestions on other developers' code.
- **Tester**
  - Responsible for writing test cases and performing functional tests.
  - Provide feedback on test coverage and quality during the review process.
  - Ensure code meets expected functional requirements and quality standards.

## Review Examples

 Due to problems with our github actions, we will use chatgpt directly, combined with the following prompt.

```

1 Please evaluate the {code} below.
2
3 Use the following checklist to guide your analysis:
4
5 Documentation Defects: a. Naming: Assess the quality of software element names. b. Comment: Analyze the quality
6 Visual Representation Defects: a. Bracket Usage: Identify any issues with incorrect or missing brackets. b. Inde
7 Structure Defects: a. Dead Code: Find any code statements that serve no meaningful purpose. b. Duplication: Iden
8 New Functionality: a. Use Standard Method: Determine if a standardized approach should be used for single-purpos
9 Resource Defects: a. Variable Initialization: Identify variables that are uninitialized or incorrectly initializ
10 Check Defects: a. Check User Input: Analyze the validity of user input and its handling.
11 Interface Defects: a. Parameter: Detect incorrect or missing parameters when calling functions or libraries.
12 Logic Defects: a. Compute: Identify incorrect logic during system execution. b. Performance: Evaluate the effici
13
14 Provide your feedback in a numbered list for each category. At the end of your answer, summarize the recommended

```

### Example 1 - Crawler function

#### Code Source

[Github Link - src/url\\_to\\_text.py](#)

#### AI Code Review Responded





- Documentation Defects: a. Naming: The names of the functions and variables are descriptive and follow Python naming conventions. b. Comment: The code lacks inline comments, which could clarify the purpose of certain sections, especially the dynamic content handling.
- Visual Representation Defects: a. Bracket Usage: No issues found with bracket usage. b. Indentation: The code is consistently indented properly. c. Long Line: The line where the print statement is located could be split for better readability.
- Structure Defects: a. Dead Code: There are no obvious instances of dead code. b. Duplication: The code to fetch and handle dynamic content could be refactored to avoid duplication.
- New Functionality: a. Use Standard Method: The code follows standard practices for web scraping and text formatting.
- Resource Defects: a. Variable Initialization: All variables are properly initialized. b. Memory Management: Memory management seems appropriate for the task. The driver instance is properly closed after use.
- Check Defects: a. Check User Input: The code does not handle user input directly, as it receives the URL as input. However, it does not perform any validation on the URL.
- Interface Defects: a. Parameter: No incorrect or missing parameters when calling functions or libraries.

- Logic Defects: a. Compute: The logic seems sound for the intended functionality. b. Performance: The code may face performance issues due to iterating over potentially a large number of dynamic divs. Additionally, the use of `time.sleep(2)` could be considered instead of relying on a fixed delay.

Recommended changes to improve code quality:

- Add inline comments to clarify the purpose of each section, especially the dynamic content handling.
- Refactor the code to avoid duplication in handling dynamic content.
- Implement input validation for the URL parameter.
- Consider optimizing performance, especially in handling dynamic content and waiting for page load.
- Split the long line containing the print statement for better readability.

#### Adopt or not

- Add inline comments 
- Refactor the code to avoid duplication in handling dynamic content. 
- validation for the URL 
- Split the long line containing the print statement 

#### Reason

1. We will add inline comments, which will make it easier for us and other readers to understand code blocks and functions.
2. There was an additional manual review and discussion of refactoring the code to avoid duplication when working with dynamic content. Since the module is related to RAG core functions, in order to ensure the delivery of the current overall system, we will not change it for the time being, but we will consider its suggestions in the next sprint.
3. For the URL validation part, we did not fully consider in the development of the project. The input URL may not be accessible, URL input errors and other problems caused by the system crash, so we decided to adopt the review suggestion, add a new URL verification module.
4. Our current print statement is just to test the integrity of the functionality. Because it does not affect the functionality, so we will not adopt this recommendation.

## Example 2 - Rag framework

### Original Code

[Github Link - src/RagFramework.py](#)

[Github Link - src/demo.py](#)

[Github Link - src/chatbot.py](#)

### AI Code Review Responded

1. Documentation Defects:
  - a. Naming: The naming convention for functions and variables seems reasonable, though `RagFramework.py` should perhaps be named `rag_framework.py` to adhere to Python conventions.
  - b. Comment: The comments provided are minimal but sufficient to understand the purpose of each function and class.
2. Visual Representation Defects:
  - a. Bracket Usage: The bracket usage appears to be correct throughout the code.
  - b. Indentation: Indentation is consistent and follows the Python standard.
  - c. Long Line: There are a few long lines in the code, particularly in `RagFramework.py`, which might affect readability. Consider breaking them down or refactoring where necessary.
3. Structure Defects:
  - a. Dead Code: There doesn't appear to be any dead code in the provided snippets.
  - b. Duplication: Some code segments, especially related to text splitting and retrieval, are duplicated across different functions. Refactoring these segments into reusable functions could improve code maintainability.

#### 4. New Functionality:

- Use Standard Method: The code follows standard Python practices for the most part. However, ensuring consistency in function parameter naming and coding style could further enhance readability.

#### 5. Resource Defects:

- Variable Initialization: Variables seem to be initialized where necessary. However, it's essential to ensure proper initialization and handling of variables, especially in larger codebases.
- Memory Management: The code doesn't explicitly deal with memory management, which might not be a concern at this scale. However, for larger applications, ensuring efficient memory usage and proper management would be crucial.

#### 6. Check Defects:

- Check User Input: User input checking is limited, especially in the `main()` function. Adding robust input validation mechanisms can improve the robustness of the program.

#### 7. Interface Defects:

- Parameter: Parameters seem to be appropriately passed to functions and libraries.





#### 8. Logic Defects:

- Compute: The logic seems sound, but thorough testing is necessary to identify any potential issues.
- Performance: The performance might be impacted by the usage of external APIs and web scraping. Optimizing these operations and caching results where possible could enhance performance.

To improve the quality of the code:

- Refactor duplicated code segments into reusable functions or methods.
- Enhance input validation mechanisms, especially in functions that interact with external resources.
- Consider breaking down long lines of code for improved readability.
- Ensure consistency in naming conventions and coding style.
- Conduct thorough testing, especially focusing on edge cases and error handling.
- Optimize performance by minimizing redundant operations and optimizing resource-intensive tasks.

#### Adopt or not

- Refactor duplicated code segments 
- Enhance input validation 
- breaking down long lines of code 
- Ensure consistency in naming 

#### Reason

- At present, many duplicate blocks of code retain interfaces and functions, which are convenient for subsequent calls and tests.
- The current development projects that need to be delivered are only presented using the specific website specified. However, multiple input cases need to be considered in the future, so we will adopt this article review of it.
- Considering the functional integrity of the whole system, we should not adopt the measure of excessively long code lines splitting for the time being. This is because such changes can make development more difficult and potentially affect the functionality of the system.
- Since the code content is being worked on by different developers at the same time, naming and development styles may differ, requiring manual review and communication by developers to ensure consistency.

#### Example 3 - Test cases

##### Original Code

[Github Link - test/](#)

##### AI Code Review Responded

#### 1. Documentation Defects:

- Naming: Variable names like `rag_model`, `docs`, `question`, etc., are descriptive and follow Python naming conventions, which is

good.

b. Comment: The code lacks inline comments explaining complex logic or rationale behind certain decisions. Comments could be added to explain the purpose of functions, especially in `answer_generator.py`.

## 2. Visual Representation Defects:

a. Bracket Usage: Brackets seem to be used correctly throughout the code.

b. Indentation: Indentation looks consistent and follows the Python standard.

c. Long Line: There are a few lines exceeding the recommended 79 characters, particularly in `answer_generator.py`. It's advisable to break them down for better readability.

## 3. Structure Defects:

a. Dead Code: No dead code seems to be present in the provided code.

b. Duplication: Some code segments for handling conversations in `answer_generator.py` seem duplicated and could be refactored for better maintainability.

## 4. New Functionality:

a. Use Standard Method: The code generally follows standard Python practices. However, some parts, especially in `answer_generator.py`, could be structured more conventionally to improve readability.

## 5. Resource Defects:

a. Variable Initialization: All variables seem to be correctly initialized before use.

b. Memory Management: The code doesn't have explicit memory management issues, but efficiency could be improved in some areas, especially in `answer_generator.py`.

## 6. Check Defects:

a. Check User Input: The code doesn't perform extensive validation of user input. It might be beneficial to add error handling for cases where input data is missing or malformed.

## 7. Interface Defects:

a. Parameter: Function parameters seem appropriate and correctly used.

## 8. Logic Defects:





a. Compute: There don't appear to be any obvious logic errors, but thorough testing would be necessary to ensure correctness.

b. Performance: Performance could potentially be improved, especially in `answer_generator.py`, where multiple message transformations are applied.

To summarize, here are some recommended changes to improve the code quality:

- Add inline comments to explain complex logic or rationale behind certain decisions.
- Refactor duplicated code segments in `answer_generator.py` for better maintainability.
- Break down long lines of code for improved readability, especially in `answer_generator.py`.
- Implement more robust error handling for user input validation.
- Consider optimizing performance, especially in `answer_generator.py`, where multiple message transformations are applied.

## Adopt or not

- Add inline comments 
- Refactor duplicated code segments in `answer_generator.py` for better maintainability. Consider optimizing performance 
- Break down long lines of code 
- User input validation 

## Reason

1. We will add inline comments, which will make it easier for us and other readers to understand code blocks and functions.
2. `answer_generator.py` is the function we used for testing, and it may be called repeatedly, so we'll keep it
3. Considering the functional integrity of the whole system, we should not adopt the measure of excessively long code lines splitting for the time being. This is because such changes can make development more difficult and potentially affect the functionality of the system.

4. Because we are using test cases as input in the current testing phase, we are not considering input issues. Therefore, in our core system and subsequent development, it is necessary for us to pay attention to the multiple possibilities of user input to ensure that the system does not crash due to input problems.

## 👉 Learn From the AI Code Review [↗](#)

The utilization of automated AI code review tools has provided us with an opportunity to deploy and explore the capabilities of such tools. Furthermore, it has relieved us of the arduous task of manually reviewing code while reinforcing the significance of code review. Through this experience, we have recognized the importance of uncovering potential issues that are often overlooked, thus addressing bugs at their source.

### **Pros:**

- Automation of code review reduces the workload associated with human review.
- The feedback provided by AI, particularly when leveraging LLM technology, tends to be accurate and comprehensive.
- The tool's ability to identify nuanced details between lines of code, which might otherwise lead to program crashes when overlooked.

### **Cons:**

- Despite leveraging sophisticated techniques, such as large language model (LLM) technology, these tools may still struggle to comprehend the broader context and business logic underlying specific code segments, occasionally resulting in misinterpretations or overinterpretations.
- Because the feedback from the technology is not 100% correct, it still needs human review of the AI's evaluation conclusions.
- AI code reviews may ignore the realities of the development team, such as the technical capabilities, style, and tool differences of the developers.