



Code Review for Sprint 3



Review Standard



Number of Reviews



The Participants of Reviews



Review Examples

Example 1 - Mutil-modal RAG framework

Example 2 - Images Crawler Function

Example 3 - Furhat robot Function



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 3, including the images crawler function, mutil-modal RAG framework and the final product integrates the interactive capabilities of furhat robot.
- **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. Images crawler function
2. Mutil-modal RAG framework
3. Furhat robot function



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

Example 1 - Mutil-modal RAG framework

Source Code

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

AI Code Review Responded

Documentation Defects:

- a. Naming: The variable names used in the code are descriptive and relevant to their purpose. No issues found in this aspect.
- b. Comment: The code contains comments that explain the purpose of functions and prompts. However, more detailed comments could be added to explain the logic within the functions for better understanding.

Visual Representation Defects:

- a. Bracket Usage: The code uses brackets appropriately, and there are no issues with missing or incorrect brackets.
- b. Indentation: The code follows consistent indentation, which enhances readability.
- c. Long Line: Some lines of code are quite long, which may hinder readability. Consider breaking them into multiple lines or adding line breaks for better readability.

Structure Defects:

- a. Dead Code: No dead code is present in the provided code.
- b. Duplication: The code does not contain duplicate code statements that need refactoring.

New Functionality:

- a. Use Standard Method: The code follows a structured approach with functions for different tasks, which is a good practice for maintainability and reusability.

Resource Defects:

- a. Variable Initialization: All variables are initialized before use, ensuring proper handling of data.
- b. Memory Management: The code does not show any explicit memory management issues.

Check Defects:

- a. Check User Input: The code does not perform explicit validation or handling of user input. Consider adding input validation to ensure data integrity.

Interface Defects:

- a. Parameter: The function parameters are used appropriately when calling functions or libraries.




Logic Defects:

- a. Compute: The logic in the code seems to be logically sound based on the provided context.
- b. Performance: The efficiency of the algorithm used cannot be evaluated without further context on the specific requirements and constraints.

Recommended Changes:

1. Add more detailed comments within the functions to explain the logic and flow of operations.
2. Consider breaking long lines of code into multiple lines for improved readability.
3. Implement input validation to ensure the integrity of user input.
4. Review the code for potential optimizations based on specific performance requirements.

Adopt or not

- Add more detailed comments within the functions
 - Adopt: 
 - Reason: Adding more detailed comments will improve code readability and help other developers understand the purpose and logic of the code more easily. It ensures better maintainability and easier debugging.
- Consider breaking long lines of code into multiple lines
 - Adopt: 
 - Reason: Although breaking long lines can improve readability, it does not affect the overall functionality of the current code. Whether to break long sentences also depends on the developer's ability and the specific scenario of the code, so it is not adopted.
- Implement input validation
 - Adopt: 
 - Reason: The input of the current framework is only used for testing and will not affect the final presentation of the project, so it will not be adopted.

Example 2 - Images Crawler Function

Source Code

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

AI Code Review Responded

Documentation Defects:

- a. **Naming:** The variable names used in the code are descriptive and relevant to their purpose. No issues found in this aspect.
- b. **Comment:** The code lacks sufficient comments explaining the logic, especially within the loop and the regex functions. Adding comments to describe the purpose of each block and the regex matching logic would improve readability and maintainability.

Visual Representation Defects:

- a. **Bracket Usage:** The code uses brackets appropriately, and there are no issues with missing or incorrect brackets.
- b. **Indentation:** The code follows consistent indentation, which enhances readability.
- c. **Long Line:** There are no excessively long lines that hinder readability. The lines are well within a reasonable length.

Structure Defects:

- a. **Dead Code:** No dead code is present in the provided code.
- b. **Duplication:** The code checks for image sources in a loop with different attributes. This could be refactored to reduce repetition.

New Functionality:

- a. **Use Standard Method:** The approach used is standard for web scraping. However, it could benefit from a more structured error handling mechanism.

Resource Defects:

- a. **Variable Initialization:** All variables are initialized before use, ensuring proper handling of data.
- b. **Memory Management:** The code manages memory well by quitting the WebDriver after use.

Check Defects:

- a. **Check User Input:** The code does not validate the input URL. Adding validation to ensure it's a valid URL would be beneficial.

Interface Defects:

a. **Parameter:** The function parameters are used appropriately when calling functions or libraries.

Logic Defects:

a. **Compute:** The logic for extracting URLs and handling different image source attributes is correct. However, the URL construction logic may fail if the image URL is an absolute path without the domain. Using `urljoin` from `urllib.parse` would be more reliable.

b. **Performance:** The performance of the algorithm is reasonable. However, the WebDriver initialization and quitting could be optimized if the function is called multiple times.






Recommended Changes:

1. Add more detailed comments within the functions to explain the logic and flow of operations.
2. Refactor the loop that checks different attributes to reduce redundancy.
3. Implement input validation to ensure the integrity of user input.
4. Use `urljoin` from `urllib.parse` to handle URL construction more reliably.
5. Implement a more structured error handling mechanism to manage potential exceptions during WebDriver operations.

Here's the refactored code with recommended improvements:

These changes improve the clarity, reliability, and maintainability of the code.

Adopt or not

- Add more detailed comments
 - Adopt: 
 - Reason: Adding more detailed comments will improve code readability and help other developers understand the purpose and logic of the code more easily. It ensures better maintainability and easier debugging.
- Refactor the loop
 - Adopt: 
 - Reason: There may be a loop instruction structure in the crawler. This is because we need to continuously visit certain specific websites for functional requirements. We need to keep the loop structure for functionality, so we decided not to adopt its suggestion.
- Implement input validation
 - Adopt: 
 - Reason: The input of the current framework is only used for testing and will not affect the final presentation of the project, so it will not be adopted.
- handle URL construction more reliably
 - Adopt: 
 - Reason: If the current method for constructing URLs is reliable and not prone to errors, changing it may not be necessary. However, if there's any indication that URL construction could fail or be improved, it should be addressed.
- manage potential exceptions during WebDriver operations
 - Adopt: 
 - Exception management is crucial for robust code. If WebDriver operations are already stable and there's a low risk of encountering exceptions, additional exception handling might not be needed.

Example 3 - Furhat robot Function

Source Code

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

AI Code Review Responded

Documentation Defects:

- a. **Naming:** The variable names used in the code are generally descriptive and relevant to their purpose. However, some names could be more specific for better clarity. For example, `judge_type` could be renamed to `determine_question_type`, and `rag_chain` could be renamed to `question_type_chain` and `query_generation_chain` in their respective contexts.
- b. **Comment:** The code contains some comments, but they are sparse and do not explain the logic within the functions. Adding more detailed comments would help in understanding the purpose and flow of operations within the functions.

Visual Representation Defects:

- a. **Bracket Usage:** The code uses brackets appropriately, and there are no issues with missing or incorrect brackets.
- b. **Indentation:** The code follows consistent indentation, which enhances readability.
- c. **Long Line:** Some lines of code are quite long, which may hinder readability. Consider breaking them into multiple lines or adding line breaks for better readability.

Structure Defects:

- a. **Dead Code:** There are sections of commented-out code under `# '''load from db'''` that appear to be old or unused. Remove these sections or clarify their purpose if they are intended for future use.
- b. **Duplication:** The code contains some duplication in functions like `save_images` and `load_images`. These functions perform related tasks and could be refactored to share common code. Similarly, the logic for invoking the LLM with a prompt in `judge_type` and `generate_based_history_query` could be abstracted into a common helper function.

New Functionality:

- a. **Use Standard Method:** The code follows a structured approach with functions for different tasks, which is a good practice for maintainability and reusability. However, for tasks like question classification in `judge_type`, consider using a standardized library or approach if available.

Resource Defects:

- a. **Variable Initialization:** All variables are initialized before use, ensuring proper handling of data. However, there should be a check to ensure `image_urls` and `image_ls` are not empty before proceeding with functions like `save_images`.
- b. **Memory Management:** The code does not show any explicit memory management issues, but ensure that large objects like images and vectors are deleted when no longer needed to optimize memory usage.

Check Defects:

- a. **Check User Input:** The code does not perform explicit validation or handling of user input. The `listen_to_user` function listens indefinitely without a timeout or user input validation. Consider adding a timeout and validation to ensure the input is sensible.

Interface Defects:

- a. **Parameter:** The function parameters are used appropriately when calling functions or libraries, but consider adding type hints for better clarity.

Logic Defects:






- a. **Compute:** The logic in the code seems to be logically sound based on the provided context. Ensure that functions like `qa_based_image_retrieval` handle cases where no relevant images or texts are found.
- b. **Performance:** The efficiency of the algorithm used is generally acceptable, but the infinite loop in `listen_to_user` could be optimized or bounded to avoid potential performance issues.

Recommended Changes:

1. Add more detailed comments within the functions to explain the logic and flow of operations.
2. Consider breaking long lines of code into multiple lines for improved readability.
3. Implement input validation to ensure the integrity of user input.
4. Refactor duplicated code into common helper functions to improve maintainability.
5. Remove or clarify commented-out sections that serve no current purpose.

6. Review the code for potential optimizations based on specific performance requirements.
7. Use type hints for function parameters and return types for better clarity and readability.
8. Ensure efficient memory usage by deleting large objects when not needed.

Adopt or not

- Add more detailed comments
 - Adopt: 
 - Reason: Adding more detailed comments will improve code readability and help other developers understand the purpose and logic of the code more easily. It ensures better maintainability and easier debugging.
- Consider breaking long lines
 - Adopt: 
 - Reason: Although breaking long lines can improve readability, it does not affect the overall functionality of the current code. Whether to break long sentences also depends on the developer's ability and the specific scenario of the code, so it is not adopted.
- Implement input validation
 - Adopt: 
 - Reason: Due to the application and demand limitations of the robot itself, our input method is only to monitor the voice through the robot's SDK, so there are no multiple input methods, so it does not affect our system
- Refactor duplicated code
 - Adopt: 
 - Reason: For the current sprint, it is mainly used for final project presentation and reporting to customers. It is necessary to delete some previous duplicate and irrelevant codes.
- Remove or clarify commented-out sections
 - Adopt: 
 - Reason: For the current sprint, it is mainly used for final project presentation and reporting to customers. It is necessary to delete some previous repeated and irrelevant comments to make the content clearer when the code is delivered and evaluated.

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.

Based on the code review of the current sprint, we found that the main problems are the lack of sufficient comments, redundant comments or repeated code blocks, and the lack of sufficient validation for input and URLs. However, this may also be affected by the current project requirements and does not affect the robustness of the overall system.

Pros:

- A clear structure allows us to understand the deficiencies in code logic and structure.
- There are some repeated and similar code review suggestions, indicating that these problems are easy to occur and be ignored.
- Provided positive suggestions for our subsequent code delivery

Cons:

- There may be a lack of understanding of project requirements and code structure, which may cause some of the suggestions to be problematic.
- If the current code review does not find problems that affect logic and functionality, most of the suggestions it makes are mainly optimization-oriented and require human consideration and decision-making.

