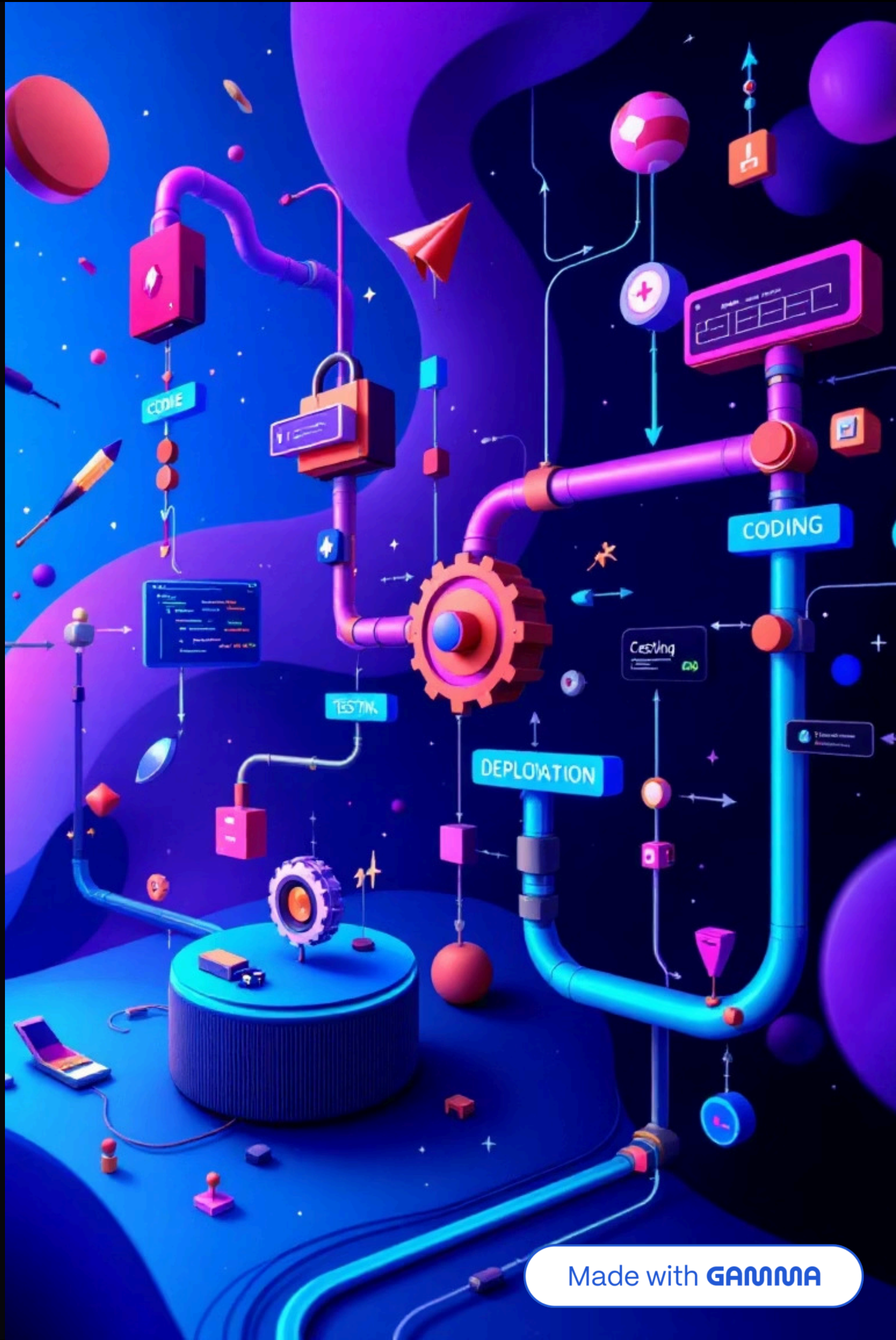
GitHub Pipeline Calculator Project – Project Report

Project Overview

- Python-based calculator application
- Focus on software engineering best practices
- Emphasis on Git workflow, testing, CI, and code quality
- Functionality includes add, subtract, multiply, and divide operations



Project Objectives

Practice feature-based Git workflows

Understand branch creation and merging

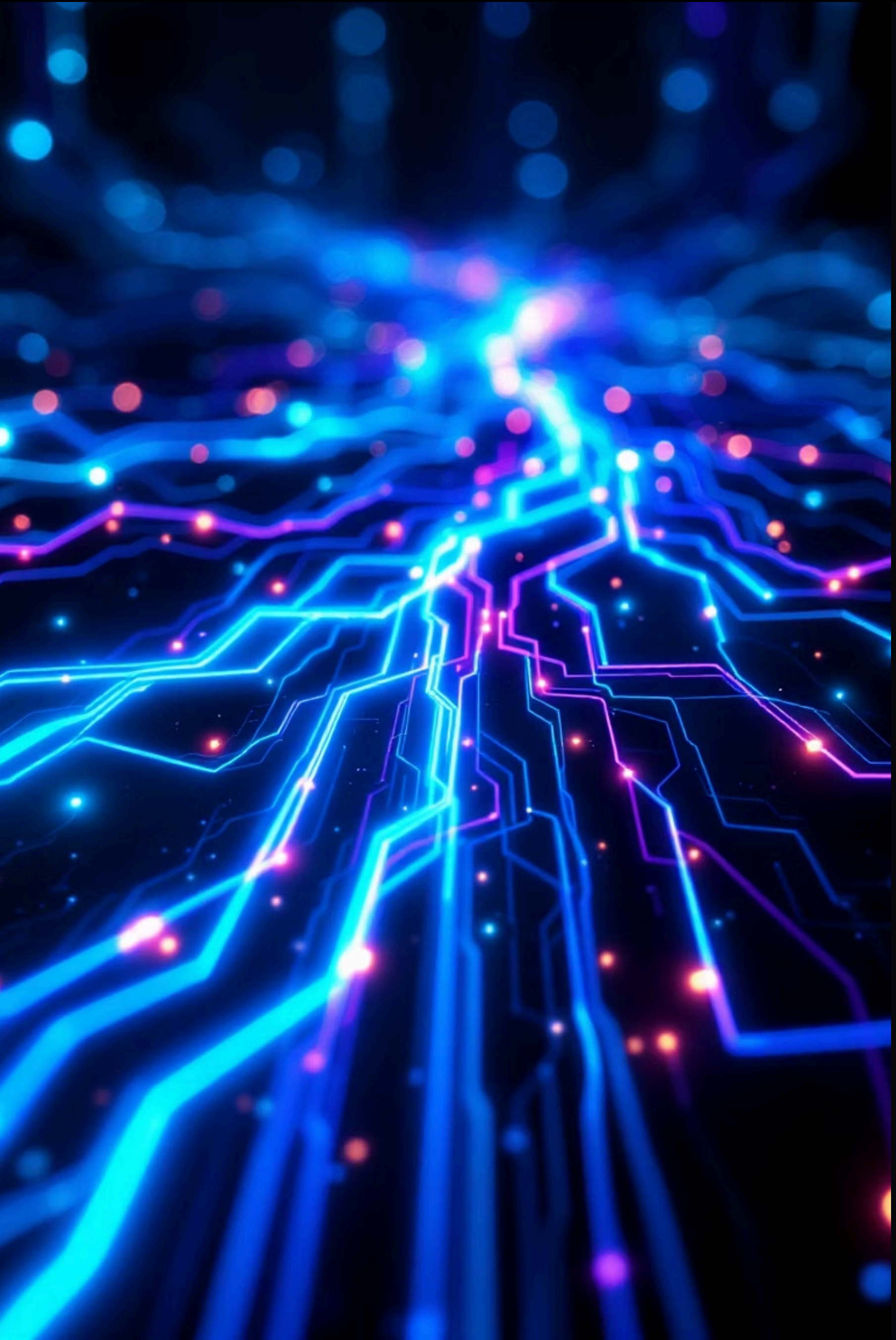
Apply automated unit testing

Use static code analysis tools

Implement Continuous Integration (CI)

Tools and Technologies Used

- Python
- Git and GitHub
- pytest for unit testing
- Pylint for code quality analysis
- GitHub Actions for CI automation



Version Control Strategy

- Used feature-based branching strategy
- main branch always kept stable
- No direct development on main
- Each feature developed in a separate branch
- Feature branches merged only after completion
- Ensured clean and understandable commit history

Feature-Based Development

1

Add operation

Implemented in its own feature branch

2

Subtract operation

Implemented in its own feature branch

3

Multiply operation

Implemented in its own feature branch

4

Divide operation

Implemented in its own feature branch

Each feature branch contained:

- Code changes
- Corresponding test cases

Features merged incrementally into main

Testing Strategy

- Used pytest for unit testing
- Each function tested independently
- One test focuses on one behavior
- Edge cases explicitly tested
- Division by zero tested using exception handling
- Tests executed locally and through CI pipeline

Continuous Integration (CI)

- Implemented using GitHub Actions
- Workflow triggered on:
  - Push to main branch
  - Pull requests targeting main
- CI steps include:
  - Code checkout
  - Python environment setup
  - Dependency installation
  - Automated pytest execution
- Ensures only tested code reaches main

Code Quality with Pylint

Used Pylint to analyze code quality

Generated configuration file using:

```
pylint --generate-rcfile > .pylintrc
```

Disabled selected rules to reduce noise:

- Missing module docstrings (C0114)
- Missing function docstrings (C0116)

Followed industry approach of balancing strictness and readability

Pylint Results

Initial Pylint score was low due to default strict rules

Incremental fixes applied to improve readability

Final Pylint score achieved: **9.13 / 10**

Demonstrates measurable improvement in code quality

Minor remaining warnings were informational, not critical

Key Learnings

- Feature branches must always be created from latest main
- Git merges track file history, not individual functions
- Tests prevent accidental regressions
- CI pipelines enforce discipline automatically
- Linters guide developers but do not replace judgment
- Clean code is about clarity, not perfection

Best Practices Applied

- Feature-based Git workflow
- Incremental development
- Automated testing
- CI enforcement
- Static code analysis
- Clean commit messages
- Clear documentation

Final Outcome

- Fully functional calculator application
- Clean Git history with meaningful commits
- Automated test execution
- CI pipeline integrated successfully
- High code quality score
- Interview-ready portfolio project

Conclusion

- Simple functionality used to demonstrate professional practices
- Strong focus on engineering discipline
- Suitable for learning, portfolio, and interviews
- Demonstrates real-world development workflow clearly