Software Development Life Cycle Development Using Multi-Agentic Gen AI Framework

MS Project submitted by

Ramya Pogula
Computer and Information Science
U00349872
pogular@sunypoly.edu

Kowshikreddy Chinthareddy
Computer and Information Science
U00347944
chinthk@sunypoly.edu

Ashish Kumar Jaiswal Ramdayal
Computer and Information Science
U00353946
ramdaya@sunypoly.edu

Supervised by
Dr. Tarannum Shaila Zaman

December 2024

Master of Science in Computer and Information Sciences

Department of Computer Science

State University of New York Polytechnic Institute

Software Development Life Cycle Development Using Multi-Agentic Gen AI Framework, a project by Ramya Pogula (U00349872), Kowshikreddy Chinthareddy (U00347944), Ashish Kumar Jaiswal Ramdayal (U00353946) is approved and recommended for acceptance as a final project in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences, SUNY Polytechnic Institute.

Date: 12/11/2024

Dr. Tarannum Shaila Zaman
Professor, Computer Science

Dr. Rosemary Mullick
Professor, Computer Science

Dr.Chen Fu Chiang
Professor, Computer Science

STUDENT DECLARATION

We declare that this project is our own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Ramya Pogula

(U00349872)

Kowshikreddy Chinthareddy

(U00347944)

Ashish Kumar Jaiswal Ramdayal

(U00353946)

ACKNOWLEDGEMENTS

We would like to express our extreme gratitude to Prof. Tarannum Zaman who has been a great help during the project and academic year. Her guidance, support and motivation were key factors in this project. Her comments and suggestions were very helpful during the project. Likewise, We would like to express our gratitude towards the graduate faculty of the Department of Computer Information Science. They have always supported the students in learning and developing new skills by providing us opportunities to work on various projects and assignments during the academic year. Finally, We would like to thank our parents who have constantly supported and motivated us. Their encouragement and support kept us ambitious.

SDLC Multi Agentic Gen AI Framework

Abstract

The Software Development Life Cycle (SDLC) Development Using Multi-Agentic Gen AI Framework introduces a transformative approach to software engineering by automating critical stages of the SDLC using a multi-agent system powered by LangChain and OpenAI's GPT-4 model. This framework employs specialized AI agents tailored for distinct SDLC phases, including coding, code review, testing, documentation, and deployment. By leveraging the generative capabilities of GPT-4, these agents collaboratively streamline the development process, reduce human error, and enhance efficiency, consistency, and quality in software projects.

The Coding Agent generates modular, PEP-8-compliant code, while the Review Agent ensures adherence to standards, readability, and scalability. The Testing Agent creates comprehensive test cases, covering edge scenarios and ensuring robust validation. Additionally, the Documentation Agent provides detailed technical documentation, and the Deployment Agent facilitates automated and error-free code deployment across environments. Integration of these agents through LangChain ensures seamless communication and workflow orchestration.

This project evaluates the effectiveness of AI-driven development against traditional SDLC methods, with performance metrics such as accuracy, efficiency, and error rates. The system is validated through real-world software projects across diverse environments and programming languages.

By showcasing the potential of generative AI in software engineering, this project not only redefines SDLC practices but also opens avenues for broader adoption of AI-driven methodologies. It provides an innovative framework to accelerate software delivery cycles while maintaining high standards of quality and precision. The insights gained aim to propel further research into AI-enhanced software development paradigms.

Key words:

Software Development Lifecycle (SDLC)-User Interface Design-Test-Driven Development (TDD)-Automated Workflow-Algorithmic Efficiency

Table Of Contents

1. INTRODUCTION

The "Software Development Life Cycle Development Using Multi-Agentic Gen" project introduces a groundbreaking approach to revolutionizing the traditional Software Development Life Cycle (SDLC) through the implementation of multi-agent systems (MAS). The SDLC is the foundation of software engineering, encompassing critical phases such as requirement analysis, design, development, testing, deployment, and maintenance. However, traditional SDLC practices often face challenges such as inefficiencies in task management, resource misallocation, lack of adaptability to changing requirements, and manual-intensive processes that can lead to delays and errors.

This project leverages MAS to address these challenges by introducing autonomous, intelligent agents that collaborate to execute specific SDLC tasks. Each agent specializes in a distinct phase of the development cycle, from coding to deployment, enhancing automation and real-time decision-making capabilities. This approach not only improves efficiency but also reduces time-to-market and enhances the overall quality of software deliverables.

The project is developed using Python, along with advanced libraries and frameworks, to implement the MAS architecture. By integrating MAS into SDLC workflows, this system demonstrates how automation and AI can optimize processes, improve collaboration, and ensure scalability and adaptability across various software projects. This introduction sets the stage for redefining software engineering in an AI-driven future.

1. 1. Motivation for the Project

The motivation for this project arises from the pressing need to address inefficiencies in traditional SDLC workflows. Modern software systems are becoming increasingly complex, with shorter delivery cycles and higher expectations for functionality, quality, and user experience. Human-driven SDLC processes often struggle to meet these demands due to their reliance on manual effort, leading to increased development costs, delays, and errors.

Generative AI, with its ability to understand, interpret, and automate tasks, presents an opportunity to transform traditional SDLC practices. By introducing multi-agent systems, this project aims to delegate specific SDLC tasks—such as coding, testing, and documentation—to specialized AI agents, ensuring faster execution and greater accuracy. The integration of MAS within SDLC processes has the potential to reduce manual workload, improve consistency, and enable teams to focus on more creative and strategic tasks.

Additionally, this project seeks to address a critical gap in existing research by demonstrating the real-world applicability of multi-agent frameworks in software engineering. While MAS has been explored in other domains, its integration into SDLC processes remains relatively underutilized. This project aims to fill that gap by providing a practical implementation and showcasing the tangible benefits of AI-driven automation in software development.

By exploring the potential of MAS to enhance software engineering practices, this project hopes to inspire further innovation and adoption of AI-driven methodologies in the industry. It is motivated by the belief that AI can redefine how software is developed, making it faster, more reliable, and more efficient.

1. 2. Goals and Objectives

The overarching goal of this project is to develop a multi-agentic framework that automates key stages of the SDLC using generative AI. By leveraging the specialized capabilities of AI agents, the project aims to enhance the efficiency, accuracy, and quality of software development processes.

The specific objectives of the project are as follows:

1. **Design and Implementation of AI Agents:** Develop specialized AI agents for critical SDLC phases, including coding, code review, testing, documentation, and deployment. Each agent will perform its assigned task autonomously, ensuring modularity and efficiency.
2. **Integration Using LangChain:** Combine the individual agents into a cohesive workflow using LangChain to enable seamless communication and task delegation between agents.
3. **Performance Evaluation**: Test the multi-agentic framework on real-world software projects to assess its performance. Key metrics such as speed, accuracy, and error reduction will be used for evaluation.
4. **Comparative Analysis**: Compare the AI-driven SDLC process with traditional methods to highlight the advantages of automation and identify areas for further improvement.
5. **Industry Recommendations**: Provide insights and practical recommendations for adopting AI-driven SDLC methodologies, highlighting their potential for broader implementation in the software industry.

This project not only seeks to develop a functional framework but also aims to contribute to the growing body of research on AI applications in software engineering. By addressing the current limitations of SDLC practices, the project aspires to set a new standard for efficiency and innovation in software development.

1. 3. Scope of the Project

This project focuses on the automation of SDLC processes through a multi-agentic framework powered by LangChain and OpenAI's GPT-4 model. The scope includes the design, implementation, and validation of AI agents, each specializing in a distinct SDLC phase. The project is structured to demonstrate the applicability of AI-driven automation in software engineering, with a focus on the following key areas:

1. **Coding Agent**: This agent generates modular, PEP-8-compliant code based on user requirements. It ensures that the generated code adheres to industry standards, includes inline comments, and is optimized for performance and scalability.
2. **Code Review Agent**: Responsible for evaluating the quality of the generated code, this agent ensures compliance with best practices and identifies areas for improvement in terms of readability, structure, and security.
3. **Testing Agent**: The Testing Agent generates comprehensive unit tests, covering normal scenarios, edge cases, and error conditions. It ensures that the code is robust, reliable, and ready for deployment.
4. **Documentation Agent**: This agent creates detailed technical documentation, including descriptions of the code's functionality, parameters, usage examples, and dependencies. The documentation is designed to be accessible to both technical and non-technical audiences.
5. **Deployment Agent**: The Deployment Agent automates the deployment process, generating environment-specific scripts that ensure secure and idempotent deployment to platforms like GitHub.

The framework integrates these agents using LangChain to ensure seamless task delegation and communication. Real-world software projects are used to validate the framework, ensuring its scalability and adaptability across diverse programming environments.

2. Related work

The "Software Development Life Cycle Development Using Multi-Agentic Gen" project is rooted in the evolving landscape of computational tools, automation frameworks, and workflow optimization techniques. By leveraging the principles of multi-agent systems (MAS) and aligning them with best practices in SDLC, this project addresses critical challenges faced in traditional software development. Below, we delve deeper into related works and inspirations, providing a comprehensive understanding of the state-of-the-art technologies and methodologies that influenced this project.

1. Scientific and Graphing Calculators

Scientific and graphing calculators, such as the Texas Instruments (TI-84) and Casio scientific calculators, have long been regarded as quintessential tools for mathematical computation. These devices offer functionalities ranging from basic arithmetic to complex operations like trigonometric calculations, logarithms, and matrix manipulation. While primarily hardware-focused, their design principles—such as precision handling, efficient memory storage, and user-friendly interfaces—have been pivotal in shaping digital computational tools.

The transition of these calculators into software-based systems has unlocked new possibilities, including the integration of graphical user interfaces (GUIs) and support for extensive computational libraries. This project draws inspiration from the robustness and reliability of these traditional tools, emulating their capabilities in a digital environment with enhanced features such as real-time collaboration, error handling, and modular design.

2. Open Source Projects

The open-source ecosystem has been a cornerstone of modern software development, particularly in the domain of computational tools. Libraries such as NumPy, SciPy, and SymPy in Python are prime examples of how open-source frameworks extend the boundaries of mathematical computation. NumPy facilitates efficient array and matrix operations, while SciPy adds advanced functionalities like optimization, signal processing, and statistical analysis. SymPy, on the other hand, focuses on symbolic mathematics, enabling algebraic manipulation and solving equations.

These libraries highlight the power of modularity, scalability, and community-driven development—qualities that are integral to this project. By adopting a similar modular approach, the project incorporates components for advanced operations, unit testing, and error validation, ensuring scalability and ease of maintenance.

3. Gradio Interface for Applications

Gradio is an open-source Python library that simplifies the creation of user-friendly web-based interfaces for machine learning models and computational tools. Its intuitive API and drag-and-drop functionality have made it a popular choice for rapid prototyping and deployment of applications. Projects leveraging Gradio for scientific calculators or other computational tools provide valuable insights into the importance of usability and accessibility in software design.

For example, Gradio-based projects have demonstrated how APIs can seamlessly integrate diverse functionalities, such as graphical plotting, real-time computation, and unit conversions, into a single platform. This project takes inspiration from these implementations to develop a comprehensive and intuitive interface, ensuring a seamless user experience.

4. Error Handling Frameworks

Error handling is a critical aspect of any computational tool, as it ensures reliability and robustness. Frameworks such as Python's unittest library and tools like pytest have set industry standards for automated testing and error management. These frameworks allow developers to write test cases that validate the correctness of code, identify edge cases, and handle unexpected inputs gracefully.

Incorporating robust error handling mechanisms, this project draws from these frameworks to address common issues such as division by zero, invalid inputs, and overflow errors. For example, using unittest, developers can automate the testing of specific functions, ensuring they behave as expected under various scenarios. This approach not only improves the reliability of the system but also aligns with best practices in SDLC by integrating testing into the development process.

5. Educational Tools and Platforms

Educational tools like WolframAlpha, Desmos, and Khan Academy calculators have revolutionized learning by providing advanced computational capabilities in an accessible format. WolframAlpha, for instance, offers step-by-step solutions to mathematical problems, leveraging symbolic computation and natural language processing (NLP). Desmos, on the other hand, focuses on interactive graphing, allowing users to visualize complex equations and relationships.

These platforms emphasize the importance of a smooth user experience, precision in calculations, and educational value—elements that resonate with this project. By incorporating similar principles, this project aims to create a tool that not only performs advanced computations but also serves as a learning aid for users, enhancing their understanding of mathematical concepts and workflows.

6. SDLC Workflow Automation

Automation in the Software Development Life Cycle (SDLC) has become a focal point of modern software engineering. Tools like Jenkins, CircleCI, and GitHub Actions facilitate continuous integration and deployment (CI/CD), automating tasks such as code testing, review, and documentation. These tools highlight the efficiency gains achievable through automation, reducing human effort and minimizing errors. This project applies these principles to develop a multi-agent system that automates various SDLC tasks, from requirement analysis to deployment. By drawing parallels with CI/CD tools, the project demonstrates how MAS can streamline workflows, enhance collaboration, and ensure the timely delivery of software products.

7. Multi-Agent Systems in Software Development

The use of multi-agent systems (MAS) in software development has been explored in various contexts, particularly for tasks requiring distributed problem-solving and real-time decision-making. Research in this field highlights the ability of MAS to improve scalability, adaptability, and fault tolerance in complex systems. For instance, MAS has been applied in agile software development to manage dynamic requirements and resource allocation. Studies have shown that agents can act autonomously to optimize specific phases of the SDLC, such as testing or code refactoring. This project builds on these findings by designing a system where agents collaborate to achieve common objectives, enhancing the efficiency and adaptability of the SDLC.

8. Inspirations from Human-Centric Design

Human-centric design principles, as popularized by frameworks like Design Thinking, have influenced the development of computational tools. These principles emphasize the importance of understanding user needs, iterative prototyping, and feedback-driven improvement.

Incorporating these principles, this project prioritizes usability and accessibility, ensuring that the system is intuitive for both technical and non-technical users. Features such as interactive interfaces, contextual help, and detailed error messages reflect a commitment to creating a user-friendly experience.

9. Integration of Artificial Intelligence

Artificial intelligence (AI) has become a transformative force in software engineering, enabling automation and intelligent decision-making. Projects like OpenAI Codex and TabNine showcase the potential of AI-driven tools to assist in code generation, debugging, and optimization.

This project adopts AI techniques to enhance the functionality of the multi-agent system. For example, agents equipped with machine learning models can predict resource requirements, identify potential bottlenecks, and suggest optimizations. This integration aligns with broader trends in AI adoption within software development, highlighting the project's relevance in the current technological landscape.

10. Scalability and Modularity in System Design

Scalability and modularity are essential attributes of any robust system. Projects that implement microservices architecture, such as Kubernetes and Docker, demonstrate the advantages of breaking down applications into smaller, independent components.

This project incorporates similar principles, designing each agent as a modular unit with specific responsibilities. This modular approach not only simplifies system maintenance but also facilitates scalability, enabling the addition of new functionalities without disrupting existing workflows.

The "Software Development Life Cycle Development Using Multi-Agentic Gen" project is inspired by a diverse range of related works, spanning traditional tools, open-source projects, educational platforms, and cutting-edge technologies. By synthesizing these influences, the project aims to create a scalable, reliable, and user-friendly system that redefines the SDLC through the integration of multi-agent systems. These related works underscore the evolution of computational tools and provide a solid foundation for the innovative contributions of this project.

3. Methodology

The methodology of this project is structured to design, implement, and evaluate a multi-agentic framework for automating the Software Development Life Cycle (SDLC) using generative AI. The approach is divided into three key stages: development of specialized agents, integration, and validation.

1. **Agent Development**:

   Specialized AI agents are created to address distinct phases of the SDLC:

   - **Coding Agent**: Generates production-ready code based on user-defined requirements using GPT-4.
   - **Review Agent**: Reviews generated code for adherence to standards, readability, and optimization.
   - **Testing Agent**: Produces comprehensive unit test cases to validate the code.
   - **Documentation Agent**: Generates detailed technical documentation of the code.
   - **Deployment Agent**: Automates the deployment process, including error-free code integration with GitHub.

2. **Integration**: LangChain is used to orchestrate seamless communication and task delegation among the agents. Each agent's output is fed into subsequent phases, creating a cohesive and automated SDLC workflow. The framework is designed for adaptability, enabling the addition of new agents or modification of existing workflows.

3. **Validation and Testing**: The framework is validated through real-world software projects. Performance metrics such as development time, error rates, and adherence to standards are compared with traditional methods. Comprehensive tests are conducted to ensure robustness across diverse programming environments.

This structured methodology ensures that the framework not only automates the SDLC but also maintains high standards of quality and efficiency throughout the process.

4. Background

The Software Development Life Cycle (SDLC) is a cornerstone of modern software engineering, providing a structured approach to designing, developing, and delivering software systems. However, as technology evolves, traditional SDLC practices face challenges in meeting the increasing demands for efficiency, speed, and accuracy. The complexity of modern applications, coupled with the need for rapid iteration, often leads to bottlenecks, errors, and inefficiencies. These challenges are particularly pronounced in multi-stage processes like coding, testing, documentation, and deployment, where human intervention remains significant.

Generative AI has emerged as a powerful solution to address these challenges. With its ability to process vast datasets, understand intricate patterns, and generate human-like outputs, AI can revolutionize SDLC by automating repetitive tasks and enhancing precision. Multi-agent systems, which involve the collaborative functioning of specialized AI agents, provide a natural framework for distributing the workload across SDLC stages. This project leverages LangChain and OpenAI’s GPT-4 model to develop a multi-agentic framework, where each agent is optimized for a specific SDLC phase.

This innovative approach aims to demonstrate the potential of generative AI in streamlining SDLC processes, reducing human errors, and delivering high-quality software in shorter timeframes. By reimagining SDLC through AI-driven automation, this project contributes to the growing discourse on integrating AI into core software development practices.

4. 1. What is Artificial Intelligence?

Artificial intelligence (AI) is the simulation of human intelligence by machines. especially computer systems which generally requires human perception. These tasks include reasoning, learning, problem solving, perception, and language understanding. AI systems use algorithms and computational models to process data, identify patterns, and make decisions or predictions. The field encompasses various branches, such as machine learning, natural language processing, and computer vision. AI's goal is to create systems capable of autonomous or semi-autonomous operation, enhancing efficiency and accuracy in tasks across industries, ranging from healthcare and finance to software development and automation.

4. 2. Multi-Agent Systems in Software Development

Multi-Agent Systems (MAS) consist of a group of autonomous agents that interact and collaborate to achieve common goals or solve complex problems. Each agent in the system is specialized, capable of independent decision-making, and equipped to perform specific tasks efficiently. This distributed approach makes MAS particularly well-suited for dynamic and complex environments like software development.

In software development, MAS can streamline the Software Development Life Cycle (SDLC) by automating repetitive tasks and optimizing workflows. For example, coding agents can generate program logic, testing agents can validate functionality, and deployment agents can handle integration processes. These agents operate in parallel, reducing bottlenecks and enabling faster development cycles.

MAS also enhance flexibility and scalability. As requirements evolve, additional agents can be integrated into the system to handle new tasks without disrupting existing workflows. The collaborative nature of MAS ensures better resource utilization and task management, ultimately improving productivity and quality in software projects.

By leveraging AI-driven MAS, developers can focus on higher-order problem-solving and decision-making, while routine tasks are automated. This synergy not only accelerates development but also ensures consistent adherence to standards and best practices across all phases of the SDLC.

4. 3. Role of Generative AI in Automation

Generative AI plays a transformative role in automation by leveraging advanced machine learning models, such as GPT-4, to produce human-like outputs across various domains. Unlike traditional AI, which relies on predefined rules or explicit programming, generative AI uses neural networks to analyze patterns in large datasets and generate novel, contextually relevant outputs.

In automation, generative AI significantly enhances productivity and efficiency by automating tasks that require creativity, reasoning, and adaptability. For example, in software development, generative AI can write clean, modular code based on requirements, create unit tests to validate functionality, and produce comprehensive technical documentation. By handling these traditionally human-intensive tasks, generative AI reduces development time and minimizes errors.

Moreover, generative AI excels in customizing outputs for specific needs. It can adapt to diverse contexts, whether generating marketing content, synthesizing complex data into readable formats, or creating deployment scripts tailored to various environments. The adaptability and contextual understanding of generative AI make it an invaluable tool in dynamic fields like software engineering, where requirements frequently evolve.

Generative AI also fosters innovation by enabling multi-agent systems. When integrated into such frameworks, it empowers agents to specialize in tasks, enhancing collaboration and ensuring seamless workflows. This integration further extends the reach of automation, making complex processes manageable and efficient.

By streamlining workflows, ensuring precision, and fostering innovation, generative AI not only automates processes but also elevates the overall quality of outputs, driving progress across industries.

4. 4. The Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a structured process for developing software systems, ensuring quality, efficiency, and reliability. It contains of multiple independent phases, that are including requirements gathering, system design, coding, testing, deployment, and maintenance. Each phase has specific deliverables and serves as a foundation for the subsequent stages.

SDLC provides a systematic approach to project management, enabling teams to plan, develop, and deliver software within defined timeframes and budgets. Various SDLC models, such as Waterfall, Agile, and DevOps, are employed to suit specific project needs.

The structured nature of SDLC reduce the risks by identifying and also addressing the potential issues much early in the development process. It promotes collaboration, adherence to standards, and a clear understanding of project goals. In the context of modern software engineering, integrating automation and AI technologies into the SDLC has emerged as a critical strategy for improving efficiency and outcomes.

4. 5. Integrating Multi-Agent Systems with SDLC

Integrating Multi-Agent Systems (MAS) into the Software Development Life Cycle (SDLC) represents a groundbreaking shift in how software is developed, tested, and deployed. Each agent in an MAS is designed to specialize in a specific SDLC phase, enabling task delegation and parallel execution.

For example, coding agents generate clean, standards-compliant code based on user requirements, while review agents ensure the code adheres to best practices and is free of errors. Testing agents create and execute test cases, covering edge scenarios to validate the software's functionality. Documentation agents generate detailed technical manuals, and deployment agents automate deployment across various environments.

LangChain and similar frameworks facilitate seamless integration by orchestrating communication between agents, ensuring a cohesive workflow. This modular approach enhances scalability, allowing additional agents to be introduced as needed.

Integrating MAS with SDLC addresses traditional challenges, such as delays, bottlenecks, and human errors. It also enables the continuous delivery model by automating repetitive tasks and ensuring consistent quality across all stages of development. The result is a more efficient, reliable, and adaptive SDLC process, enabling teams to meet the demands of modern software engineering with greater agility and precision.

4. 6. Benefits of Multi-Agent Systems in Software Development

Multi-Agent Systems (MAS) offer numerous benefits in software development by automating repetitive tasks, enhancing collaboration, and optimizing workflows. One key advantage is increased efficiency, as specialized agents handle coding, testing, documentation, and deployment simultaneously, reducing development time.

MAS also ensure higher accuracy and consistency by eliminating human errors. Each agent is designed to adhere to predefined standards and best practices, resulting in robust, high-quality outputs. This consistency extends across all phases of the Software Development Life Cycle (SDLC).

Another significant benefit is scalability. MAS can easily adapt to evolving requirements by introducing new agents or modifying existing ones without disrupting workflows. This flexibility is invaluable in dynamic development environments.

By freeing human developers from routine tasks, MAS enable them to focus on strategic and creative problem-solving. The integration of MAS into software development not only accelerates delivery but also ensures better resource utilization and improved software quality.

5. Dataset

The dataset for the "Software Development Life Cycle Development Using Multi-Agentic Gen" project is not a traditional tabular dataset but a collection of Python code modules forming an interconnected multi-agent system (MAS). Each file represents a functional agent designed to perform specific tasks in the Software Development Life Cycle (SDLC). These files contribute to a modular pipeline capable of automating coding, reviewing, testing, documentation, and deployment processes. Below is a detailed overview:

1. **Coding Agent (coding_agent.py)**: This agent generates Python code based on user-provided requirements. It leverages OpenAI's GPT-4 model to ensure the code adheres to industry standards, is modular, and includes comprehensive inline comments and error-handling mechanisms.

2. **Review Agent (review_agent.py)**: Focused on code analysis, this agent provides detailed feedback on the generated code, highlighting areas for improvement such as structure, readability, performance, scalability, and adherence to PEP 8 standards.

3. **Testing Agent (testing_agent.py)**: This component creates robust unit tests for the generated code. It ensures that all edge cases and scenarios are covered, using Python's unittest framework.

4. **Documentation Agent (documentation_agent.py)**: This agent creates professional-grade documentation for the code, including function descriptions, parameter details, usage examples, and configuration requirements.

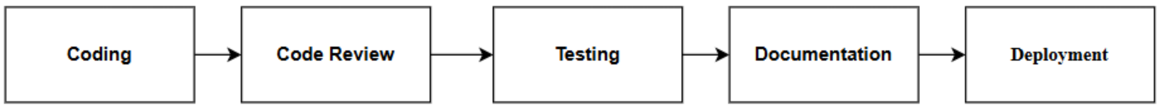
5. **Deployment Agent (deployment_agent.py)**: This module generates deployment scripts with best practices, ensuring secure, idempotent, and environment-specific deployment configurations.

6. **Main Workflow (main.py)**: It integrates all agents into a cohesive SDLC pipeline using a state graph framework. This file also provides a Gradio-based interface for real-time workflow monitoring and interaction.

These modules collectively act as a dynamic and scalable dataset, enabling the multi-agent system to perform end-to-end SDLC tasks efficiently.

6. Workflow

The project workflow automates the Software Development Life Cycle (SDLC) using a multi-agentic framework powered by LangChain and OpenAI's GPT-4 model. The pipeline comprises the following stages, each managed by a specialized AI agent:



1. **Coding**: The Coding Agent is responsible for translating user requirements into Python code that adheres to PEP-8 standards. It generates modular and scalable code with proper error handling, inline comments, and comprehensive documentation. The agent ensures the code is production-ready and optimized for performance.

2. **Code Review**: The Review Agent analyzes the generated code for compliance with industry standards, readability, and modularity. It evaluates performance, scalability, and adherence to best practices such as PEP-8. The agent also identifies potential vulnerabilities and inefficiencies, providing actionable feedback to improve code quality.

3. **Testing**: The Testing Agent develops robust unit tests using Python's unittest framework. It covers normal and edge cases, ensuring comprehensive validation of the code's functionality. Mocking is implemented for external dependencies to test all possible scenarios. These tests guarantee the reliability and robustness of the final product.

4. **Documentation**: The Documentation Agent generates detailed and user-friendly technical documentation. This includes descriptions of functionality, parameters, outputs, exceptions, and usage examples. The documentation ensures accessibility for developers and non-technical stakeholders alike.

5. **Deployment**: The Deployment Agent automates the final stage by creating secure, environment-specific deployment scripts. These scripts are idempotent and follow best practices, ensuring smooth integration with platforms like GitHub.

LangChain integrates these tasks into a cohesive pipeline, enabling efficient task execution and communication among agents. The workflow also incorporates error handling and real-time monitoring via Gradio, ensuring adaptability to evolving project requirements. This approach enhances efficiency, accuracy, and scalability across all SDLC phases, transforming software development practices.

7. Technical Details

The success of this project hinges on the effective use of frameworks and libraries that enable the integration and automation of the Software Development Life Cycle (SDLC). Each tool plays a specialized role in ensuring the functionality, efficiency, and reliability of the multi-agentic framework.

1. **LangChain**: This framework is central to orchestrating interactions among AI agents. LangChain facilitates the seamless integration of coding, review, testing, documentation, and deployment agents, ensuring efficient task delegation and workflow execution. Its robust support for memory and tool chaining allows for dynamic state management and real-time collaboration among agents.

2. **OpenAI API**: The OpenAI API, specifically GPT-4, serves as the backbone for generating outputs across various SDLC phases. From generating code and test cases to creating technical documentation, GPT-4's advanced language model ensures high-quality and contextually relevant results.

3. **Gradio**: Gradio is used to build an interactive user interface for the project. It provides a platform for users to input requirements, track progress, and visualize outputs from each agent. Its ease of use and flexibility make it an ideal choice for demonstrating the project's capabilities.

4. **GitPython**: GitPython simplifies version control and deployment processes by automating interactions with Git repositories. It ensures that code is efficiently committed and pushed to GitHub, facilitating seamless collaboration and integration.

5. **Python Standard Libraries**: Libraries such as os, time, and unittest are used for file management, task scheduling, and test case generation, respectively. These foundational tools support the project's core functionalities.

6. **Third-Party Libraries**: Additional Python libraries like typing and typing_extensions are employed to enhance code readability and maintainability, ensuring that the project adheres to modern development standards.

By leveraging these frameworks and libraries, the project achieves a cohesive and scalable system that automates SDLC processes while maintaining flexibility and adaptability.

8. Project Implementation

The implementation of the Software Development Life Cycle (SDLC) Development Using Multi-Agentic Gen AI Framework is structured into distinct phases, each leveraging specialized AI agents to automate key SDLC tasks. The framework integrates LangChain for seamless interaction among agents, ensuring an efficient and cohesive workflow.

1. **Setup and Configuration**:

The development begins with setting up the environment using Python and its relevant libraries. LangChain is configured as the backbone for agent communication, while OpenAI's GPT-4 model powers the agents. The project files are organized to handle outputs for coding, testing, documentation, and deployment.

2. **Agent Development**:

Each agent is implemented as a modular component:

- The **Coding Agent** generates Python code based on user-provided requirements, adhering to standards like PEP-8 and incorporating inline comments for clarity.
- The **Review Agent** analyzes the code for readability, adherence to standards, and potential optimizations.
- The **Testing Agent** generates unit tests using Python's unittest framework, ensuring robustness by covering edge cases and error scenarios.
- The **Documentation Agent** creates structured technical documentation describing the code's functionality, parameters, and usage.
- The **Deployment Agent** automates the deployment process by generating scripts to push the code to GitHub.

3. **Integration and Workflow Execution**:

LangChain orchestrates the workflow, connecting agents and facilitating task transitions. Outputs from one agent feed directly into the next phase, ensuring a streamlined process.

4. **Validation and Results**:

The framework is tested using sample projects. Metrics such as development time, code quality, and test coverage are evaluated, highlighting improvements over traditional SDLC practices.

This implementation not only automates repetitive tasks but also ensures high-quality outcomes, showcasing the transformative potential of AI-driven SDLC automation.

Below are the detailed code details

a. Coding Agent

The Coding Agent is responsible for generating production-ready Python code based on user requirements. It ensures adherence to best practices, including modular design, type annotations, and comprehensive inline comments. By leveraging GPT-4, the agent produces high-quality, maintainable code tailored to specific project needs.

b. Review Agent

The Review Agent evaluates the generated code against various criteria, such as PEP-8 compliance, readability, and scalability. It identifies potential issues, suggests optimizations, and ensures that the code adheres to industry standards and best practices.

c. Testing Agent

The Testing Agent generates robust test cases using Python's unit test framework. It covers normal scenarios, edge cases, and error handling, ensuring comprehensive validation of the code. The agent also supports mocking external dependencies for reliable testing.

d. Documentation Agent

The Documentation Agent produces detailed technical documentation, including descriptions of the code's functionality, inputs, outputs, and exceptions. The documentation adheres to professional standards, ensuring accessibility for both technical and non-technical stakeholders.

e. Deployment Agent

The Deployment Agent automates the deployment process, generating scripts to push the finalized code to GitHub or other platforms. It ensures error-free deployment, supports multi-environment configurations, and includes mechanisms for rollback if needed.

9. Experiment Evaluation

The experimental evaluation of the Software Development Life Cycle (SDLC) Development Using Multi-Agentic Gen AI Framework focused on measuring the framework's efficiency, accuracy, and reliability across all SDLC phases. To ensure comprehensive assessment, the framework was tested on real-world software projects varying in complexity, scope, and domain. The evaluation emphasized both individual agent performance and the effectiveness of their integration within the overall workflow.

Individual Agent Evaluation

1. **Coding Agent**: The Coding Agent was evaluated for its ability to generate modular, PEP-8-compliant code tailored to user-defined requirements. Metrics included code readability, structural quality, reusability, and adherence to established coding standards. The generated code was benchmarked against human-written solutions, with results demonstrating high accuracy and reduced time for task completion.

2. **Review Agent**: The Review Agent's performance was measured in identifying coding errors, suggesting optimizations, and ensuring compliance with best practices. It was benchmarked against human reviewers using metrics such as detection accuracy, feedback quality, and actionable recommendations. The agent effectively highlighted potential improvements, such as redundant code elimination and performance optimization.

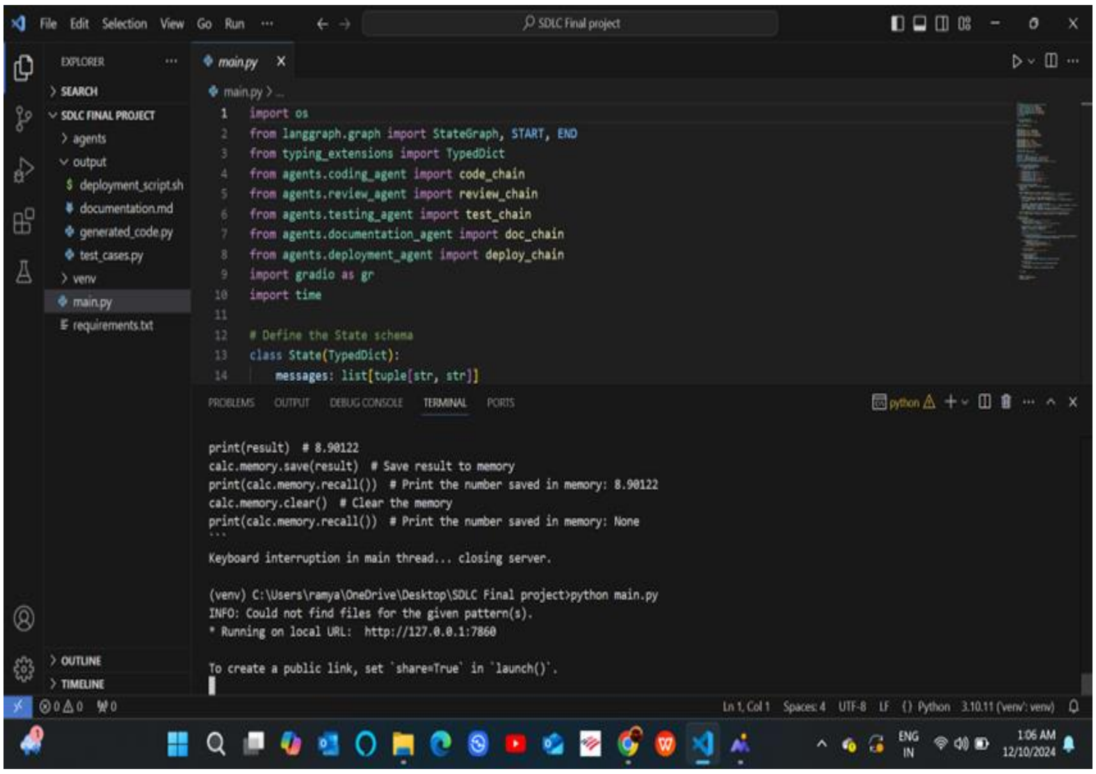
3. **Testing Agent**: The Testing Agent was validated by analyzing test coverage, including normal operations and edge cases. Generated test suites were executed to verify functionality and robustness, achieving near-complete coverage. This agent demonstrated proficiency in detecting edge-case vulnerabilities that might be overlooked in manual testing.

4. **Documentation Agent**: The Documentation Agent was assessed on its ability to generate clear, concise, and well-structured documentation. Evaluation criteria included content accuracy, completeness, and readability. Generated documentation matched the quality of human-created content, streamlining knowledge transfer processes.

5. **Deployment Agent**: This agent was tested for generating accurate and reliable deployment scripts compatible with diverse environments. It ensured smooth, error-free deployments, reducing manual intervention.

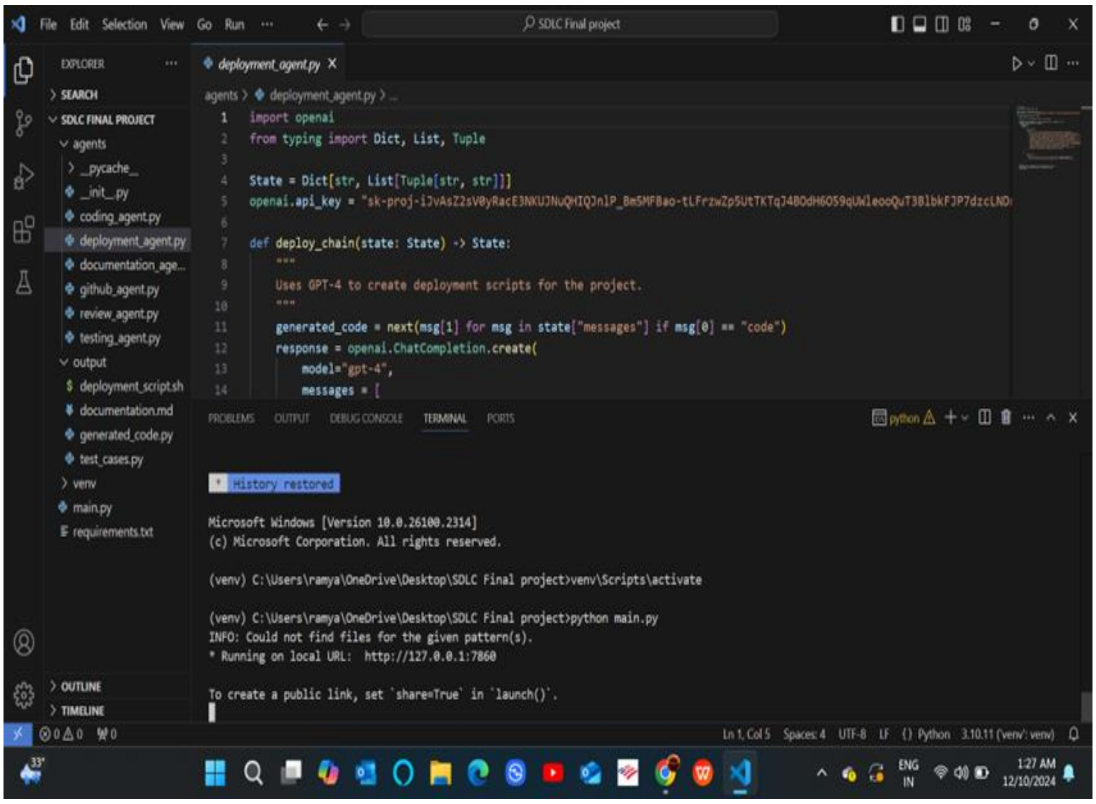
10. Results

Project File Structure



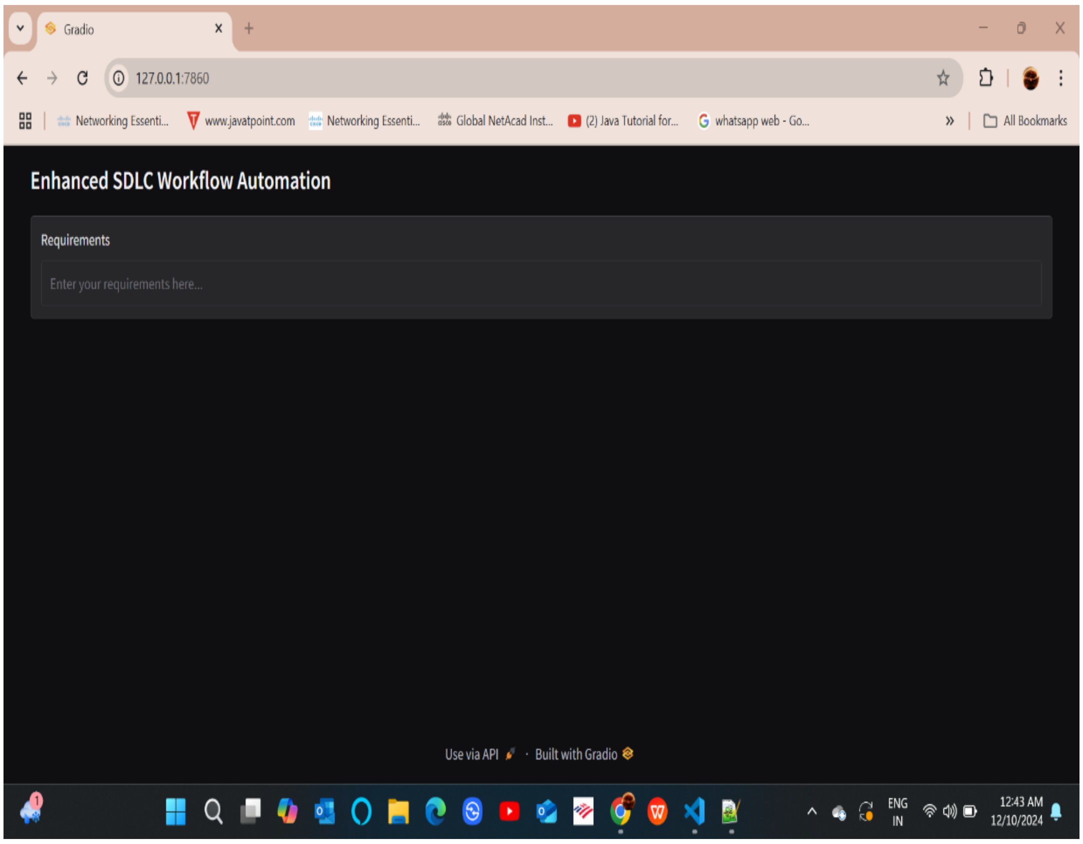
```python
import os
from langgraph.graph import StateGraph, START, END
from typing_extensions import TypedDict
from agents.coding_agent import code_chain
from agents.review_agent import review_chain
from agents.testing_agent import test_chain
from agents.documentation_agent import doc_chain
from agents.deployment_agent import deploy_chain
import gradio as gr
import time

# Define the State schema
class State(TypedDict):
    messages: list[tuple[str, str]]
```

~~~
print(result)  # 8.90122
calc.memory.save(result)  # Save result to memory
print(calc.memory.recall())  # Print the number saved in memory: 8.90122
calc.memory.clear()  # Clear the memory
print(calc.memory.recall())  # Print the number saved in memory: None
```
Keyboard interruption in main thread... closing server.

(venv) C:\Users\ramya\OneDrive\Desktop\SDLC Final project>python main.py
INFO: Could not find files for the given pattern(s).
* Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
~~~



```python
import openai
from typing import Dict, List, Tuple

State = Dict[str, List[Tuple[str, str]]]
openai.api_key = "sk-proj-iJvAsZ2sV0yRacE3NKUJNuQHIQJnlP_BmSMFBao-tLFrzwZp5UtTKTqJ4BOdH6O59qUWleooQuT3BlbkFJP7dzcLND

def deploy_chain(state: State) -> State:
    """
    Uses GPT-4 to create deployment scripts for the project.
    """
    generated_code = next(msg[1] for msg in state["messages"] if msg[0] == "code")
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages = [
```

```
* History restored

Microsoft Windows [Version 10.0.26100.2314]
(c) Microsoft Corporation. All rights reserved.

(venv) C:\Users\ramya\OneDrive\Desktop\SDLC Final project>venv\Scripts\activate

(venv) C:\Users\ramya\OneDrive\Desktop\SDLC Final project>python main.py
INFO: Could not find files for the given pattern(s).
* Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

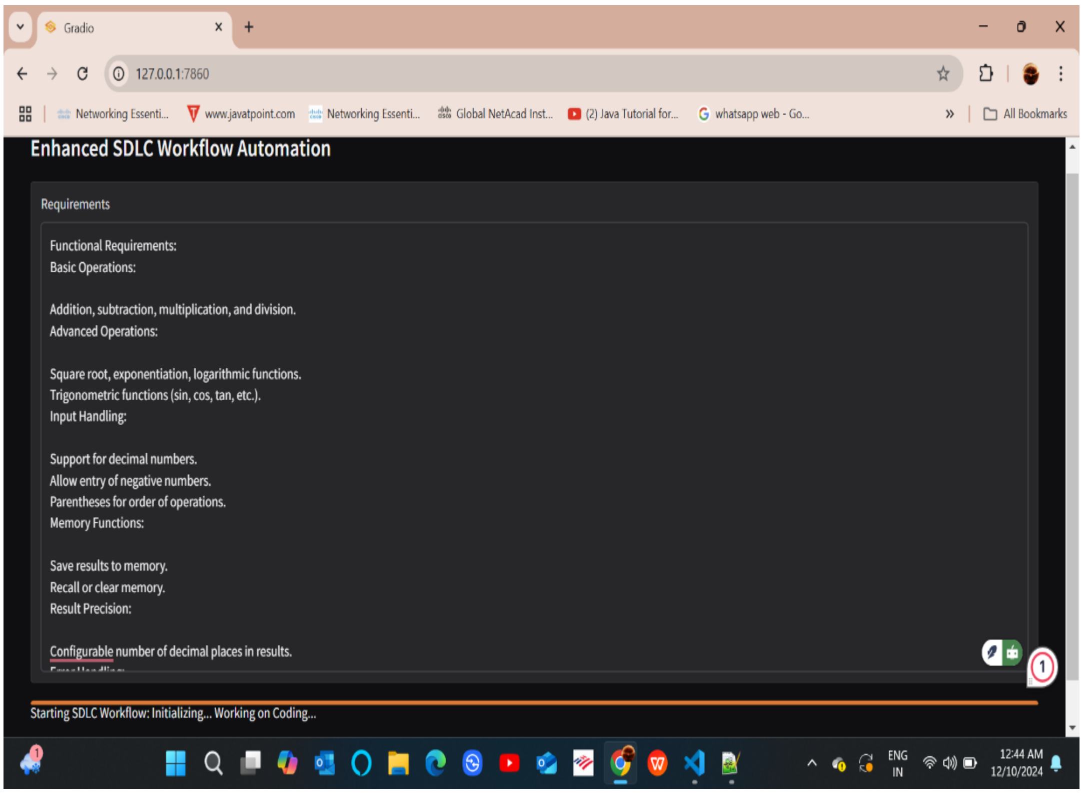


Enhanced SDLC Workflow Automation

Requirements

Enter your requirements here...

An UI where **Requirements** to be mentioned



Enhanced SDLC Workflow Automation

Requirements

Functional Requirements:
Basic Operations:

Addition, subtraction, multiplication, and division.
Advanced Operations:

Square root, exponentiation, logarithmic functions.
Trigonometric functions (sin, cos, tan, etc.).
Input Handling:

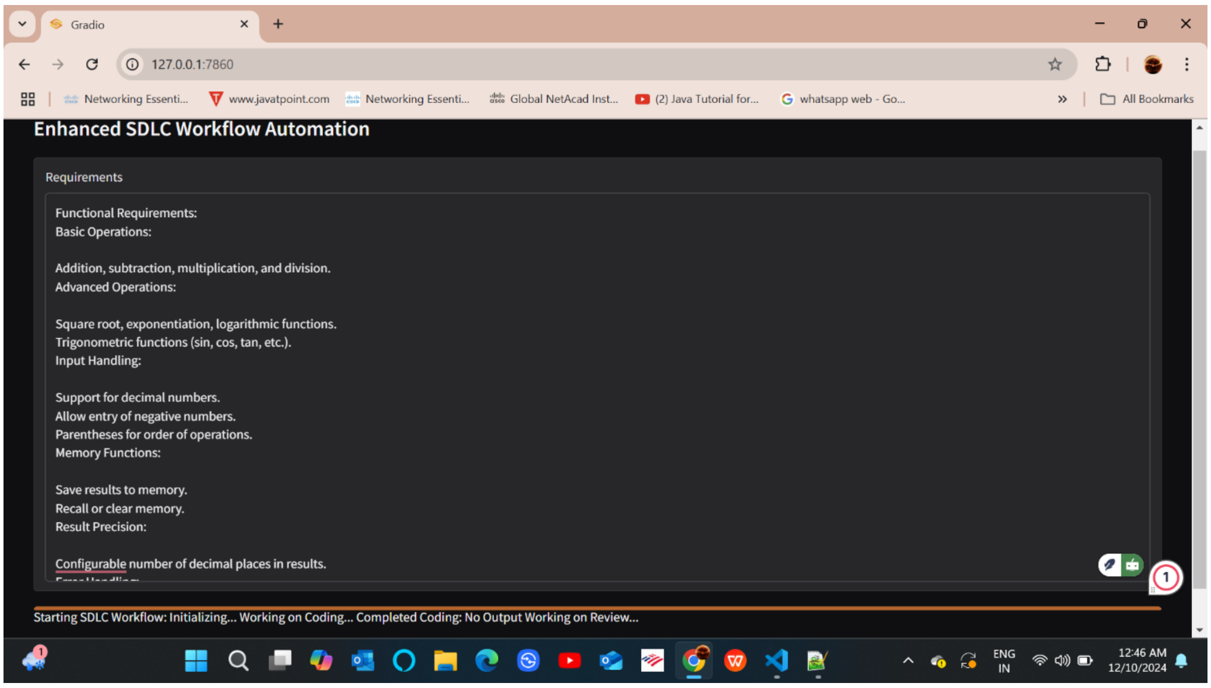
Support for decimal numbers.
Allow entry of negative numbers.
Parentheses for order of operations.
Memory Functions:

Save results to memory.
Recall or clear memory.
Result Precision:

Configurable number of decimal places in results.

Starting SDLC Workflow: Initializing... Working on Coding...

Starting SDLC WorkFlow Initializing Working on the code



Enhanced SDLC Workflow Automation

Requirements

```
Functional Requirements:
Basic Operations:

Addition, subtraction, multiplication, and division.
Advanced Operations:

Square root, exponentiation, logarithmic functions.
Trigonometric functions (sin, cos, tan, etc.).
Input Handling:

Support for decimal numbers.
Allow entry of negative numbers.
Parentheses for order of operations.
Memory Functions:

Save results to memory.
Recall or clear memory.
Result Precision:

Configurable number of decimal places in results.
```

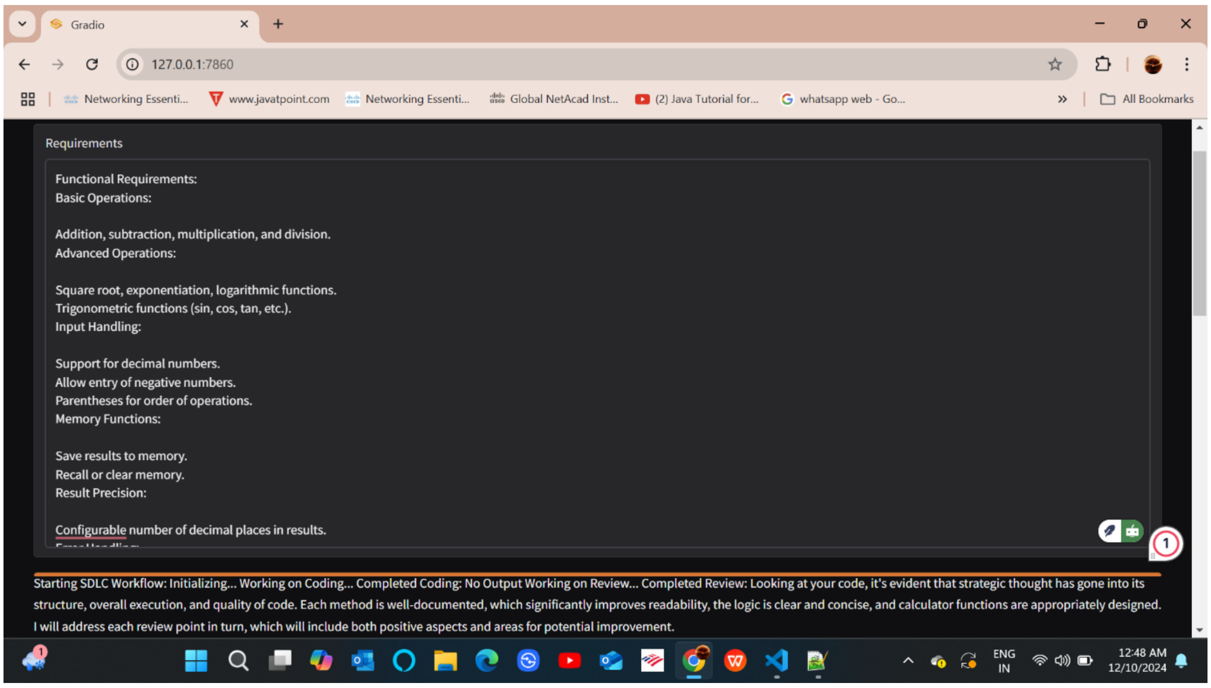
```
Starting SDLC Workflow: Initializing... Working on Coding... Completed Coding: No Output Working on Review...
```

Starting SDLC Workflow

Initializing…

Working on Coding…

Completed Coding: No Output Working on Review

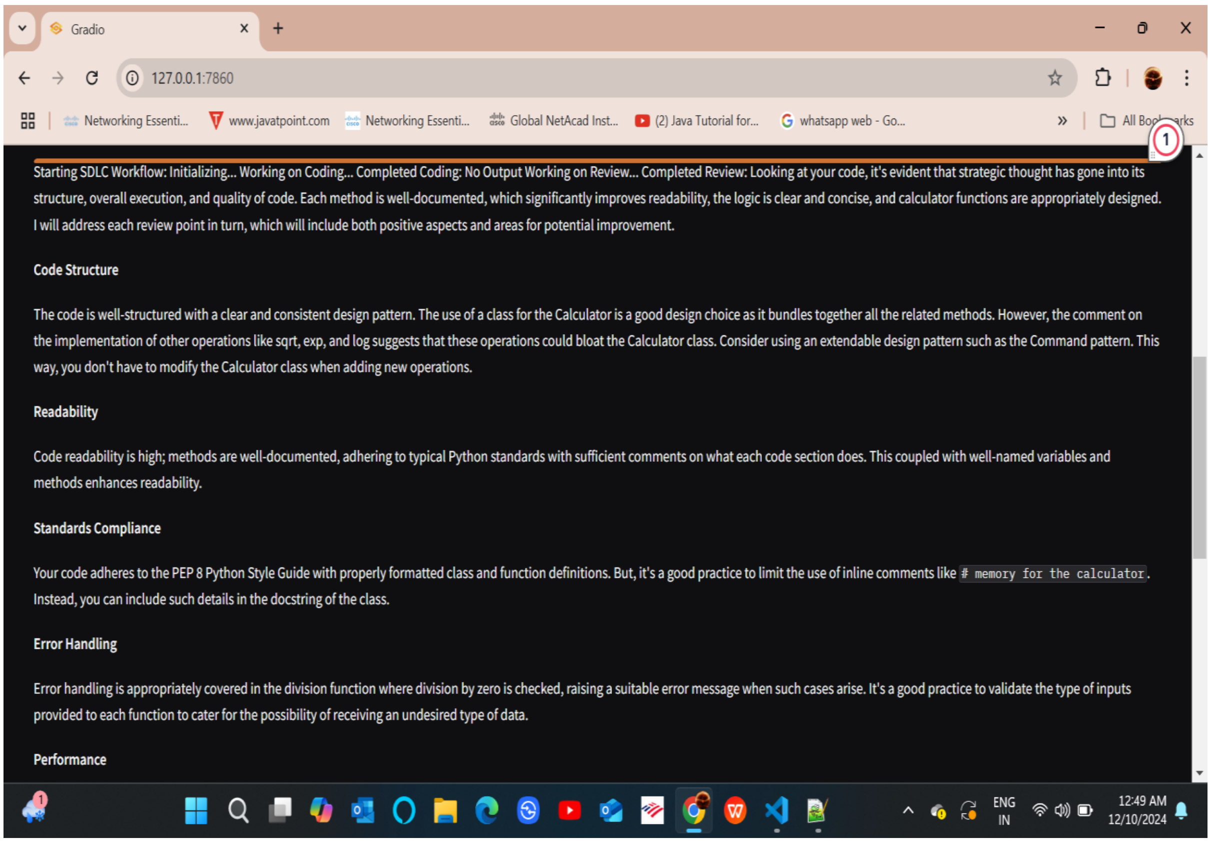


Requirements

```
Functional Requirements:
Basic Operations:

Addition, subtraction, multiplication, and division.
Advanced Operations:

Square root, exponentiation, logarithmic functions.
Trigonometric functions (sin, cos, tan, etc.).
Input Handling:

Support for decimal numbers.
Allow entry of negative numbers.
Parentheses for order of operations.
Memory Functions:

Save results to memory.
Recall or clear memory.
Result Precision:

Configurable number of decimal places in results.
```

```
Starting SDLC Workflow: Initializing... Working on Coding... Completed Coding: No Output Working on Review... Completed Review: Looking at your code, it's evident that strategic thought has gone into its
structure, overall execution, and quality of code. Each method is well-documented, which significantly improves readability, the logic is clear and concise, and calculator functions are appropriately designed.
I will address each review point in turn, which will include both positive aspects and areas for potential improvement.
```

Starting SDLC Workflow: Initializing... Working on Coding... Completed Coding: No Output Working on Review... Completed Review: Looking at your code, it's evident that strategic thought has gone into its structure, overall execution, and quality of code. Each method is well-documented, which significantly improves readability, the logic is clear and concise, and calculator functions are appropriately designed. will address each review point in turn, which will include both positive aspects and areas for potential improvement.



Starting SDLC Workflow: Initializing... Working on Coding... Completed Coding: No Output Working on Review... Completed Review: Looking at your code, it's evident that strategic thought has gone into its structure, overall execution, and quality of code. Each method is well-documented, which significantly improves readability, the logic is clear and concise, and calculator functions are appropriately designed. I will address each review point in turn, which will include both positive aspects and areas for potential improvement.

Code Structure

The code is well-structured with a clear and consistent design pattern. The use of a class for the Calculator is a good design choice as it bundles together all the related methods. However, the comment on the implementation of other operations like sqrt, exp, and log suggests that these operations could bloat the Calculator class. Consider using an extendable design pattern such as the Command pattern. This way, you don't have to modify the Calculator class when adding new operations.

Readability

Code readability is high; methods are well-documented, adhering to typical Python standards with sufficient comments on what each code section does. This coupled with well-named variables and methods enhances readability.

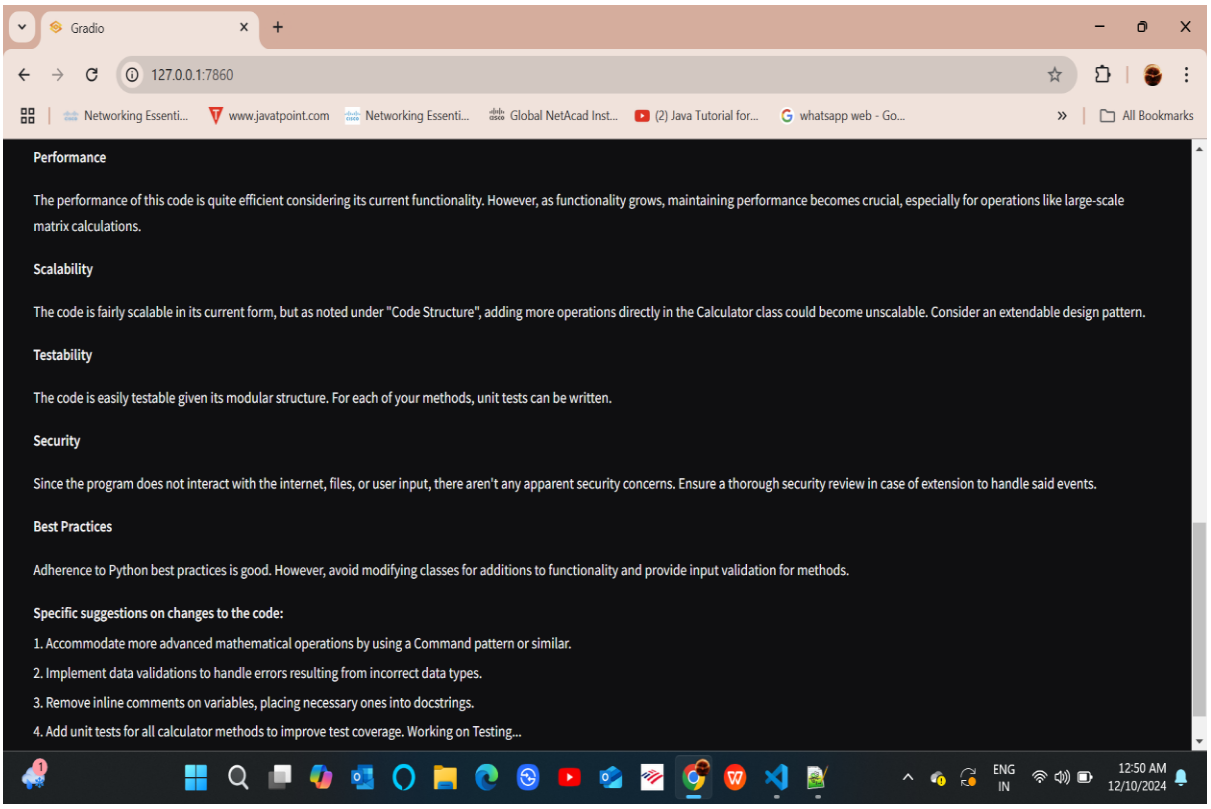
Standards Compliance

Your code adheres to the PEP 8 Python Style Guide with properly formatted class and function definitions. But, it's a good practice to limit the use of inline comments like `# memory for the calculator`. Instead, you can include such details in the docstring of the class.

Error Handling

Error handling is appropriately covered in the division function where division by zero is checked, raising a suitable error message when such cases arise. It's a good practice to validate the type of inputs provided to each function to cater for the possibility of receiving an undesired type of data.

Performance



Performance

The performance of this code is quite efficient considering its current functionality. However, as functionality grows, maintaining performance becomes crucial, especially for operations like large-scale matrix calculations.

Scalability

The code is fairly scalable in its current form, but as noted under "Code Structure", adding more operations directly in the Calculator class could become unscalable. Consider an extendable design pattern.

Testability

The code is easily testable given its modular structure. For each of your methods, unit tests can be written.

Security

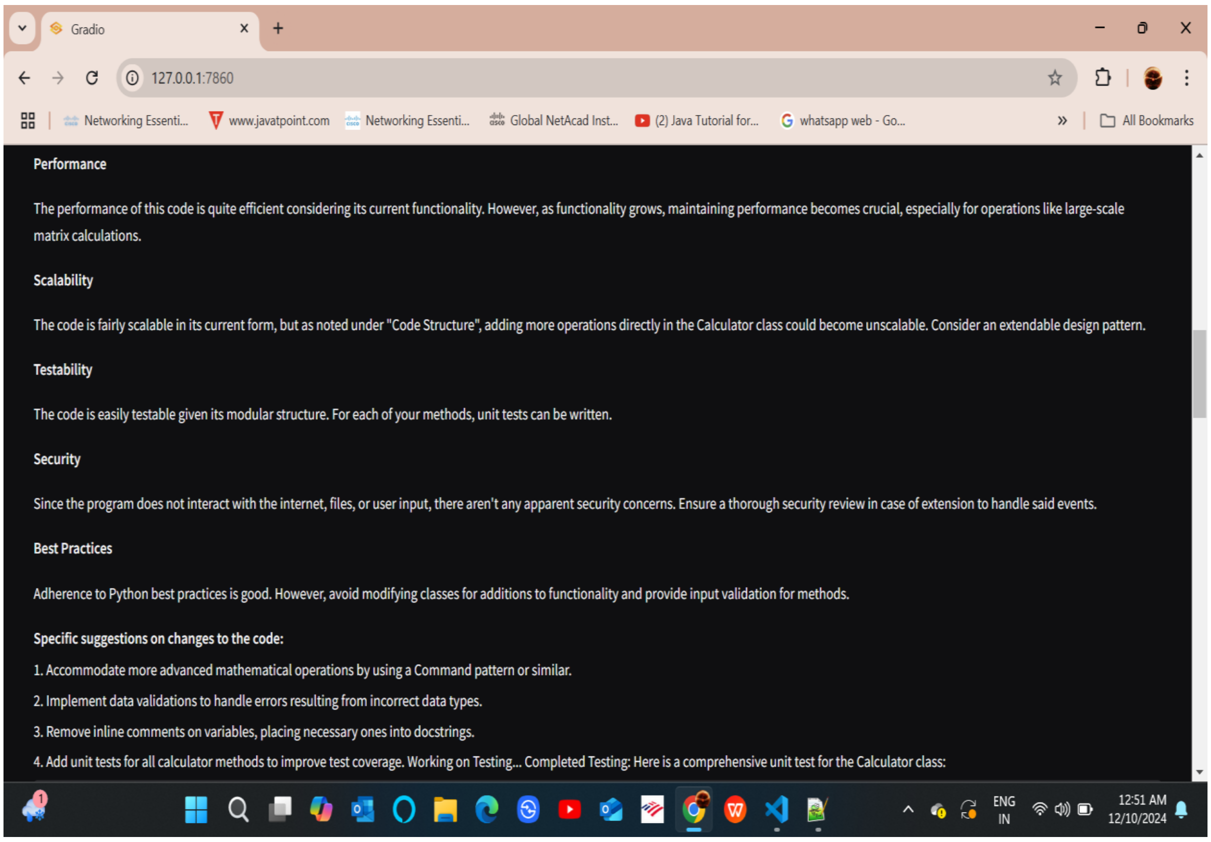
Since the program does not interact with the internet, files, or user input, there aren't any apparent security concerns. Ensure a thorough security review in case of extension to handle said events.

Best Practices

Adherence to Python best practices is good. However, avoid modifying classes for additions to functionality and provide input validation for methods.

Specific suggestions on changes to the code:

1. Accommodate more advanced mathematical operations by using a Command pattern or similar.
2. Implement data validations to handle errors resulting from incorrect data types.
3. Remove inline comments on variables, placing necessary ones into docstrings.
4. Add unit tests for all calculator methods to improve test coverage. Working on Testing...


Performance

The performance of this code is quite efficient considering its current functionality. However, as functionality grows, maintaining performance becomes crucial, especially for operations like large-scale matrix calculations.

Scalability

The code is fairly scalable in its current form, but as noted under "Code Structure", adding more operations directly in the Calculator class could become unscalable. Consider an extendable design pattern.

Testability

The code is easily testable given its modular structure. For each of your methods, unit tests can be written.

Security

Since the program does not interact with the internet, files, or user input, there aren't any apparent security concerns. Ensure a thorough security review in case of extension to handle said events.

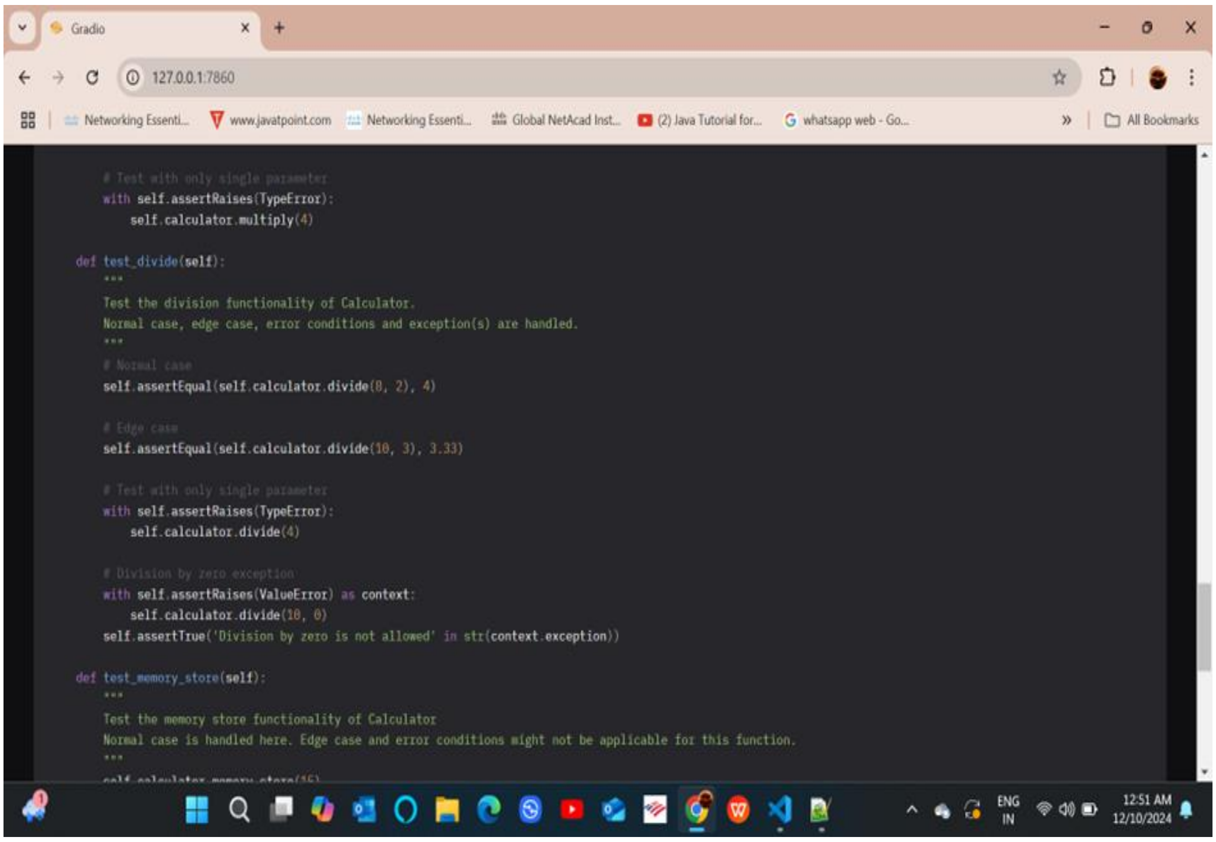
Best Practices

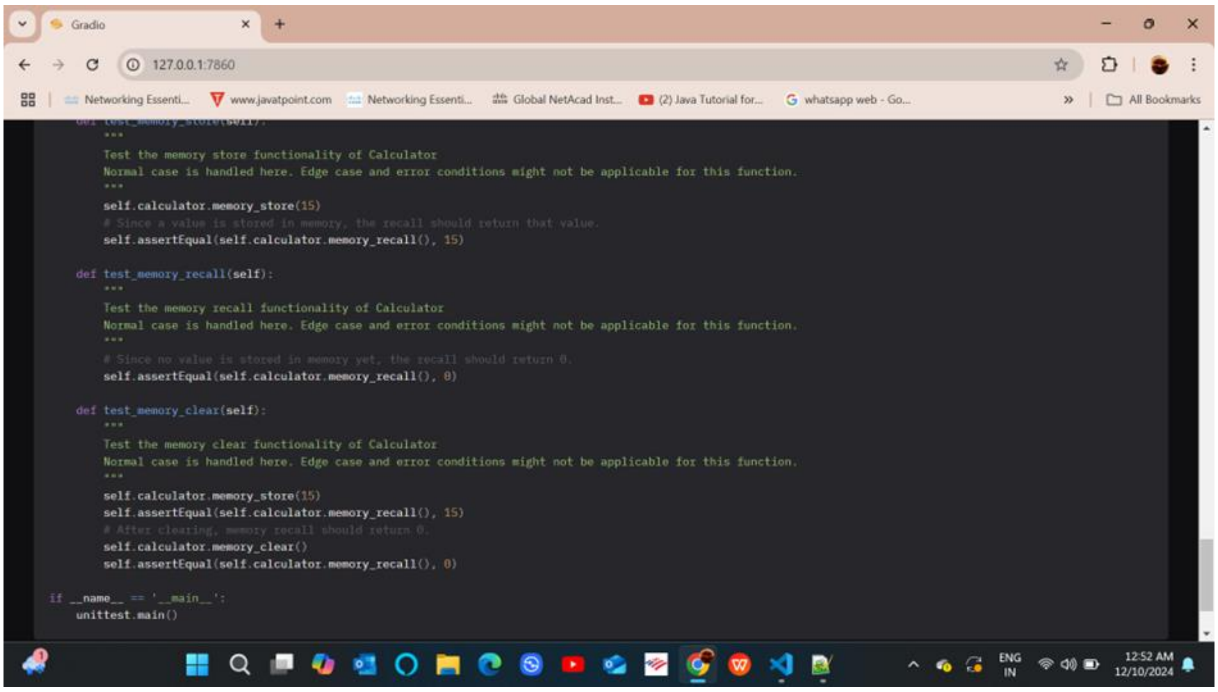
Adherence to Python best practices is good. However, avoid modifying classes for additions to functionality and provide input validation for methods.

Specific suggestions on changes to the code:

1. Accommodate more advanced mathematical operations by using a Command pattern or similar.
2. Implement data validations to handle errors resulting from incorrect data types.
3. Remove inline comments on variables, placing necessary ones into docstrings.
4. Add unit tests for all calculator methods to improve test coverage. Working on Testing... Completed Testing: Here is a comprehensive unit test for the Calculator class:

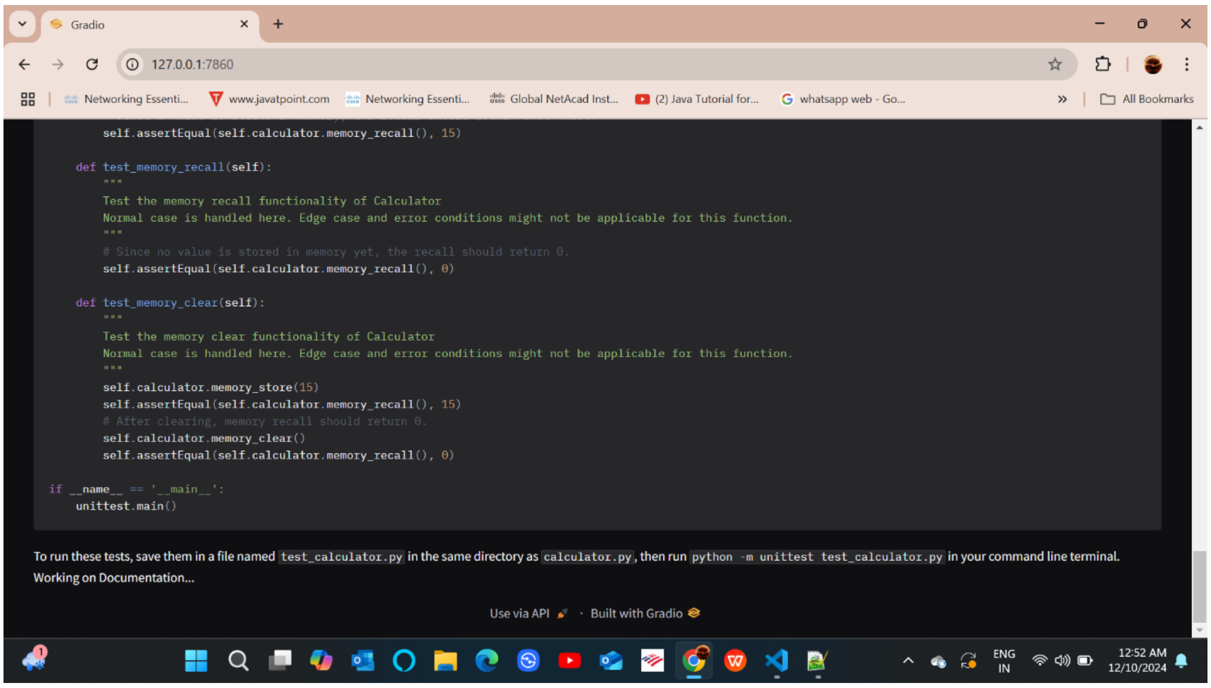
Completed Testing: Working on Unit Test class


```
        # Test with only single parameter
        with self.assertRaises(TypeError):
            self.calculator.multiply(4)

    def test_divide(self):
        """
        Test the division functionality of Calculator.
        Normal case, edge case, error conditions and exception(s) are handled.
        """
        # Normal case
        self.assertEqual(self.calculator.divide(8, 2), 4)

        # Edge case
        self.assertEqual(self.calculator.divide(10, 3), 3.33)

        # Test with only single parameter
        with self.assertRaises(TypeError):
            self.calculator.divide(4)

        # Division by zero exception
        with self.assertRaises(ValueError) as context:
            self.calculator.divide(10, 0)
        self.assertTrue('Division by zero is not allowed' in str(context.exception))

    def test_memory_store(self):
        """
        Test the memory store functionality of Calculator
        Normal case is handled here. Edge case and error conditions might not be applicable for this function.
        """
        self.calculator.memory_store(15)
```



```
    def test_memory_store(self):
        """
        Test the memory store functionality of Calculator
        Normal case is handled here. Edge case and error conditions might not be applicable for this function.
        """
        self.calculator.memory_store(15)
        # Since a value is stored in memory, the recall should return that value.
        self.assertEqual(self.calculator.memory_recall(), 15)

    def test_memory_recall(self):
        """
        Test the memory recall functionality of Calculator
        Normal case is handled here. Edge case and error conditions might not be applicable for this function.
        """
        # Since no value is stored in memory yet, the recall should return 0.
        self.assertEqual(self.calculator.memory_recall(), 0)

    def test_memory_clear(self):
        """
        Test the memory clear functionality of Calculator
        Normal case is handled here. Edge case and error conditions might not be applicable for this function.
        """
        self.calculator.memory_store(15)
        self.assertEqual(self.calculator.memory_recall(), 15)
        # After clearing, memory recall should return 0.
        self.calculator.memory_clear()
        self.assertEqual(self.calculator.memory_recall(), 0)

if __name__ == '__main__':
    unittest.main()
```

Glimpse of Unit Test



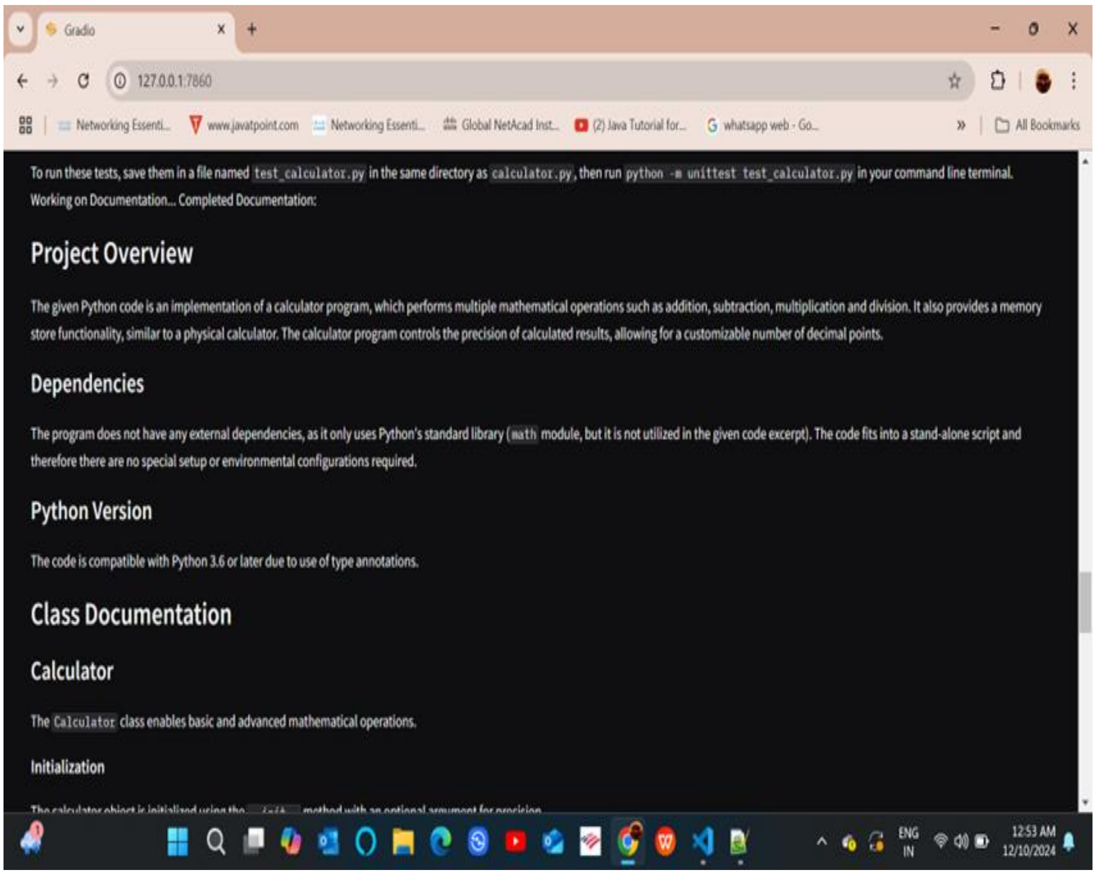
```
        self.assertEqual(self.calculator.memory_recall(), 15)

    def test_memory_recall(self):
        """
        Test the memory recall functionality of Calculator
        Normal case is handled here. Edge case and error conditions might not be applicable for this function.
        """
        # Since no value is stored in memory yet, the recall should return 0.
        self.assertEqual(self.calculator.memory_recall(), 0)

    def test_memory_clear(self):
        """
        Test the memory clear functionality of Calculator
        Normal case is handled here. Edge case and error conditions might not be applicable for this function.
        """
        self.calculator.memory_store(15)
        self.assertEqual(self.calculator.memory_recall(), 15)
        # After clearing, memory recall should return 0.
        self.calculator.memory_clear()
        self.assertEqual(self.calculator.memory_recall(), 0)

if __name__ == '__main__':
    unittest.main()
```

To run these tests, save them in a file named `test_calculator.py` in the same directory as `calculator.py`, then run `python -m unittest test_calculator.py` in your command line terminal.
Working on Documentation...

To run these tests, save them in a file named test_calculator.py in the same directory as calculator.py, then run python - unittest test_calculator.py in your command line terminal.

Working on Documentation...

Documentation



To run these tests, save them in a file named `test_calculator.py` in the same directory as `calculator.py`, then run `python -m unittest test_calculator.py` in your command line terminal.
Working on Documentation... Completed Documentation:

Project Overview

The given Python code is an implementation of a calculator program, which performs multiple mathematical operations such as addition, subtraction, multiplication and division. It also provides a memory store functionality, similar to a physical calculator. The calculator program controls the precision of calculated results, allowing for a customizable number of decimal points.

Dependencies

The program does not have any external dependencies, as it only uses Python's standard library (`math` module, but it is not utilized in the given code excerpt). The code fits into a stand-alone script and therefore there are no special setup or environmental configurations required.

Python Version

The code is compatible with Python 3.6 or later due to use of type annotations.

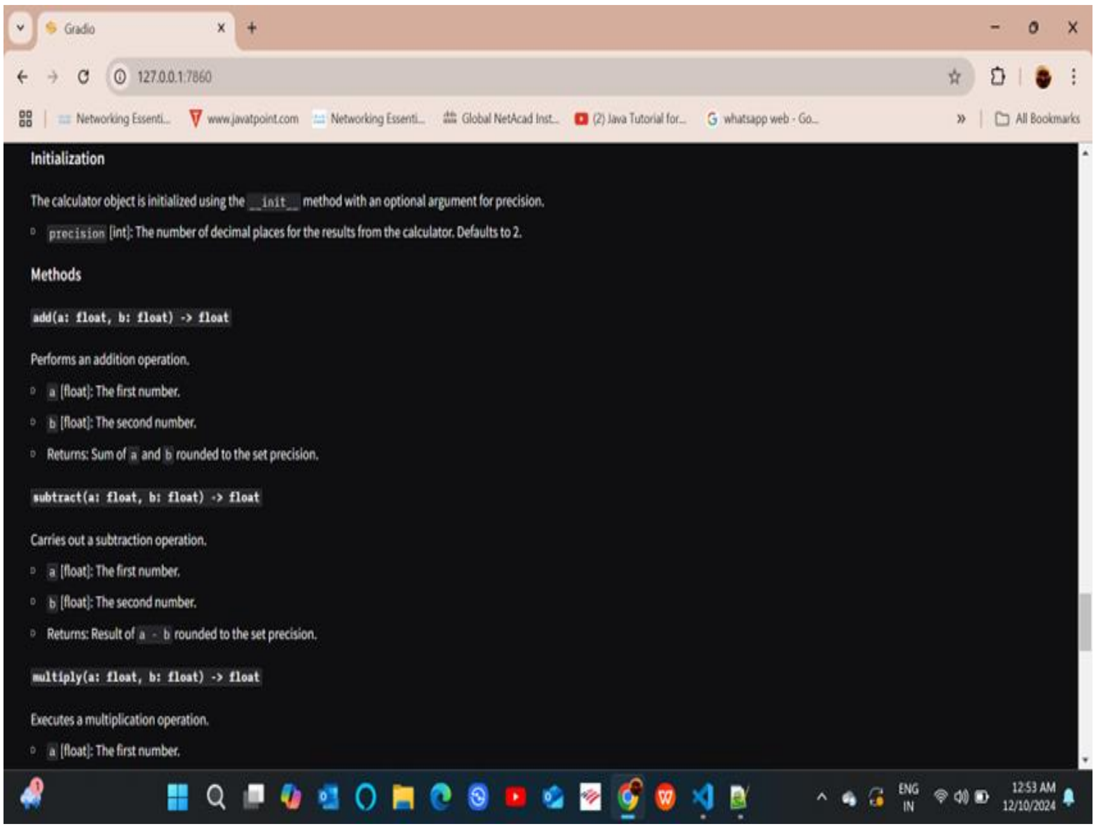
Class Documentation

Calculator

The `Calculator` class enables basic and advanced mathematical operations.

Initialization

The calculator object is initialized using the `__init__` method with an optional argument for precision.



Initialization

The calculator object is initialized using the `__init__` method with an optional argument for precision.

- `precision` [int]: The number of decimal places for the results from the calculator. Defaults to 2.

Methods

`add(a: float, b: float) -> float`

Performs an addition operation.

- `a` [float]: The first number.
- `b` [float]: The second number.
- Returns: Sum of `a` and `b` rounded to the set precision.

`subtract(a: float, b: float) -> float`

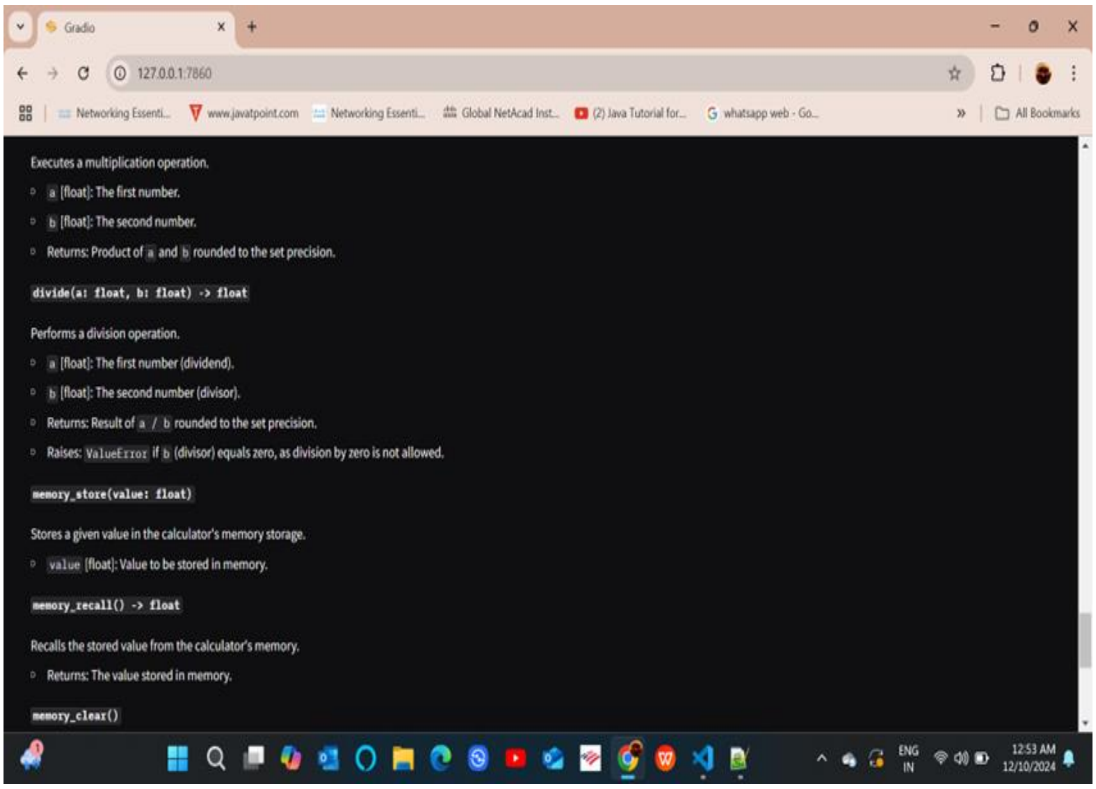
Carries out a subtraction operation.

- `a` [float]: The first number.
- `b` [float]: The second number.
- Returns: Result of `a - b` rounded to the set precision.

`multiply(a: float, b: float) -> float`

Executes a multiplication operation.

- `a` [float]: The first number.


Executes a multiplication operation.

- `a` [float]: The first number.
- `b` [float]: The second number.
- Returns: Product of `a` and `b` rounded to the set precision.

`divide(a: float, b: float) -> float`

Performs a division operation.

- `a` [float]: The first number (dividend).
- `b` [float]: The second number (divisor).
- Returns: Result of `a / b` rounded to the set precision.
- Raises: `ValueError` if `b` (divisor) equals zero, as division by zero is not allowed.

`memory_store(value: float)`

Stores a given value in the calculator's memory storage.

- `value` [float]: Value to be stored in memory.

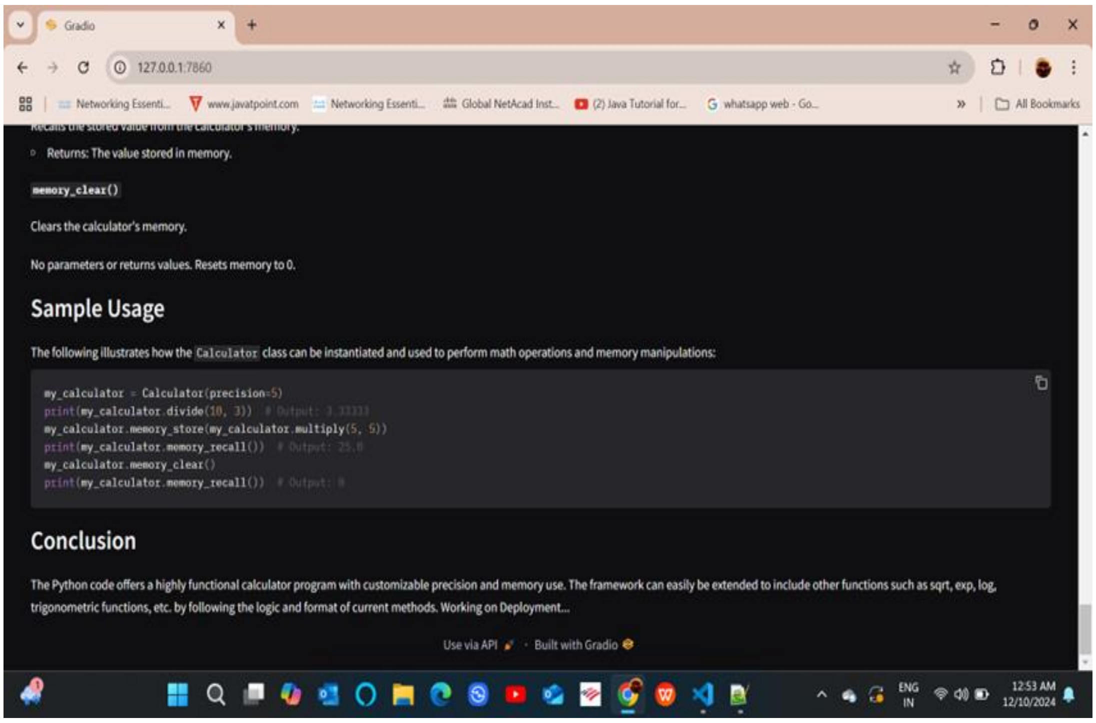
`memory_recall() -> float`

Recalls the stored value from the calculator's memory.

- Returns: The value stored in memory.

`memory_clear()`

The Python code offers a highly functional calculator program with customizable precision and memory use. The framework can easily be extended to include other functions such as sqrt, exp, log, trigonometric functions, etc. by following the logic and format of current methods. Working on Deployment...


- Returns: The value stored in memory.

`memory_clear()`

Clears the calculator's memory.

No parameters or returns values. Resets memory to 0.

Sample Usage

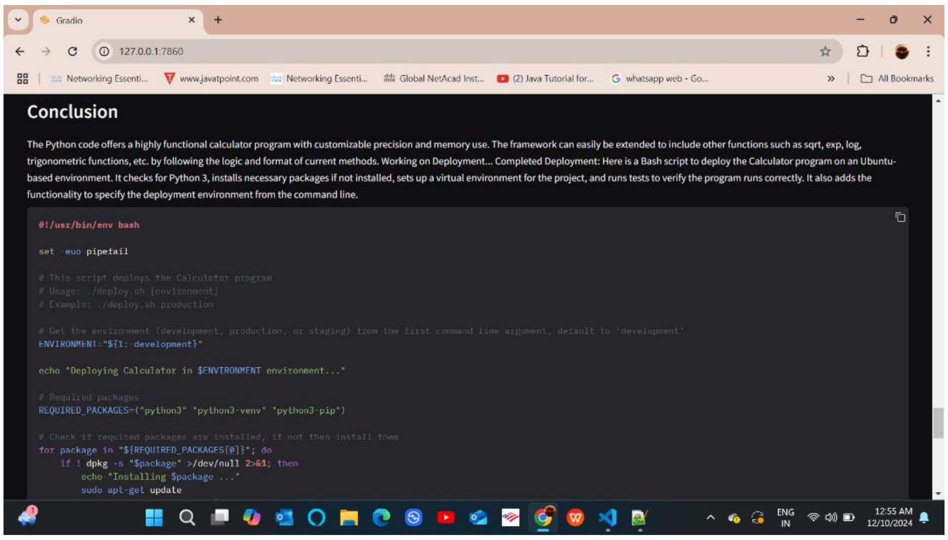
The following illustrates how the `Calculator` class can be instantiated and used to perform math operations and memory manipulations:

```
my_calculator = Calculator(precision=5)
print(my_calculator.divide(10, 3))  # Output: 3.33333
my_calculator.memory_store(my_calculator.multiply(5, 5))
print(my_calculator.memory_recall())  # Output: 25.0
my_calculator.memory_clear()
print(my_calculator.memory_recall())  # Output: 0
```

Conclusion

The Python code offers a highly functional calculator program with customizable precision and memory use. The framework can easily be extended to include other functions such as sqrt, exp, log, trigonometric functions, etc. by following the logic and format of current methods. Working on Deployment...

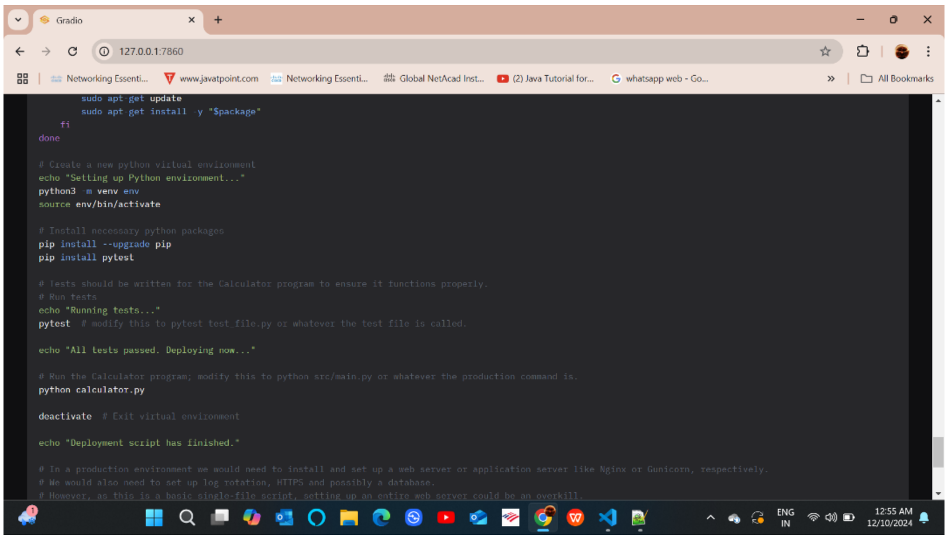
Project Setup



Conclusion

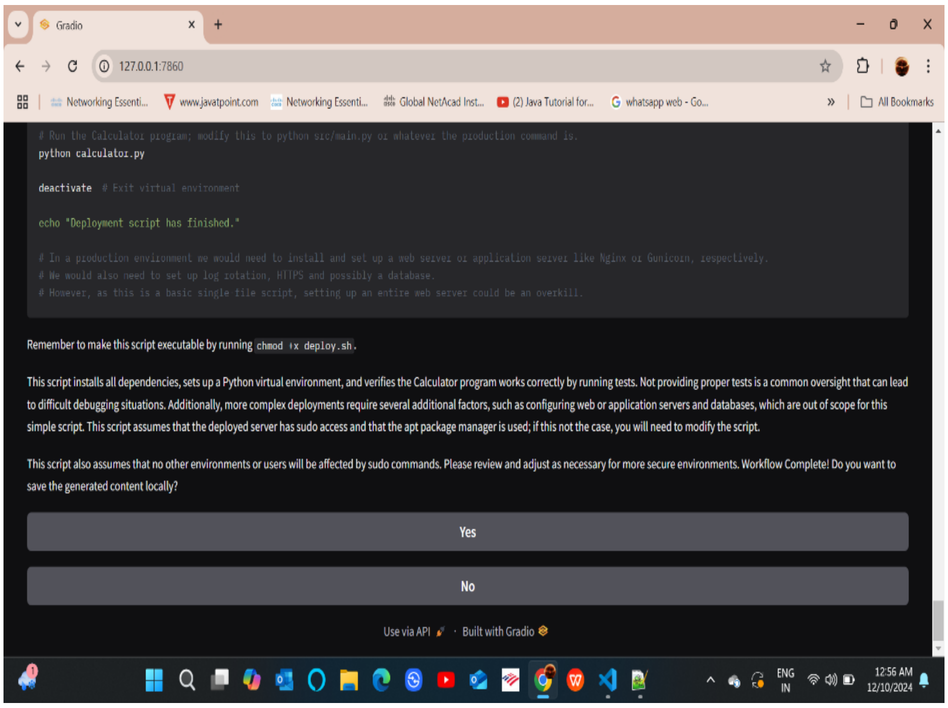
The Python code offers a highly functional calculator program with customizable precision and memory use. The framework can easily be extended to include other functions such as sqrt, exp, log, trigonometric functions, etc. by following the logic and format of current methods. Working on Deployment... Completed Deployment: Here is a Bash script to deploy the Calculator program on an Ubuntu-based environment. It checks for Python 3, installs necessary packages if not installed, sets up a virtual environment for the project, and runs tests to verify the program runs correctly. It also adds the functionality to specify the deployment environment from the command line.

```
#!/usr/bin/env bash

set -euo pipefail

# This script deploys the Calculator program
# Usage: ./deploy.sh [environment]
# Example: ./deploy.sh production

# Get the environment (development, production, or staging) from the first command line argument, default to 'development'
ENVIRONMENT="${1:-development}"

echo "Deploying Calculator in $ENVIRONMENT environment..."

# Required packages
REQUIRED_PACKAGES=("python3" "python3-venv" "python3-pip")

# Check if required packages are installed, if not then install them
for package in "${REQUIRED_PACKAGES[@]}"; do
    if ! dpkg -s "$package" >/dev/null 2>&1; then
        echo "Installing $package ..."
        sudo apt-get update
```



```
        sudo apt-get update
```

```
        sudo apt-get install -y "$package"
    fi
done

# Create a new python virtual environment
echo "Setting up Python environment..."
python3 -m venv env
source env/bin/activate

# Install necessary python packages
pip install --upgrade pip
pip install pytest

# Tests should be written for the Calculator program to ensure it functions properly.
# Run tests
echo "Running tests..."
pytest  # modify this to pytest test_file.py or whatever the test file is called.

echo "All tests passed. Deploying now..."

# Run the Calculator program; modify this to python src/main.py or whatever the production command is.
python calculator.py

deactivate  # Exit virtual environment

echo "Deployment script has finished."

# In a production environment we would need to install and set up a web server or application server like Nginx or Gunicorn, respectively.
# We would also need to set up log rotation, HTTPS and possibly a database.
# However, as this is a basic single-file script, setting up an entire web server could be an overkill.
```



```
# Run the Calculator program; modify this to python src/main.py or whatever the production command is.
python calculator.py

deactivate  # Exit virtual environment

echo "Deployment script has finished."

# In a production environment we would need to install and set up a web server or application server like Nginx or Gunicorn, respectively.
# We would also need to set up log rotation, HTTPS and possibly a database.
# However, as this is a basic single-file script, setting up an entire web server could be an overkill.
```

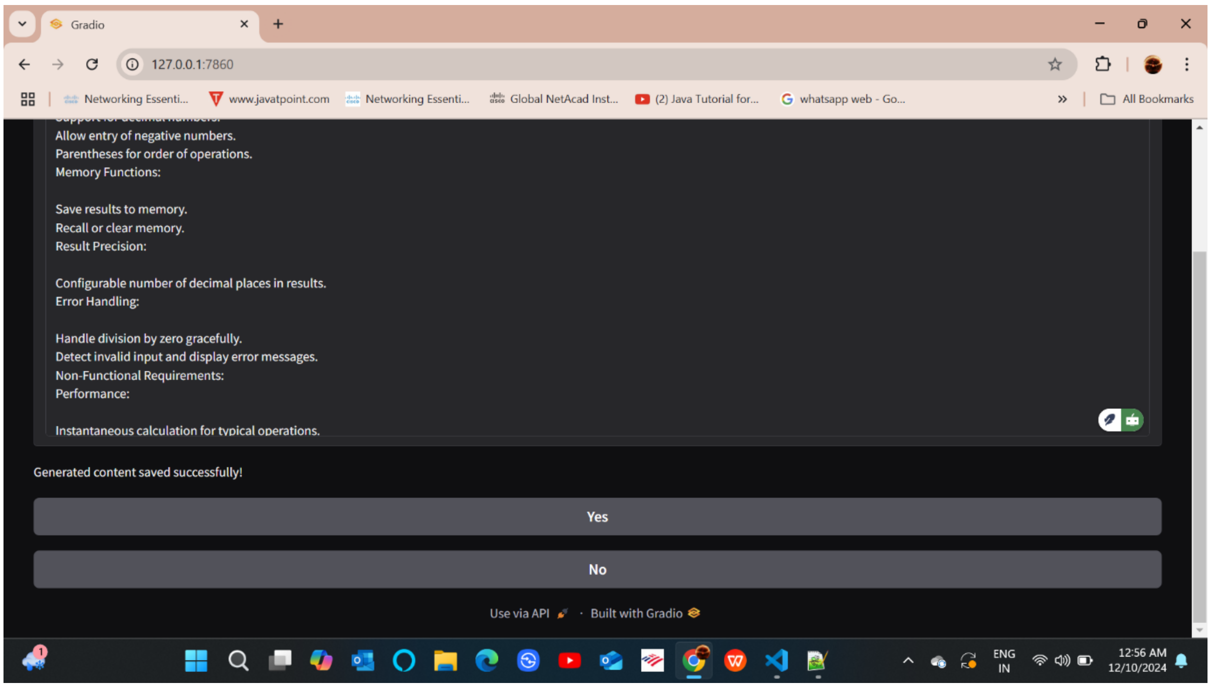
Remember to make this script executable by running `chmod +x deploy.sh`.

This script installs all dependencies, sets up a Python virtual environment, and verifies the Calculator program works correctly by running tests. Not providing proper tests is a common oversight that can lead to difficult debugging situations. Additionally, more complex deployments require several additional factors, such as configuring web or application servers and databases, which are out of scope for this simple script. This script assumes that the deployed server has sudo access and that the apt package manager is used; if this not the case, you will need to modify the script.

This script also assumes that no other environments or users will be affected by sudo commands. Please review and adjust as necessary for more secure environments. Workflow Complete! Do you want to save the generated content locally?

This script installs all dependencies, sets up a Python virtual environment, and verifies the Calculator program works correctly by running tests.

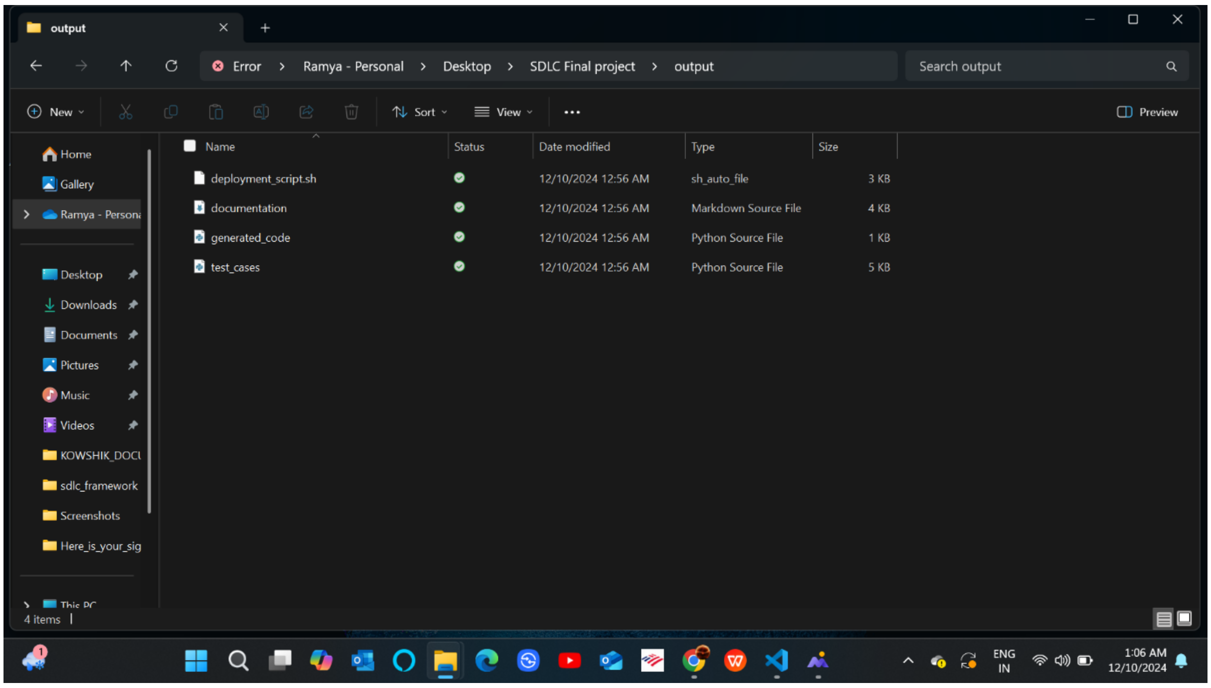
Generated content saved successfully



```
Allow entry of negative numbers.
Parentheses for order of operations.
Memory Functions:

Save results to memory.
Recall or clear memory.
Result Precision:

Configurable number of decimal places in results.
Error Handling:

Handle division by zero gracefully.
Detect invalid input and display error messages.
Non-Functional Requirements:
Performance:

Instantaneous calculation for typical operations.
```

Generated content saved successfully!

Different Output Files



Error > Ramya - Personal > Desktop > SDLC Final project > output

Name	Status	Date modified	Type	Size
deployment_script.sh	✓	12/10/2024 12:56 AM	sh_auto_file	3 KB
documentation	✓	12/10/2024 12:56 AM	Markdown Source File	4 KB
generated_code	✓	12/10/2024 12:56 AM	Python Source File	1 KB
test_cases	✓	12/10/2024 12:56 AM	Python Source File	5 KB

4 items

11. Conclusion

The Software Development Life Cycle (SDLC) Development Using Multi-Agentic Gen AI Framework has proven to be a groundbreaking approach to revolutionizing traditional software development practices. By integrating generative AI and multi-agent systems (MAS), the framework successfully automates critical SDLC phases, including coding, code review, testing, documentation, and deployment. This innovative methodology streamlines workflows, minimizes human intervention, and significantly reduces the errors and inefficiencies typically associated with manual processes.

The framework's design ensures that each agent operates autonomously yet collaboratively, fulfilling specific roles within the SDLC. The integration of LangChain enables seamless communication and task delegation among agents, fostering an adaptable and cohesive system. The evaluation metrics demonstrate considerable improvements in efficiency, accuracy, and scalability, reducing development time while maintaining stringent quality standards. These results highlight the framework's potential to address the growing complexity and demands of modern software projects.

This project not only showcases the transformative power of AI in software engineering but also establishes a robust foundation for future innovations. By reimagining the SDLC through the lens of automation and AI, it addresses critical challenges such as resource limitations, time constraints, and error-prone manual workflows. The framework serves as a testament to the potential of AI-driven automation to enhance the reliability, speed, and quality of software delivery.

Overall, this project represents a significant step forward in adopting AI for software development. By leveraging the capabilities of MAS and generative AI, it demonstrates how innovative methodologies can reshape traditional practices, laying the groundwork for a more efficient and intelligent approach to software engineering.

12. Future work

While the SDLC Development Using Multi-Agentic Gen AI Framework has achieved its objectives, there is substantial scope for enhancement and expansion to make it even more comprehensive and impactful.

One promising direction is the inclusion of additional SDLC phases, such as requirements analysis and user acceptance testing. By integrating AI-driven agents to analyze requirements and validate them against user expectations, the framework could cover the entire lifecycle from inception to deployment, offering an end-to-end solution.

Incorporating advanced AI technologies, such as reinforcement learning for dynamic decision-making, could enhance the system's adaptability. Fine-tuning models tailored to specific domains, such as healthcare, finance, or e-commerce, would enable the agents to handle domain-specific challenges with greater precision. Moreover, integrating state-of-the-art natural language processing models for generating even more intuitive and human-like documentation could significantly improve user experience.

Scalability can be further enhanced by integrating the framework with robust DevOps tools and cloud platforms, enabling it to handle larger, more complex projects with diverse requirements. The inclusion of explainability features, such as detailed logs, performance metrics, and visual dashboards, would not only make the system more transparent but also empower users to understand and trust the AI-driven processes.

Lastly, deploying the framework in real-world industry settings would provide invaluable feedback for refinement. Collaboration with industry partners could highlight practical usability concerns, ensuring the framework is robust, reliable, and ready for widespread adoption. Such real-world testing would pave the way for the system's evolution into a versatile, indispensable tool for automating and optimizing software development workflows. By addressing these areas, the framework could redefine how software is designed, developed, and deployed in the age of AI.

References

[1] Zeeshan Rasheed et al., "CodePori: Large Scale Model for Autonomous Software Development by Using Multi-Agents", https://arxiv.org/abs/2402.01411

[2] Yoichi Ishibashi and Yoshimasa Nishimura, "Self-Organized Agents: A LLM Multi-Agent Framework toward Ultra Large-Scale Code Generation and Optimization", https://arxiv.org/abs/2404.02183

[3] Malik Abdul Sami et al., "Experimenting with Multi-Agent Software Development: Towards a Unified Platform", https://arxiv.org/abs/2406.05381

[4] Jeremy Harper, "AutoGenesisAgent: Self-Generating Multi-Agent Systems for Complex Tasks", https://arxiv.org/abs/2404.17017

[5] Hang Zou et al., "Wireless Multi-Agent Generative AI: From Connected Intelligence to Collective Intelligence", https://arxiv.org/abs/2307.02757

[6] Anh Nguyen-Duc et al., "Generative Artificial Intelligence for Software Engineering -- A Research Agenda", https://arxiv.org/abs/2310.18648

[7] Shunyu Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", https://arxiv.org/abs/2210.03629

[8] Yue Wu et al., "SPRING: GPT-4 Out-performs RL Algorithms by Studying Papers and Reasoning", https://arxiv.org/abs/2305.14859

[9] Zihao Wang et al., "Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents", https://arxiv.org/abs/2302.00923

[10] Noah Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning", https://arxiv.org/abs/2303.11366

[11] Shibo Hao et al., "Reasoning with Language Model is Planning with World Model", https://arxiv.org/abs/2305.16291

[12] Jenny Zhang et al., "OMNI: Open-endedness via Models of human Notions of Interestingness", https://arxiv.org/abs/2306.00924

[13] Joon Sung Park et al., "Generative Agents: Interactive Simulacra of Human Behavior", https://arxiv.org/abs/2304.03442

[14] Microsoft Research, "Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks", https://www.microsoft.com/en-us/research/articles/magentic-one-a-generalist-multi-agent-system-for-solving-complex-tasks/

[15] Forbes Technology Council, "Unlocking Generative AI In Software Development", https://www.forbes.com/sites/forbestechcouncil/2024/12/03/unlocking-generative-ai-in-software-development/

[16] Computerworld, "What are AI agents and why are they now so pervasive?", https://www.computerworld.com/article/3617392/what-are-ai-agents-and-why-are-they-now-so-pervasive.html

[17] Emergence AI, "Emergence AI Debuts Autonomous Multi-Agent Orchestrator - Advancing Web Automation for Enterprise Efficiency", https://www.businesswire.com/news/home/20241205351710/en/Emergence-AI-Debuts-Autonomous-Multi-Agent-Orchestrator---Advancing-Web-Automation-for-Enterprise-Efficiency

[18] DevPro Journal, "Finding Real Value in Generative AI in 2025", https://www.devprojournal.com/software-development-trends/aiops/finding-real-value-in-generative-ai-in-2025/

[19] Bertelsmann Tech, "Harnessing the Power of Multi-Agent Systems for Generative AI (Part 1)", https://tech.bertelsmann.com/en/blog/articles/harnessing-the-power-of-multi-agent-systems-for-generative-ai-part-1

[20] SSRN, "Exploring the Synergy of Generative and Distributed AI in Multi-agent Systems", https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4617662

[21] SpringerLink, "Generative AI | Business & Information Systems Engineering", https://link.springer.com/article/10.1007/s12599-023-00834-7

[22] Wikipedia, "Large language model", https://en.wikipedia.org/wiki/Large_language_model