

AI Enabled IDE Project Paper

By Samyak Navad

Mentored by Foaad Khosmood

https://github.com/samyak-n/PyzoAIRevamp/tree/AI_IDE

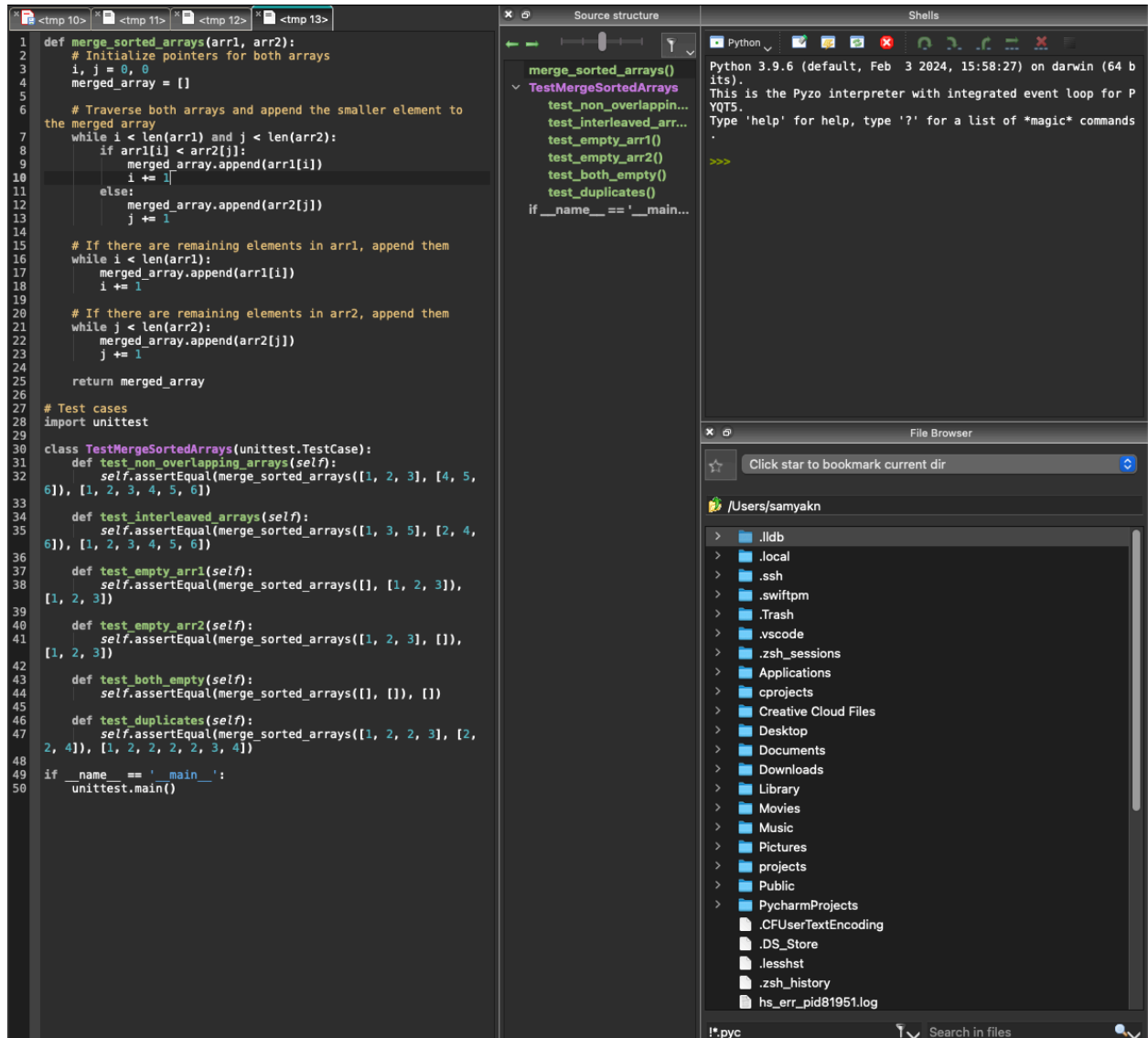


Table of Contents

Introduction.....	2
Background/Tools.....	5
Related Work.....	7
System Development.....	9
Results.....	17
Conclusion.....	26
Future Directions.....	28
Bibliography.....	29

Introduction

In the evolving landscape of software development, the integration of Artificial Intelligence (AI) into development tools has become increasingly important. This project proposes the enhancement of the Pyzo IDE, a simple yet powerful open-source tool for Python development, by incorporating AI capabilities from the ground-up to create a more robust, intelligent, and user-friendly development environment in a simple and strategic way.

Objective

The goal is to develop an AI-enhanced version of Pyzo IDE that provides intelligent coding assistance, leveraging AI models like GPT-4 and AI Agent architecture.

The vision for the AI-enhanced Pyzo IDE is to revolutionize the Python programming experience by seamlessly integrating advanced AI capabilities into the IDE itself. This integration aims to not only change the coding process but also to transform Pyzo into a more intelligent, interactive, and intuitive tool that caters to both educational and professional development needs. The envisioned AI features are designed to be more than just tools; they are to act as a companion to the developer, providing insights, suggestions, and learning aids that adapt to individual coding styles and preferences.

This vision is rooted in my personal belief that the future of software development lies in the synergy between human creativity and AI efficiency. By embedding AI into Pyzo, we aim to create an environment where developers, students, and educators can achieve more with less effort, making Python programming more accessible, enjoyable, and productive.

With this vision in mind, the project will focus on developing an AI-enhanced version of Pyzo IDE that not only retains the simplicity and user-friendliness of the original IDE but also introduces incredibly beneficial AI functionalities. This balance will ensure that Pyzo remains a preferred tool for Python development, both for those new to programming and for experienced developers seeking a more efficient workflow.

Project Scope

- **AI Integration:** Implement GPT-4o or 3.5 AI Agents for code completion, bug detection, and test development.
- **Loop Integration:** Agents rely on each other for outputs, to form a pipeline that creates a full program.
- **Testing Requirements:** Users input two major test cases they want their program to be built around.
- **Documentation & Tutorials:** Create comprehensive guides for using AI features.
- **Testing & Feedback:** Conduct testing and ensure a robust final product.

Key Features

- Intelligent Code Completion: Fully detached system for correct code completion.
- Error Detection & Suggestions: Real-time error detection through AI generated corrective suggestions in the pipeline.
- Agent Driven Approach: Various AI agents work together in a pipeline to create a working final product.
- Robust Testing: User submits two key test cases around which the program is built.
- Seamless Integration: AI features integrated into Pyzo's core functionality.

Technology Stack

- IDE Framework: Pyzo IDE (<https://github.com/pyzo/pyzo/tree/main>)
- AI Models: OpenAI GPT, GPT Agents
- Programming Languages: Python
- Communication: JSON

AI Agents (need to refresh diagrams)

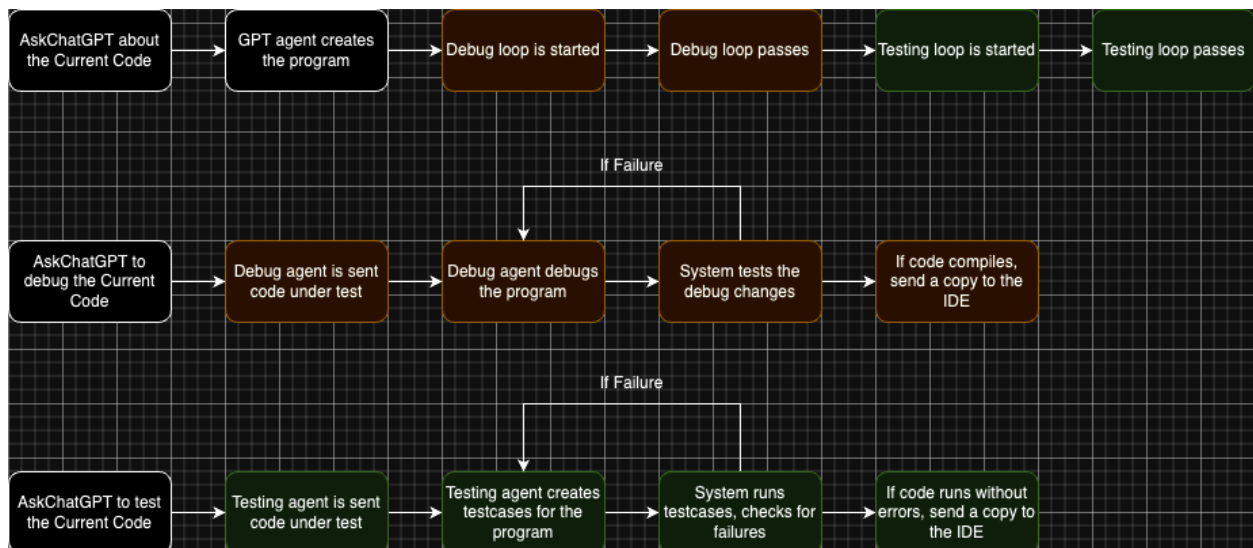


Fig 1.1: A [comprehensive diagram](#) focusing on the interaction and data flow between the AI engine and the UI.

System Architecture

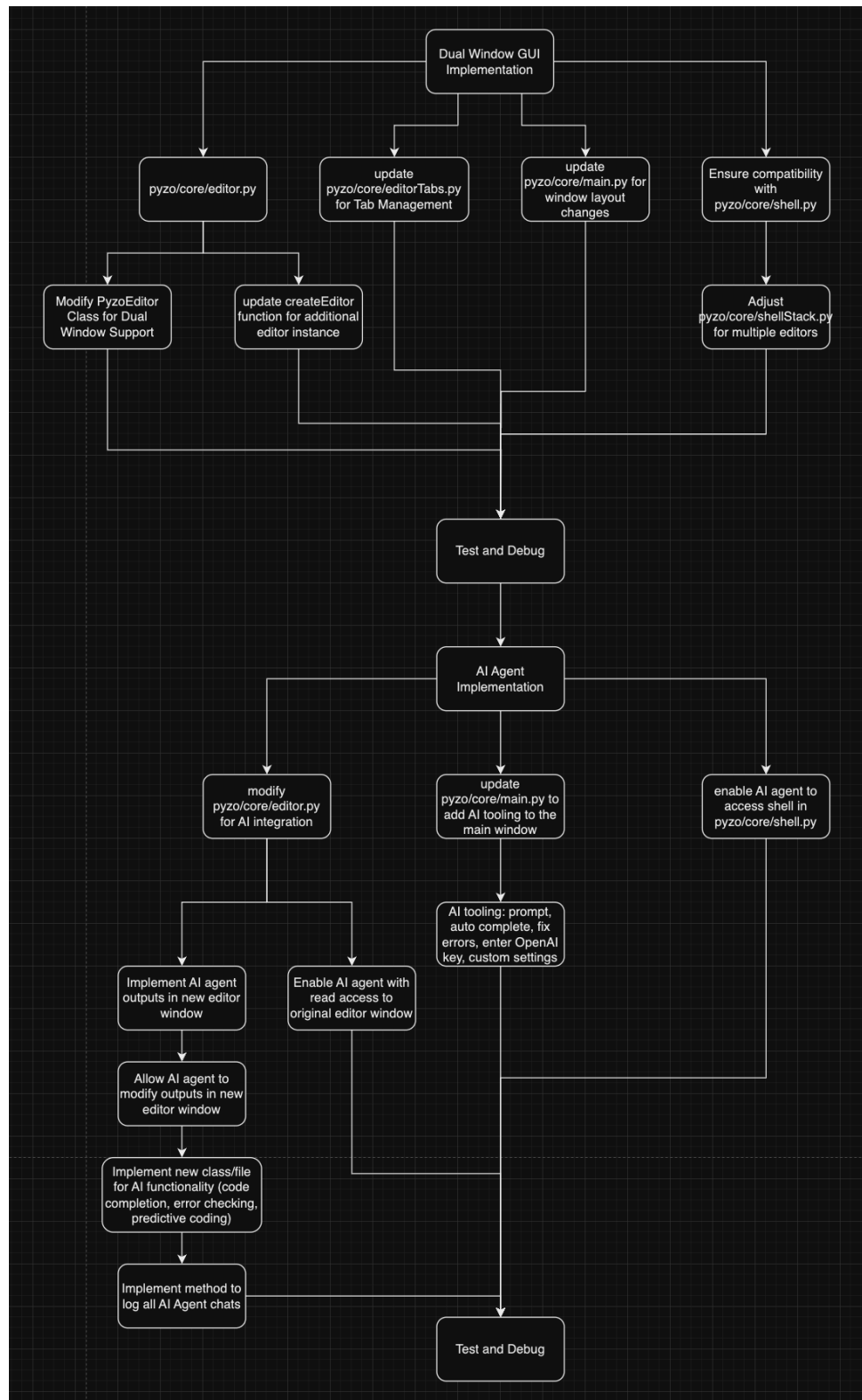


Fig 1.2: High-level [architecture diagram](#) showing the integration of AI components within the Pyzo IDE.

Background / Tools

Knowledge Requirements

To develop an AI-enhanced Integrated Development Environment (IDE) like the proposed AI-enabled Pyzo IDE, a deep understanding of various theories, algorithms, and tools is essential. This section outlines the specific knowledge areas and technical skills necessary to achieve the project's goals.

Pyzo IDE Existing Framework

Pyzo is a user-friendly Python Integrated Development Environment (IDE) specifically designed with educational purposes in mind. Its architecture is characterized by a clear hierarchical structure, promoting ease of navigation and understanding, which is particularly beneficial for community driven updates. The IDE's modular design not only facilitates the incorporation of new features and tools but also promotes adaptability to evolving educational needs and technological advancements. Pyzo encourages contributions from its user base, allowing it to continuously evolve through collaborative efforts. This approach ensures that the IDE remains up-to-date with the latest Python trends and educational methodologies, making it a dynamic tool for learning and teaching programming. Furthermore, Pyzo has strong cross-platform compatibility despite being a smaller project. It works across MacOS (Arm & Intel), Windows, and Linux systems. Lastly, Pyzo's ease of installation and setup is a significant advantage. It requires minimal configuration, enabling users to quickly get started with their Python projects. We want to carry this ease of setup and configuration over to the AI integrated version as well.

Artificial Intelligence (AI)

1. Natural Language Processing: Packages, Tools and Research
 - a. Transformer Models: Understanding the architecture and functioning of transformer models, particularly OpenAI's GPT-4o, which will be used for generating code completions and suggestions.
 - b. Language Modeling: Knowledge of how language models are trained and fine-tuned for specific tasks such as code completion and error detection.
 - c. Prompt Engineering: Crafting effective prompts to maximize the utility of GPT-4 in providing relevant and contextually appropriate coding assistance.
 - d. Agent Architecture: Creating a functioning pipeline in which agents feed each other information to create a robust final product.

Software Development and IDEs

1. Python Programming
 - a. Advanced Python: Proficiency in Python, including advanced features like connections, context managers, GUI implementations, and asynchronous programming, to enhance the Pyzo IDE.

- b. Python Libraries: Familiarity with libraries such as NumPy, pandas, scikit-learn, OpenAI, requests, and PyQt5, which are often used in conjunction with AI models, API requests, and graphical user interfaces (GUI).
- 2. Integrated Development Environments (IDEs)
 - a. IDE Architecture: Understanding the internal architecture of IDEs, focusing on Pyzo, to facilitate seamless integration of AI components.
 - b. User Interface Design: Principles of designing a user-friendly and intuitive interface that can incorporate complex AI functionalities without overwhelming the user.

Fundamental Problem Overview

The primary challenge is to enhance the Python programming experience by integrating AI capabilities into the Pyzo IDE. This involves:

1. Improving Code Quality and Productivity
 - a. Implementing AI-driven code completions, debugging, and testing to help developers write better code more efficiently.
 - b. Error detection and correction to minimize bugs and improve code quality in the AI agent pipeline responses.
2. Enhancing User Experience
 - a. Designing an intuitive and interactive user interface that integrates AI features seamlessly within the IDE.
 - b. Ensuring users have the ability to create not just the code that they want, but code that passes their own provided tests.
3. Ensuring Customizability and Flexibility
 - a. Providing options for users to pick what specific AI features they would like to use.
 - b. Supporting both beginners and experienced developers with tailored AI functionalities.
4. Maintaining Simplicity and Accessibility
 - a. Retaining the ease of use and minimal configuration that Pyzo is known for.
 - b. Making advanced AI capabilities accessible to a wide range of users, from students to professionals.
 - c. By addressing these challenges, the AI-enhanced Pyzo IDE aims to create a more efficient, enjoyable, and productive development environment, revolutionizing the Python programming experience for all users.

Related Work

1. Natural Language Processing (NLP), Code Generation, and Multi Agent Processes

Recent advancements in NLP, particularly with models like OpenAI's GPT-3 and GPT-4, have significantly impacted the field of software development. These models have been utilized for code generation, code completion, and error detection, transforming the development process.

Key Papers:

Title	Authors	Year	Key Contributions
Communicative Agents for Software Development	Chen Qian et al.	2023	Introduces ChatDev, a framework leveraging GPT models for collaborative software development. Demonstrates the potential of LLMs in automating and enhancing coding tasks.
Experiential Co-Learning of Software-Developing Agents	Chen Qian et al.	2023	Explores co-learning techniques where AI agents learn from each other's experiences, improving their problem-solving capabilities over time.
Iterative Experience Refinement of Software-Developing Agents	Chen Qian et al.	2024	Introduces Iterative Experience Refinement (IER), enhancing AI agents' efficiency in adapting to new tasks through experience propagation.

These papers highlight how AI models can be integrated into an IDE to provide intelligent coding assistance, such as real-time code development and AI based testing. By leveraging some of these techniques, the AI-enhanced Pyzo IDE aims to provide robust coding support.

Relation to Our Work:

We integrated GPT-4o agents for intelligent code completion and error detection, inspired by the frameworks discussed in these papers. By leveraging the agent based approach, we create a pipeline that iteratively improves its product and ensures fundamentally correct generated code.

- Inspired by ChatDev's multi-agent structure, the AI-enhanced Pyzo IDE incorporates various AI agents (coding agents, testing agents, and debugging agents) to collaboratively assist the developer.

- This approach ensures a comprehensive support system within the IDE, enhancing overall efficiency and code quality.

2. Integrated Development Environments and User Experience

Recent research focuses on improving the user experience within IDEs through better interface design and enhanced functionalities.

Key Papers:

Title	Authors	Year	Key Contributions
Visual Studio IntelliCode: AI-Assisted Development	Microsoft	2019	Describes the integration of AI into Visual Studio, providing intelligent code suggestions and improving developer productivity.
Toward a Better IDE: Lessons Learned from Improving IntelliJ IDEA	JetBrains	2020	Discusses improvements in IntelliJ IDEA, focusing on user experience and integrating intelligent features.

These papers provide insights into how AI can be effectively integrated into an IDE to enhance the development experience by providing contextual assistance and improving code quality.

Relation to Our Work:

The AI-enhanced Pyzo IDE draws on these examples to design a user-friendly interface that incorporates AI features seamlessly. We focus on maintaining the simplicity and accessibility of Pyzo while adding powerful AI functionalities, similar to the enhancements seen in Visual Studio and IntelliJ IDEA.

By integrating the insights and methodologies from these diverse areas of research, the AI-enhanced Pyzo IDE aims to create a revolutionary development environment. The key ideas and techniques from NLP, IDE enhancements, and multi-agent systems are synthesized to provide an intelligent, intuitive, and efficient coding assistant that caters to the needs of both novice and experienced developers.

System Development

1. OpenAI API Integration

In the AI-enhanced Pyzo IDE project, one of the key components is the integration of OpenAI's API to provide intelligent code suggestions and error detection. This section details the development and functionality of the `call_openai_api` method, which communicates with the OpenAI API, and discusses the design decisions made during its implementation. **In order for this to work, you must provide your OpenAI API key in the function itself.**

a. Function Overview

The `call_openai_api` method is responsible for sending user content to the OpenAI API and receiving a response. This method was designed to ensure robust and efficient communication with the API, handling various potential issues such as rate limiting and network errors.

1. Content Preparation:

- The method starts by preparing the content to be sent to the OpenAI API. The user's input content is appended with a directive to "Always use codeblocks and python." This ensures that the response from the API is formatted appropriately for coding purposes.

2. API Endpoint and Authentication:

- The API URL and API key are specified. The API key should be securely stored and not hard-coded in production applications. The headers are prepared to include the authorization token and content type.

3. Request Data:

- The data payload includes the model name (`gpt-3.5-turbo/gpt-4o`), the message content, and the temperature setting, which controls the creativity of the response. The model can be changed by the user at any time by changing this logic.

4. Retry Logic:

- A retry mechanism is implemented to handle potential issues such as rate limiting and transient network errors. The method attempts the request up to five times, with specific handling for HTTP errors and general exceptions.
- Rate Limiting: If the API returns a status code 429 (rate limit), the method waits for 5 seconds before retrying.
- Blank Responses: If a blank response is received, the method retries after a 2-second delay.

5. Response Handling:

- Upon receiving a valid response, the method extracts the message content and emits a `chatRequest` signal to update the IDE interface. This integration ensures that the user receives immediate feedback from the AI.

6. Error Handling:

- Comprehensive error handling ensures that various exceptions are caught and appropriately managed, providing informative messages to the user and avoiding application crashes.

b. Design Decisions

Below are some design decisions that I made while creating this function. This function has the main functionality for the entire application, so it has to be robust, reliable, and efficient.

1. **Content Formatting:** Ensuring the response includes code blocks and Python-specific content was crucial for maintaining context and readability within the IDE. This decision enhances the usability of the AI-generated suggestions.
2. **Robustness and Reliability:** Implementing retry logic improves the robustness of the method, ensuring that temporary issues do not disrupt the user experience. Handling rate limits and other HTTP errors gracefully is essential for a smooth integration.
3. **User Feedback:** Emitting signals to update the IDE interface provides immediate visual feedback to the user, making the interaction with the AI seamless and intuitive.
4. **Security Considerations:** Although the API key is hard-coded here for demonstration, in a real application, it would be securely stored and managed using environment variables or secure vaults to prevent unauthorized access.

Overall, the `call_openai_api` method is a cornerstone of our AI-enhanced Pyzo IDE, enabling intelligent code suggestions and error detection. The careful design and robust implementation of this method ensure that users receive high-quality assistance, enhancing their coding experience.

2. Agent Pipeline Integration

The agent logic pipeline in our AI-enhanced Pyzo IDE project involves several key components designed to automate coding assistance, debugging, and unit test generation. This section provides a detailed breakdown of the agent logic functions, focusing on the `askChat`, `askChatUnitTests`, and `askChatDebug` methods. Each function is designed to interact with the OpenAI API and process the responses to enhance the developer's experience.

a. askChat Method Overview

The `askChat` method initiates the interaction with the OpenAI API based on user-selected code or the entire content of the editor. It processes the response to execute the code and handle potential errors.

1. Text Cursor and User Input:

- The method retrieves the current text cursor from the editor. If there is selected text, it uses that as the user input; otherwise, it uses the entire content of the editor.
- This flexibility ensures that users can request AI assistance on specific code snippets or their entire script.

2. Content Preparation and API Call:

- The method appends a prompt to the user input, requesting the OpenAI API to generate responses with specific requirements (e.g., including test cases).
- It then calls the `call_openai_api` method to send the content to the OpenAI API and receive a response.

3. Processing API Response:

- Using a regular expression pattern, the method extracts Python code blocks from the API response.
- The extracted code blocks are concatenated and prepared for execution.

4. Code Execution and Error Handling:

- The method checks if the code contains input statements, which would require user interaction during execution.
- If no input is required, it attempts to execute the code using a subprocess. If execution fails, it retries by calling a debugging method (`askChatDebug`) to refine the code based on errors encountered.

5. Handling Successful Execution:

- If the code executes successfully, the method prints the output and proceeds to generate unit tests using the `askChatUnitTests` method.

b. askChatUnitTests Method Overview

The `askChatUnitTests` method generates and executes unit tests for the given Python code, ensuring comprehensive test coverage and robustness.

1. Code Selection:

- Depending on the flag value, the method selects either the currently highlighted code in the editor or the entire script.

2. Unit Test Request:

- The method prepares a prompt requesting the OpenAI API to generate unit tests for the selected code.
- It calls the `call_openai_api` method to obtain the unit tests from the OpenAI API.

3. Executing Unit Tests:

- The generated unit tests are extracted from the API response and executed using a subprocess.
- The method captures the output and errors, checking for test failures.

4. Handling Test Failures:

- If there are test failures, the method enters a retry loop, invoking the `askChatDebug` method to debug and refine the tests.
- The process continues until the tests pass without errors, ensuring the reliability of the generated tests.

c. `askChatDebug` Method Overview

The `askChatDebug` method assists in debugging the Python code by leveraging the OpenAI API to identify and resolve errors.

- 1. Code Selection:**
 - Similar to the other methods, it selects either the currently highlighted code or the entire script for debugging.
- 2. Debugging Request:**
 - The method prepares a prompt asking the OpenAI API to help debug the code, including any error messages encountered during execution.
 - It calls the `call_openai_api` method to receive debugging assistance from the API.
- 3. Refining Code:**
 - The method processes the API response to extract the debugged code and attempts to re-execute it.
 - This iterative approach ensures that errors are systematically identified and resolved, enhancing the code's robustness.

d. Design Decisions

- 1. User Interaction and Flexibility:**
 - Allowing users to select specific code snippets or use the entire script provides flexibility and control over the AI assistance. This design decision ensures that the tool can cater to various coding scenarios and user preferences.
- 2. Robust Error Handling:**
 - Implementing retry logic and error handling mechanisms ensures that transient issues (such as rate limiting) do not disrupt the user experience. The methods are designed to handle multiple attempts and provide informative feedback, enhancing reliability.
- 3. Automated Debugging and Testing:**
 - Integrating automated debugging and unit test generation reduces the manual effort required for code verification and error resolution. This approach leverages the AI's capabilities to streamline the development process and improve code quality.
- 4. Comprehensive Feedback Loop:**
 - The feedback loop between the `askChat`, `askChatUnitTests`, and `askChatDebug` methods creates a robust system for refining and validating

code. By continuously improving based on errors and test results, the system ensures high-quality outputs.

Overall, the agent logic pipeline is a critical component of the AI-enhanced Pyzo IDE, providing intelligent coding assistance, automated debugging, and comprehensive unit testing. The thoughtful design and robust implementation ensure that users receive high-quality support, enhancing their coding experience and productivity.

3. Seamless Frontend Integration

To ensure that the AI-enhanced features are accessible directly from the Pyzo IDE's interface, we seamlessly integrated the new functionality into the front end of the IDE. This integration was achieved by adding context menu options that allow users to interact with the AI for various coding tasks such as asking for code explanations, debugging, and creating unit tests.

a. Editor Context Menu Integration

The `EditorContextMenu` class defines the context menu for the editor, where we added new items to interact with the AI. Below is the implementation of how these integrations were performed.

Menu.py

```
self.addItem(
    translate("menu", "Ask ChatGPT about the current Code"),
    None,
    self._editItemCallback,
    "askChat",
)
self.addItem(
    translate("menu", "Ask ChatGPT to debug the current Code"),
    None,
    self._editItemCallback,
    "askChatDebug",
)
self.addItem(
    translate("menu", "Ask ChatGPT to create tests for the current Code"),
    None,
    self._editItemCallback,
    "askChatUnitTests",
)
```

Fig 1.3: Context Menu Logic

b. Adding New Menu Items

We added three new items to the context menu, each corresponding to a specific AI-assisted functionality:

1. **Ask ChatGPT about the current Code (begins the overall pipeline)**
2. **Ask ChatGPT to debug the current Code (starts the debug pipeline)**
3. **Ask ChatGPT to create tests for the current Code (starts the testing pipeline)**

These items were added using the `addItem` method within the `build` method of the `EditorContextMenu` class.

Integration Details

- **Consistency:** The new menu items were added in a manner consistent with the existing menu structure, ensuring a seamless user experience.
- **Accessibility:** Placing the new options within the context menu makes them easily accessible to users, facilitating quick interaction with the AI.
- **Functionality Linking:** Each new menu item is linked to the corresponding method (`askChat`, `askChatDebug`, `askChatUnitTests`) using the `_editItemCallback`. This ensures that the appropriate function is called when the user selects an option from the context menu.

c. Real Time IDE GUI Updates

In the AI-enhanced Pyzo IDE, one of the significant features is the ability to receive real-time updates from the AI and reflect those changes directly in the editor. This functionality was implemented by connecting backend actions to the GUI frontend using the `connect()` function, enabling seamless integration of AI responses into the IDE.

```
chatRequest = QtCore.Signal(str)
```

Fig 1.4: EditorTabs.py connection logic

newFileChat Method Overview

The `newFileChat` method handles the creation of a new file in the editor and populates it with content received from the AI.

1. **Editor Creation:**
 - A new editor instance is created using `createEditor`, and its modification state is initially set to `False`.
2. **Tab Management:**
 - A `FileItem` instance is created to represent the new file in the tab widget.

- The new file item is added to the tab widget, and the tab is set to the current item, bringing it into focus.
3. **Content Processing:**
 - A regular expression pattern is used to extract Python code blocks from the received message from the chatRequest signal.
 - The code blocks are joined and set as the content of the new editor instance.
 4. **Focus Management:**
 - The new editor is brought into focus, ensuring that the user can immediately start interacting with the newly created file.

d. Design Decisions

1. **Consistency:**
 - a. The new AI-assisted functionalities were integrated into the context menu in a manner consistent with existing menu items. This consistency ensures that users can intuitively find and use the new features without needing to learn a new interface.
2. **Accessibility:**
 - a. Placing the new options directly within the context menu made them easily accessible to users. This design choice allows users to quickly invoke AI functionalities such as code explanations, debugging, and unit test generation with minimal effort.
3. **Functionality Linking:**
 - a. Each new menu item was linked to its corresponding method (`askChat`, `askChatDebug`, `askChatUnitTests`) using the `_editItemCallback` function. This ensured that selecting a menu option would trigger the appropriate backend function, providing the intended AI assistance.
4. **Real-Time Communication:**
 - a. We used the `connect()` function to link the `chatRequest` signal to the `newFileChat` slot, allowing for real-time communication between the backend (where AI responses are generated) and the frontend (where these responses are displayed). This mechanism ensures that AI-generated responses are immediately reflected in the editor, providing users with real-time assistance.
5. **Dynamic Content Update:**
 - a. The `newFileChat` method dynamically creates new editor tabs and populates them with AI-generated content. This approach ensures that users can immediately interact with the AI-generated code or suggestions, enhancing their coding experience.
6. **Editor Creation and Management:**
 - a. When a new file is created in response to an AI message, an editor instance is created using `createEditor`. This new editor is then added to the tab widget, ensuring that it is immediately available for user interaction. Bringing the new

editor into focus immediately after creation ensures that users can start interacting with the AI-generated content without any additional steps.

This setup ensured that any response from the AI was immediately reflected in the editor, providing users with real-time assistance and enhancing their coding experience. The seamless connection between backend actions and frontend updates was critical for maintaining an intuitive and responsive interface.

Interesting Problems and Solutions

1. Handling Dynamic Input:

- One challenge was managing code that required user input during execution. By detecting input statements and handling them separately, we ensured that the system could execute code without requiring real-time user interaction.

2. Robust Error Recovery:

- Implementing a retry mechanism with detailed error messages allowed the system to recover from various errors effectively. This design ensured that users received meaningful feedback and the opportunity to refine their code iteratively.

3. Efficient API Utilization:

- Ensuring efficient and effective communication with the OpenAI API was crucial. By carefully crafting prompts and processing responses, we maximized the utility of the API and provided valuable assistance to the user.

4. Handling Rate Limits:

- One of the interesting challenges was handling API rate limits effectively. By implementing a retry mechanism with exponential backoff, we ensured that the application could recover from rate limit errors without excessive delay.

5. Ensuring Response Quality:

- Another challenge was ensuring the quality and relevance of the responses. By appending specific instructions to the user input, we guided the AI to produce responses that are more suitable for integration into the IDE.

6. Error Handling and User Feedback:

- Providing detailed error messages and implementing retry logic were critical to maintaining a smooth user experience. This approach minimized downtime and ensured that users were informed about the status of their requests.

7. Frontend and GUI Integration

- One of the more challenging problems was connecting the actions from the AI backend to the GUI frontend. This required using the `connect()` function to ensure that interactions between the backend logic and the frontend interface were seamless and responsive. This approach allowed us to update the GUI dynamically, ensuring that user interactions with the context menu items triggered the appropriate backend functions and that the results were displayed promptly.

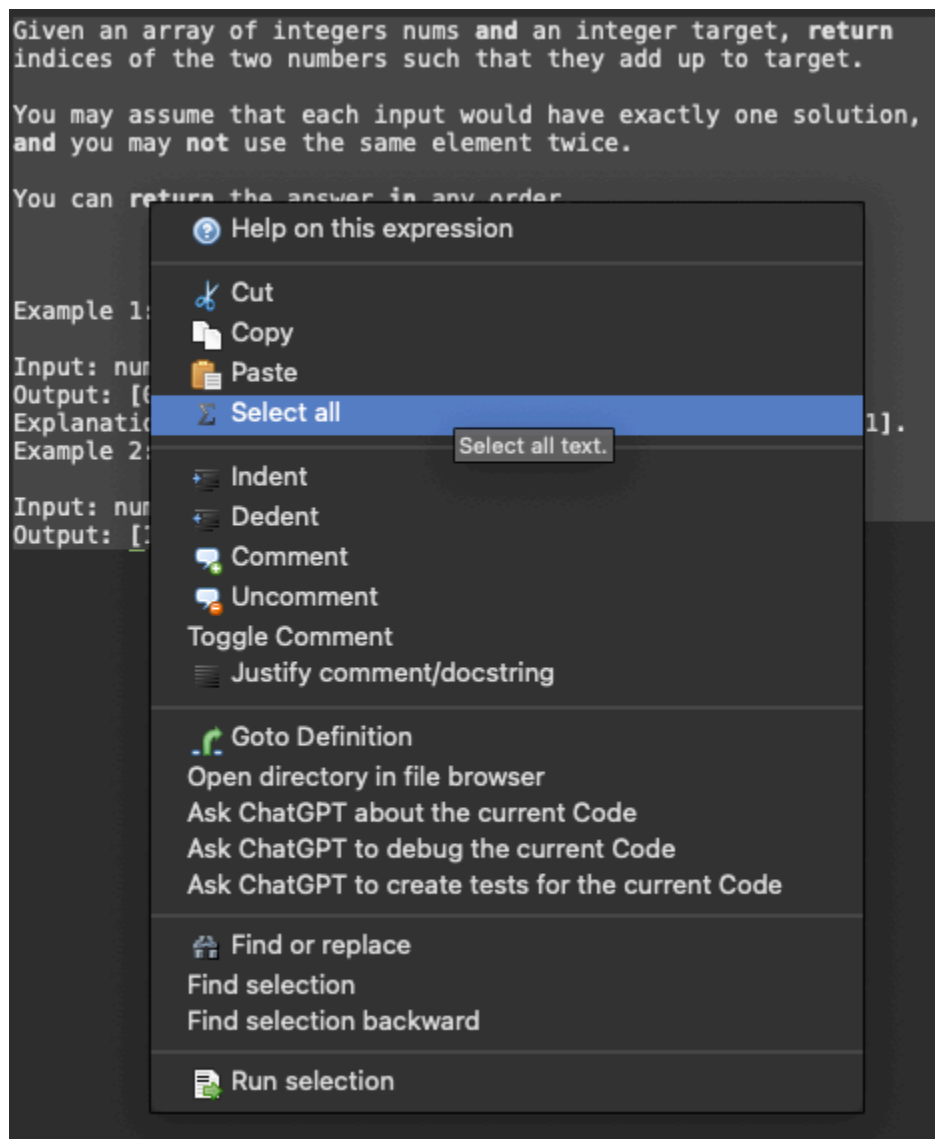
Results

1. User Interaction, System Interface, and Walkthrough

The AI-enhanced Pyzo IDE introduces several powerful features designed to streamline the coding process and enhance the developer's experience. This section provides an overview of how users interact with the system and highlights the key functionalities that make the AI integration effective and user-friendly.

a. User Interaction

Users interact with the AI-enhanced Pyzo IDE primarily through the context menu integrated into the editor. This menu provides options for AI-assisted coding tasks such as asking for code explanations, debugging, and creating unit tests. The seamless integration ensures that users can quickly access these features without disrupting their workflow.



1. Context Menu Integration:

- a. When users right-click within the editor, the context menu appears, displaying new options for interacting with the AI:
 - i. **Ask ChatGPT about the current Code:** This option initiates the pipeline for obtaining code explanations.
 - ii. **Ask ChatGPT to debug the current Code:** This option starts the debugging pipeline.
 - iii. **Ask ChatGPT to create tests for the current Code:** This option triggers the test generation pipeline.

b. System Interface

The system interface was designed to be intuitive and user-friendly, ensuring that AI-generated responses are seamlessly integrated into the user's coding environment.

1. Real-Time Updates

- a. When a user selects an AI option from the context menu, the system processes the request and updates the editor in real-time. For example, selecting "Ask ChatGPT about the current Code" would:
 - i. **Trigger the Backend Process:** The `askChat` method is called, sending the user-selected code to the AI for processing. The AI response is received and processed to extract relevant code blocks.
 - ii. **Update the Editor:** The `newFileChat` method is invoked, creating a new file in the editor populated with the AI-generated content. The new file is immediately brought into focus, allowing the user to start interacting with it.

c. Functionality Results and Walkthrough

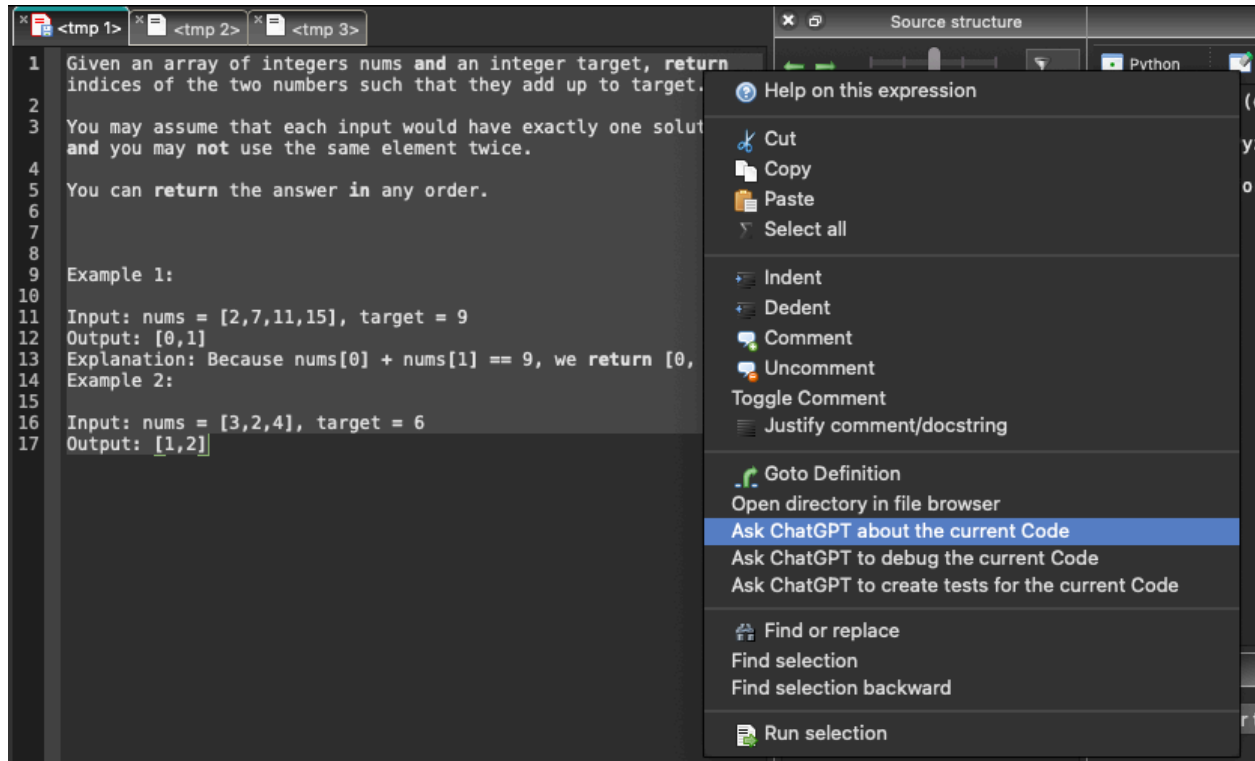
1. Ask ChatGPT About the Current Code

- a. The pipeline begins with the `askChat` method and GPT agents begin working on the functionality of the program.
- b. After each version of the program is completed, we run the program to check for syntax and logical errors with the `askChatDebug` method.
- c. After the code is debugged, we call `askChatUnitTests` to create tests for the program (both based on user input and some basic generated test cases).

Example: In this example, we use a simple LeetCode problem “twoSum” (difficulty - easy) to showcase the capability of the application through its **askChat** pipeline.

1. Initial API Call

In this image, we are calling the main function **askChat** to begin the pipeline and send the instructions for “twoSum” to a GPT agent.



2. Code Generation

In the next image, we are receiving an untested version of the code that has been debugged using the `askChatDebug` method. The next step is to send this code to the `askChatUnitTests` method.

```
<tmp 1> <tmp 2> <tmp 3>
1 def two_sum(nums, target):
2     # Dictionary to store the complement and its index
3     num_to_index = {}
4
5     # Iterate through the list of numbers
6     for index, num in enumerate(nums):
7         # Calculate the complement needed to reach the target
8         complement = target - num
9
10        # Check if the complement is already in the dictionary
11        if complement in num_to_index:
12            # If found, return the indices
13            return [num_to_index[complement], index]
14
15        # If not found, add the number and its index to the
16        # dictionary
17        num_to_index[num] = index
18
19        # If no solution is found, return an empty list (though the
20        # problem guarantees one solution)
21        return []
22
23    # Example usage:
24    nums1 = [2, 7, 11, 15]
25    target1 = 9
26    print(two_sum(nums1, target1)) # Output: [0, 1]
27
28    nums2 = [3, 2, 4]
29    target2 = 6
30    print(two_sum(nums2, target2)) # Output: [1, 2]
```

3. Test Generation

In the final image, we receive a tested version of our code that has been run by the system and vetted for any errors. This final output is the solution to the initial problem request, verified and tested by the debug and testing agents.

```
<tmp 1> <tmp 2> <tmp 3>
1 def two_sum(nums, target):
2     # Dictionary to store the complement and its index
3     num_to_index = {}
4
5     # Iterate through the list of numbers
6     for index, num in enumerate(nums):
7         # Calculate the complement needed to reach the target
8         complement = target - num
9
10        # Check if the complement is already in the dictionary
11        if complement in num_to_index:
12            # If found, return the indices
13            return [num_to_index[complement], index]
14
15        # If not found, add the number and its index to the
16        # dictionary
17        num_to_index[num] = index
18
19        # If no solution is found, return an empty list (though the
20        # problem guarantees one solution)
21        return []
22
23 # Unit tests
24 import unittest
25
26 class TestTwoSum(unittest.TestCase):
27     def test_case_1(self):
28         nums = [2, 7, 11, 15]
29         target = 9
30         result = two_sum(nums, target)
31         self.assertEqual(result, [0, 1])
32
33     def test_case_2(self):
34         nums = [3, 2, 4]
35         target = 6
36         result = two_sum(nums, target)
37         self.assertEqual(result, [1, 2])
38
39     def test_case_3(self):
40         nums = [3, 3]
41         target = 6
42         result = two_sum(nums, target)
43         self.assertEqual(result, [0, 1])
44
45     def test_case_4(self):
46         nums = [1, 2, 3, 4, 5]
47         target = 9
48         result = two_sum(nums, target)
49         self.assertEqual(result, [3, 4])
50
51 if __name__ == '__main__':
52     unittest.main()
```

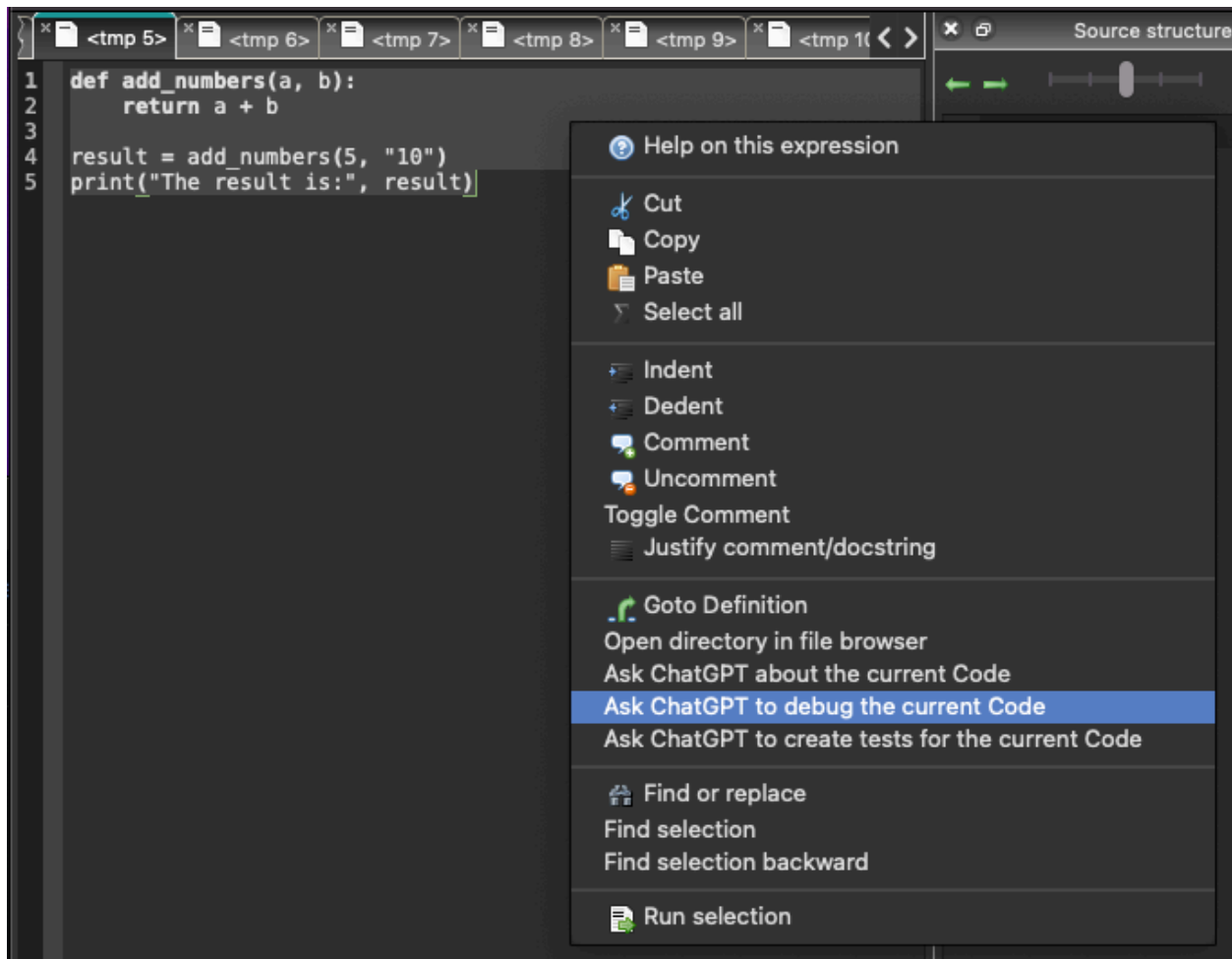
2. Ask ChatGPT to Debug the Current Code:

- d. The pipeline begins with the `askChatDebug` method and GPT agents begin working on debugging the program.
- e. After each version of the program is completed, we run the program to check for syntax and logical errors with the `askChatDebug` method.
- f. After the code is debugged, we create a new file and put the debugged code into the editor.

Example: In this example, we use a simple program that has an error to showcase the capability of the application through its `askChatDebug` pipeline.

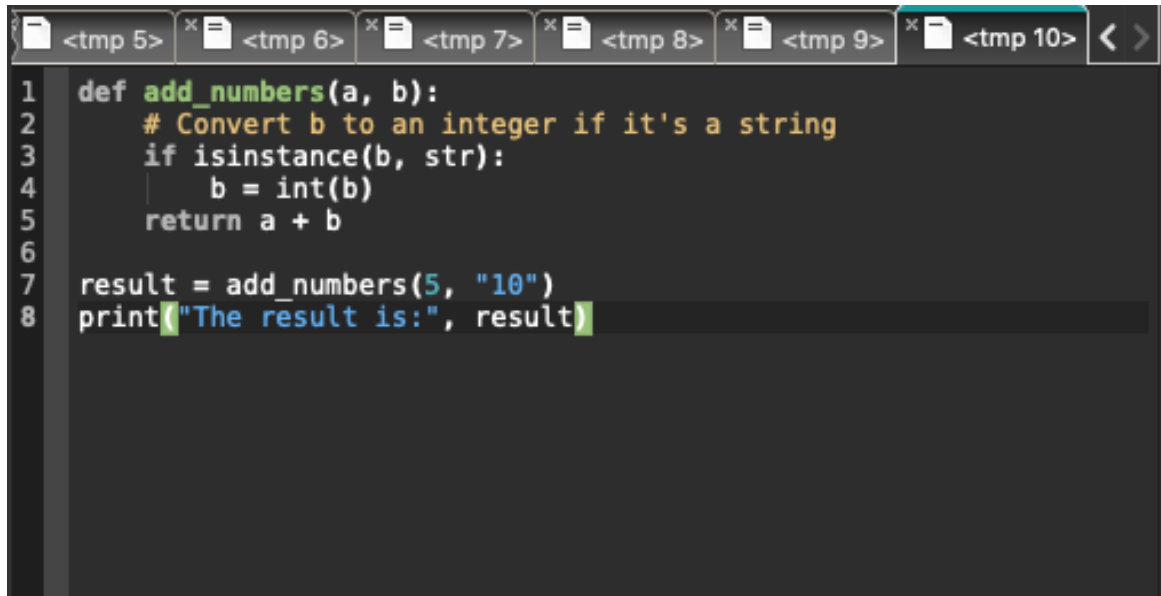
1. Initial API Call

In this image, we are calling the main function `askChatDebug` to begin the pipeline and send the code under test to GPT.



2. Post Debug

In the final image, the debugged code is pasted to a new file in the IDE.



The screenshot shows an IDE window with a tab bar at the top containing seven tabs labeled <tmp 5>, <tmp 6>, <tmp 7>, <tmp 8>, <tmp 9>, and <tmp 10>. The <tmp 10> tab is selected and highlighted in blue. The main editor area displays the following Python code:

```
1 def add_numbers(a, b):
2     # Convert b to an integer if it's a string
3     if isinstance(b, str):
4         b = int(b)
5     return a + b
6
7 result = add_numbers(5, "10")
8 print("The result is:", result)
```

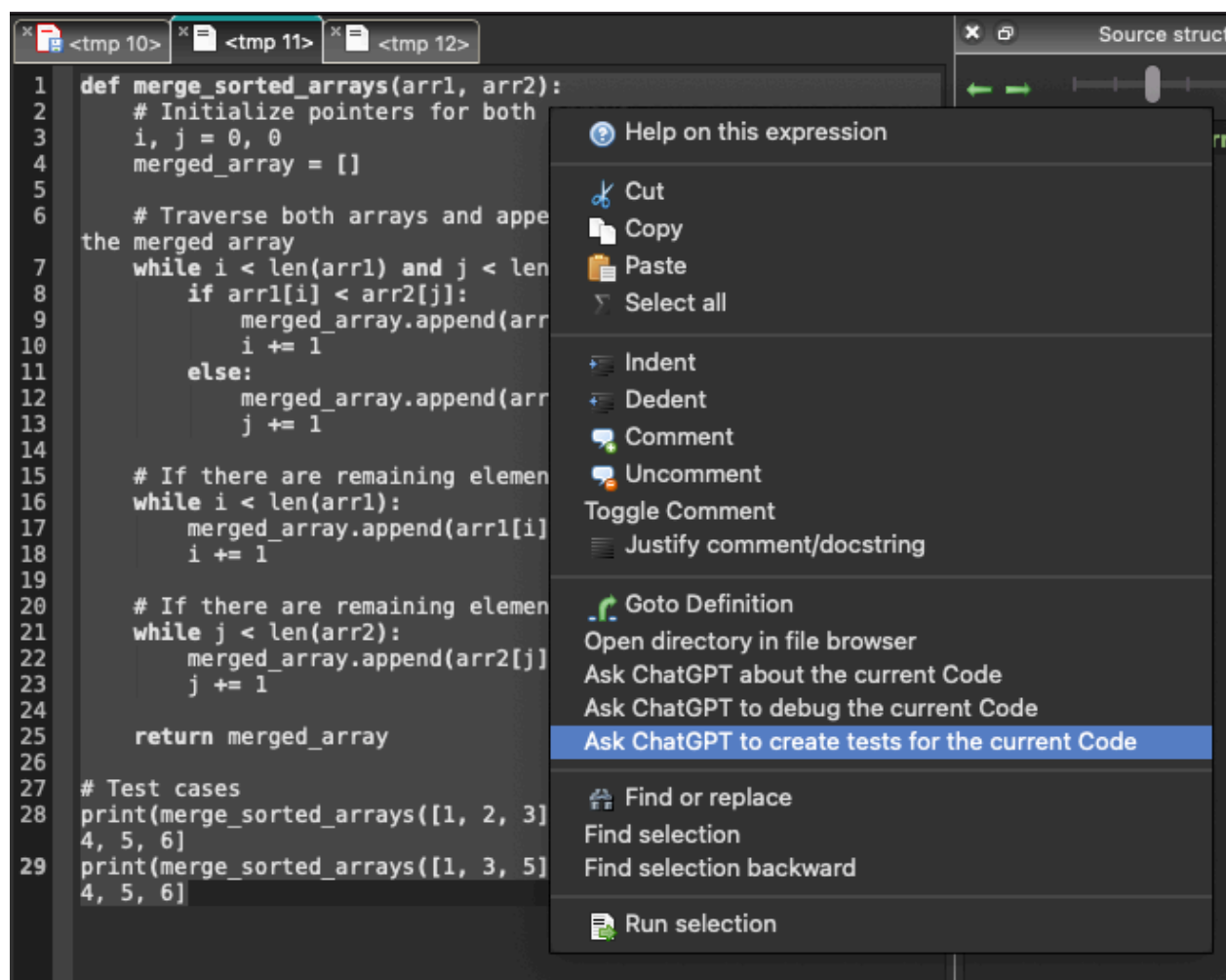

3. Ask ChatGPT to Test the Current Code

- The pipeline begins with the `askChatUnitTests` method and GPT agents begin working on creating tests for the program.
- After each version of the program is completed, we run the program to check for syntax and logical errors with the `askChatDebug` method.
- We then run the tests in the IDE to verify that all tests are passing.
- After the code is debugged and the tests are verified, we create a new file and put the debugged, tested code into the editor.

Example: In this example, we use a program that needs tests written to showcase the capability of the application through its `askChatUnitTests` pipeline.

1. Initial API Call

In this image, we are calling the main function `askChatUnitTests` to begin the pipeline and send the code under test to GPT.



2. Tests Created, and Processed

In this image, we receive the output from the GPT agents that have the code under test and all the test cases written for it.

```
<tmp 10> <tmp 11> <tmp 12> <tmp 13>
1 def merge_sorted_arrays(arr1, arr2):
2     # Initialize pointers for both arrays
3     i, j = 0, 0
4     merged_array = []
5
6     # Traverse both arrays and append the smaller element to
7     the merged array
8     while i < len(arr1) and j < len(arr2):
9         if arr1[i] < arr2[j]:
10             merged_array.append(arr1[i])
11             i += 1
12         else:
13             merged_array.append(arr2[j])
14             j += 1
15
16     # If there are remaining elements in arr1, append them
17     while i < len(arr1):
18         merged_array.append(arr1[i])
19         i += 1
20
21     # If there are remaining elements in arr2, append them
22     while j < len(arr2):
23         merged_array.append(arr2[j])
24         j += 1
25
26     return merged_array
27
28 # Test cases
29 import unittest
30 class TestMergeSortedArrays(unittest.TestCase):
31     def test_non_overlapping_arrays(self):
32         self.assertEqual(merge_sorted_arrays([1, 2, 3], [4, 5,
33         6]), [1, 2, 3, 4, 5, 6])
34
35     def test_interleaved_arrays(self):
36         self.assertEqual(merge_sorted_arrays([1, 3, 5], [2, 4,
37         6]), [1, 2, 3, 4, 5, 6])
38
39     def test_empty_arr1(self):
40         self.assertEqual(merge_sorted_arrays([], [1, 2, 3]),
41         [1, 2, 3])
42
43     def test_empty_arr2(self):
44         self.assertEqual(merge_sorted_arrays([1, 2, 3], []),
45         [1, 2, 3])
46
47     def test_both_empty(self):
48         self.assertEqual(merge_sorted_arrays([], []), [])
49
50     def test_duplicates(self):
51         self.assertEqual(merge_sorted_arrays([1, 2, 2, 3], [2,
52         2, 4]), [1, 2, 2, 2, 2, 3, 4])
53
54 if __name__ == '__main__':
55     unittest.main()
```

Conclusion

1. Lessons Learned

Throughout the development of the AI-enhanced Pyzo IDE, several key lessons were learned, contributing to a deeper understanding of integrating AI into software development tools.

a. The Power of AI in Development:

- **Intelligent Assistance:** AI models like GPT-4o demonstrated their capability to provide intelligent code suggestions, debugging assistance, and test generation, significantly enhancing developer productivity and code quality.
- **Versatility and Adaptability:** The AI agents were able to adapt to various coding scenarios, providing relevant and contextually appropriate assistance, which highlighted the versatility of AI in handling different programming tasks.

b. User-Centric Design:

- **Ease of Use:** Maintaining the simplicity and user-friendliness of the Pyzo IDE while integrating advanced AI functionalities was crucial. The project reinforced the importance of designing tools that are intuitive and accessible to users of all skill levels.
- **Feedback Mechanisms:** Implementing real-time feedback mechanisms through seamless frontend integration ensured that users received immediate and actionable insights, improving their overall coding experience.

c. Technical Challenges and Solutions:

- **Handling API Rate Limits:** One of the challenges encountered was managing rate limits imposed by the OpenAI API. Implementing a robust retry mechanism with exponential backoff ensured that the application could recover from rate limit errors without excessive delays.
- **Ensuring Response Quality:** Crafting effective prompts and processing responses to ensure high-quality and relevant AI-generated code was another critical aspect. This involved iterative testing and refining prompts to guide the AI in producing suitable outputs.

2. Shortcomings

a. Limited Testing and Validation:

- **Real-World Testing:** Due to time constraints, the project did not undergo extensive real-world testing with a diverse user base. This limited the ability to gather comprehensive feedback on the AI-enhanced features and their impact on various coding environments.
- **Comprehensive Evaluation:** While the system demonstrated promising results in controlled tests, a more thorough evaluation involving different programming tasks and real-world scenarios with multiple testing parties is necessary to fully validate the effectiveness and reliability of the AI enhancements.

b. Dependency on External APIs:

- **Reliability on OpenAI API:** The project's reliance on the OpenAI API for generating code suggestions and debugging assistance introduced potential vulnerabilities related to API availability, performance, and rate limiting. In the future, it would be best to create our own model to control how it responds and works behind the scenes.

Future Directions

1. Extensive User Testing and Feedback:

- Conduct comprehensive testing with a diverse user base to gather detailed feedback on the AI-enhanced features.
- Iterate on the design and functionality based on user insights to improve usability and effectiveness.

2. Enhanced Integration and Performance:

- Refine the integration between the AI backend and frontend interface to ensure seamless and real-time updates.
- Optimize performance to handle larger codebases and more complex programming tasks efficiently.

3. Expanded AI Capabilities:

- Explore additional AI models and techniques to enhance code suggestions, debugging, and testing capabilities.
- Integrate more advanced features such as code refactoring, performance optimization suggestions, and automated documentation generation.

By addressing these areas, the AI-enhanced Pyzo IDE can continue to evolve into a more robust and comprehensive tool, providing unparalleled assistance to developers and revolutionizing the Python programming experience.

Bibliography

Chen, Q., Cong, X., Liu, W., Yang, C., Chen, W., Su, Y., Dang, Y., Li, J., Xu, J., Li, D., Liu, Z., & Sun, M. (2023). Communicative Agents for Software Development. arXiv preprint arXiv:2307.07924. Retrieved from <https://arxiv.org/abs/2307.07924>

Chen, Q., Dang, Y., Li, J., Liu, W., Chen, W., Yang, C., Liu, Z., & Sun, M. (2023). Experiential Co-Learning of Software-Developing Agents. arXiv preprint arXiv:2312.17025. Retrieved from <https://arxiv.org/abs/2312.17025>

Chen, Q., Li, J., Dang, Y., Liu, W., Wang, Y., Xie, Z., Chen, W., Yang, C., Zhang, Y., Liu, Z., & Sun, M. (2024). Iterative Experience Refinement of Software-Developing Agents. arXiv preprint arXiv:2405.04219. Retrieved from <https://arxiv.org/abs/2405.04219>

Microsoft. (2019). Visual Studio IntelliCode: AI-Assisted Development. Retrieved from <https://visualstudio.microsoft.com/services/intellicode/>

JetBrains. (2020). Toward a Better IDE: Lessons Learned from Improving IntelliJ IDEA. Retrieved from <https://www.jetbrains.com/idea/>

OpenBMB. (2024). ChatDev GitHub Repository. Retrieved from <https://github.com/OpenBMB/ChatDev>

Pyzo. (n.d.). Pyzo IDE GitHub Repository. Retrieved from <https://github.com/pyzo/pyzo/tree/main>

OpenAI. (n.d.). OpenAI GPT Models. Retrieved from <https://openai.com/research>