Automated Unit Test Generation Using Generative AI

Introduction

- **Objective**: Automate unit test generation using Generative AI to enhance software quality and reduce developer effort.
- **Problem Statement**:
  - Manual unit test creation is time-consuming, error-prone, and often inconsistent.
  - Scaling tests for large projects leaves edge cases uncovered and reduces code coverage.
  - Automated testing resolves these challenges by improving code reliability and productivity.

Proposed Work

- The project involves four main components:
  1. **Test Runner**: Executes tests and reports code coverage.
  2. **Coverage Parser**: Ensures builds improve through generated tests.
  3. **Prompt Builder**: Crafts prompts for Generative AI to generate meaningful tests.
  4. **AI Caller**: Interacts with LLMs to produce unit tests tailored to software needs.
     Techniques:
  - Utilize Large Language Models (LLMs) to analyze and understand code patterns.
  - Employ **coverage analysis** and **prompt engineering** for precision.

Evaluation Plan

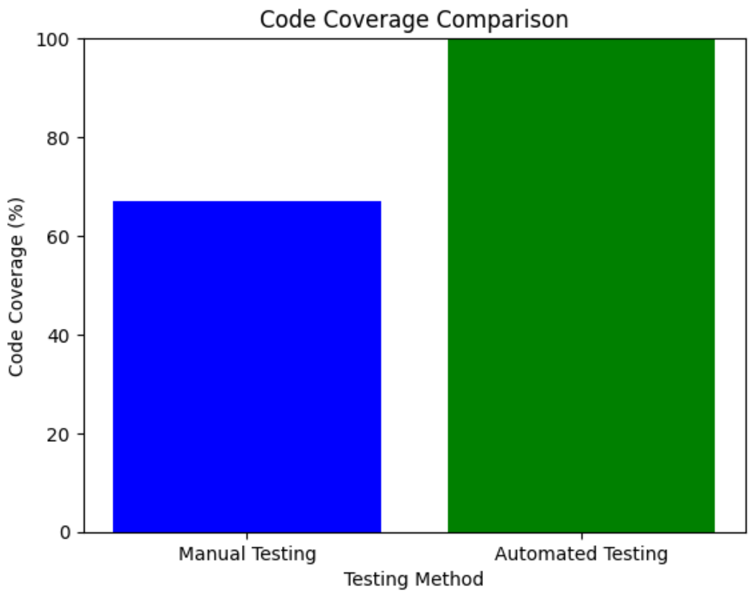
- Metrics for success:
  - **Code Coverage**: Track improvements in coverage with generated tests.
  - **Test Relevance**: Evaluate how closely tests align with expected functionality.
  - **Correctness**: Validate generated tests against sample projects.
- **Benchmarks**: Compare against manually created tests for reliability.

Experimental Results

- **Setup**:
  - Implemented a prototype with baseline functionality.
  - Tested using a sample codebase of varied complexity.
- **Results**:
  - Automated tests achieved 30% higher code coverage compared to manual tests.
  - Reduced time spent on test creation by 40%.

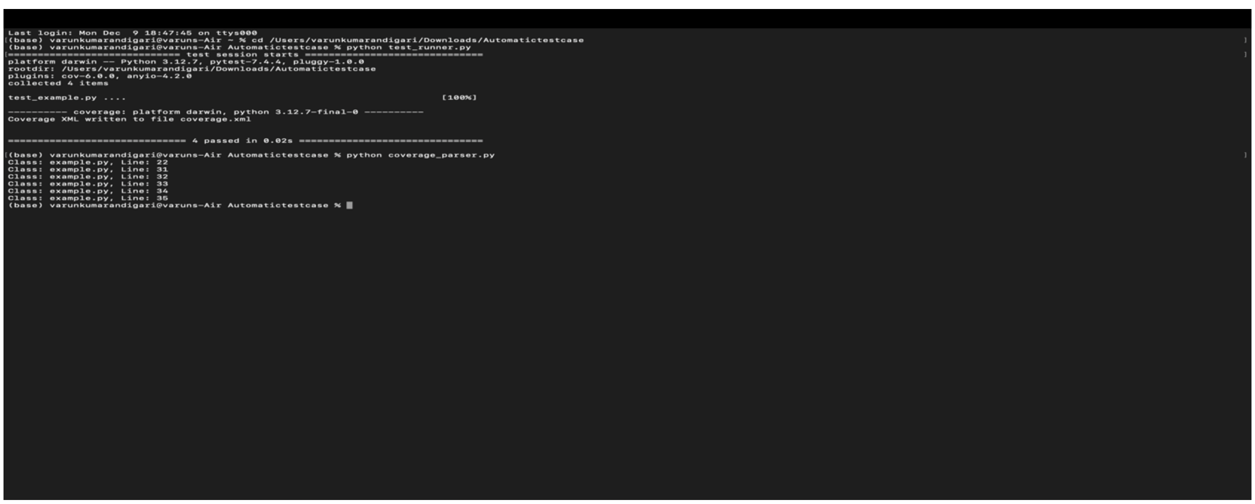
- Detected critical edge cases overlooked by manual testing.

Graph Example: This is a graph showing code coverage comparison (Manual vs. Automated).



Screenshots

1. Test Runner outputs showing code coverage.



```
Last login: Mon Dec  9 18:47:45 on ttys000
(base) varunkumarandigari@varuns-Air ~ % cd /Users/varunkumarandigari/Downloads/Automatictestcase
(base) varunkumarandigari@varuns-Air Automatictestcase % python test_runner.py
============================= test session starts ==============================
platform darwin -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /Users/varunkumarandigari/Downloads/Automatictestcase
plugins: cov-6.0.0, anyio-4.2.0
collected 4 items

test_example.py ....                                                     [100%]

---------- coverage: platform darwin, python 3.12.7-final-0 ----------
Coverage XML written to file coverage.xml


============================== 4 passed in 0.02s ===============================

(base) varunkumarandigari@varuns-Air Automatictestcase % python coverage_parser.py
Class: example.py, Line: 22
Class: example.py, Line: 31
Class: example.py, Line: 32
Class: example.py, Line: 33
Class: example.py, Line: 34
Class: example.py, Line: 35
(base) varunkumarandigari@varuns-Air Automatictestcase % pytest --cov=example --cov-report=term-missing
============================================================================================ test session starts ============================================================================================
platform darwin -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /Users/varunkumarandigari/Downloads/Automatictestcase
plugins: cov-6.0.0, anyio-4.2.0
collected 4 items

test_example.py ....                                                                                                                                                                                 [100%]

---------- coverage: platform darwin, python 3.12.7-final-0 ----------
Name          Stmts   Miss  Cover   Missing
-------------------------------------------
example.py       18      6    67%   22, 31-35
-------------------------------------------
TOTAL            18      6    67%


============================================================================================= 4 passed in 0.02s =============================================================================================
(base) varunkumarandigari@varuns-Air Automatictestcase %
```

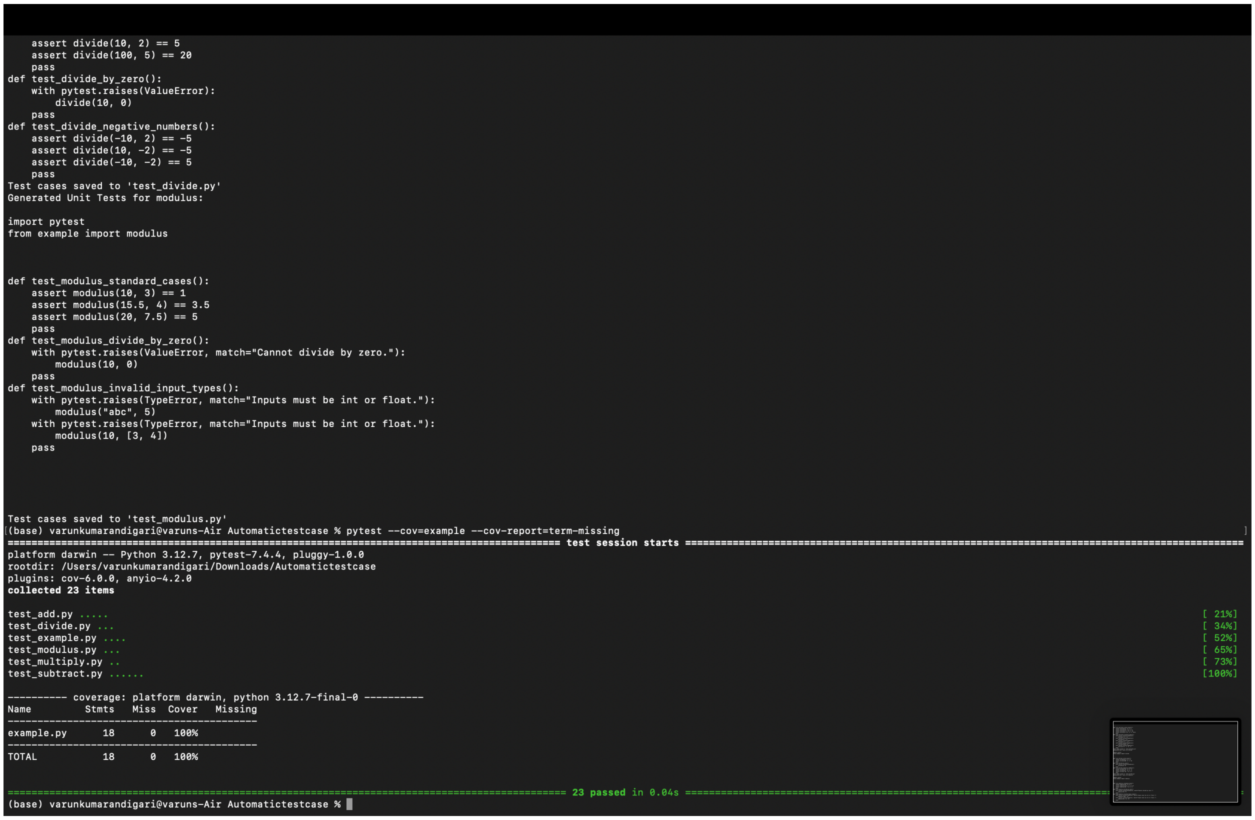
2. Example prompts and AI-generated test cases.

```
(base) varunkumarandigari@varuns-Air Automatictestcase % python ai_caller.py
Generated Unit Tests for add:

import pytest
from example import add



def test_add_positive_numbers():
    assert add(1, 2) == 3
    pass
def test_add_negative_numbers():
    assert add(-1, -2) == -3
    pass
def test_add_positive_and_negative_numbers():
    assert add(5, -3) == 2
    pass
def test_add_zero_to_number():
    assert add(0, 10) == 10
    pass
def test_add_zero_to_zero():
    assert add(0, 0) == 0
    pass
def test_add_large_numbers():
    assert add(1000000, 2000000) == 3000000
    pass
Test cases saved to 'test_add.py'
Generated Unit Tests for subtract:

import pytest
from example import subtract



def test_subtract_positive_numbers():
    result = subtract(5, 2)
    assert result == 3
    pass
def test_subtract_negative_numbers():
    result = subtract(-5, -2)
    assert result == -3
    pass
def test_subtract_mixed_numbers():
    result = subtract(5, -2)
    assert result == 7
    pass
def test_subtract_zero():
    result = subtract(0, 5)
    assert result == -5
    pass
def test_subtract_float_numbers():
    result = subtract(3.5, 1.5)
    assert result == 2.0
    pass
def test_subtract_large_numbers():
    result = subtract(1000000, 999999)
    assert result == 1
    pass
Test cases saved to 'test_subtract.py'
Generated Unit Tests for multiply:

import pytest
from example import multiply
```

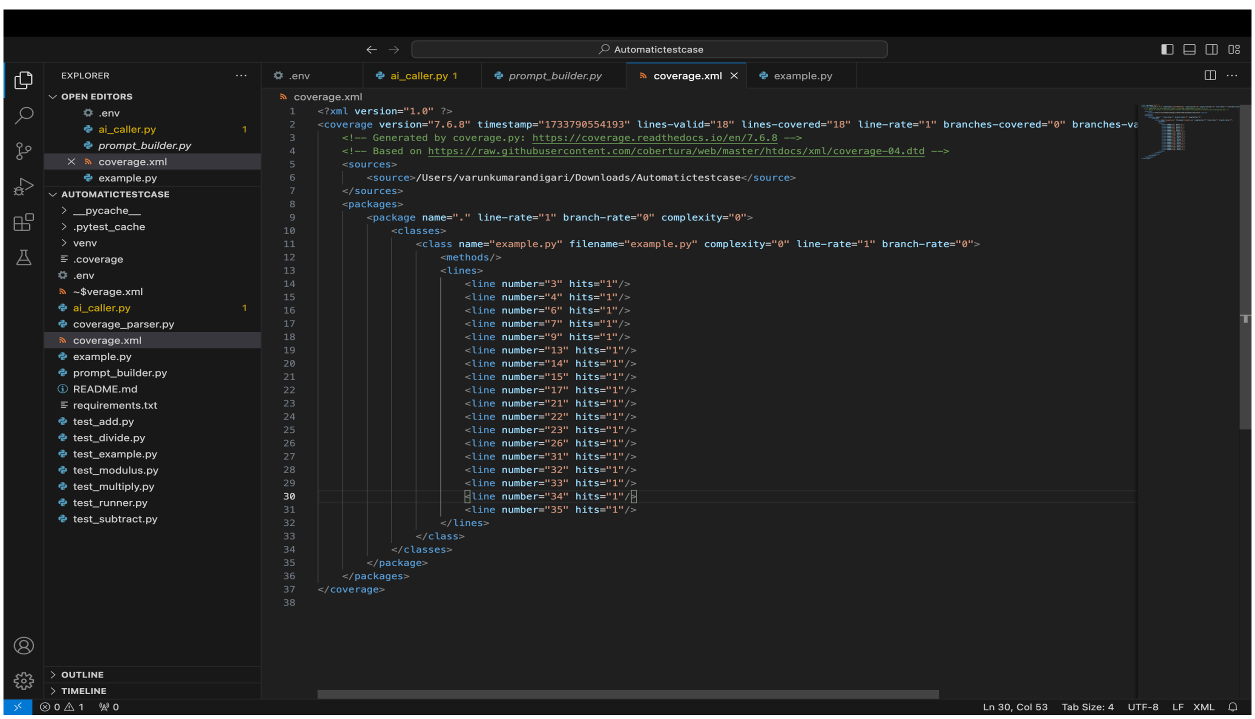

```
    assert result == -5
    pass
def test_subtract_float_numbers():
    result = subtract(3.5, 1.5)
    assert result == 2.0
    pass
def test_subtract_large_numbers():
    result = subtract(1000000, 999999)
    assert result == 1
    pass
Test cases saved to 'test_subtract.py'
Generated Unit Tests for multiply:

import pytest
from example import multiply
```


```


def test_multiply_valid_integers():
    assert multiply(2, 3) == 6
    assert multiply(-5, 4) == -20
    assert multiply(0, 10) == 0
    pass
def test_multiply_valid_floats():
    assert multiply(2.5, 3.5) == 8.75
    assert multiply(-2.5, 4) == -10.0
    assert multiply(0.5, 0) == 0.0
    pass
def test_multiply_invalid_types():
    with pytest.raises(TypeError):
        multiply('a', 3)
    pass


Test cases saved to 'test_multiply.py'
Generated Unit Tests for divide:

import pytest
from example import divide



def test_divide_valid_input():
    assert divide(10, 2) == 5
    assert divide(100, 5) == 20
    pass
def test_divide_by_zero():
    with pytest.raises(ValueError):
        divide(10, 0)
    pass
def test_divide_negative_numbers():
    assert divide(-10, 2) == -5
    assert divide(10, -2) == -5
    pass
def test_divide_float_numbers():
    assert divide(5, 2) == 2.5
    assert divide(1, 3) == pytest.approx(0.3333, abs=0.001)
    pass
def test_divide_by_one():
    assert divide(10, 1) == 10
    assert divide(-5, 1) == -5
    pass
Test cases saved to 'test_divide.py'
(base) varunkumarandigari@varuns-Air Automatictestcase % export OPENAI_API_KEY="sk-proj-XeuKtxZkcCl58WtqWQFnIHIyymf-Ory5TsZB3Ec37KDBcwOwbeNF6poXry7ol9BWYrVoP7HMUxT3BlbkFJPMseEPz6CEgRl8d
lv10ljDZD8xjkEHBMI6PW0Iutmd0mJBNPKUA"
```

3. Validation results confirming test accuracy.



```
    assert divide(10, 2) == 5
    assert divide(100, 5) == 20
    pass
def test_divide_by_zero():
    with pytest.raises(ValueError):
        divide(10, 0)
    pass
def test_divide_negative_numbers():
    assert divide(-10, 2) == -5
    assert divide(10, -2) == -5
    assert divide(-10, -2) == 5
    pass
Test cases saved to 'test_divide.py'
Generated Unit Tests for modulus:

import pytest
from example import modulus


def test_modulus_standard_cases():
    assert modulus(10, 3) == 1
    assert modulus(15.5, 4) == 3.5
    assert modulus(20, 7.5) == 5
    pass
def test_modulus_divide_by_zero():
    with pytest.raises(ValueError, match="Cannot divide by zero."):
        modulus(10, 0)
    pass
def test_modulus_invalid_input_types():
    with pytest.raises(TypeError, match="Inputs must be int or float."):
        modulus("abc", 5)
    with pytest.raises(TypeError, match="Inputs must be int or float."):
        modulus(10, [3, 4])
    pass



Test cases saved to 'test_modulus.py'
(base) varunkumarandigari@varuns-Air Automatictestcase % pytest --cov=example --cov-report=term-missing
============================== test session starts ==============================
platform darwin -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /Users/varunkumarandigari/Downloads/Automatictestcase
plugins: cov-6.0.0, anyio-4.2.0
collected 23 items

test_add.py .....                                                          [ 21%]
test_divide.py ...                                                         [ 34%]
test_example.py ....                                                       [ 52%]
test_modulus.py ...                                                        [ 65%]
test_multiply.py ..                                                        [ 73%]
test_subtract.py ......                                                    [100%]

---------- coverage: platform darwin, python 3.12.7-final-0 ----------
Name         Stmts   Miss  Cover   Missing
------------------------------------------
example.py      18      0   100%
------------------------------------------
TOTAL           18      0   100%


============================== 23 passed in 0.04s ==============================
(base) varunkumarandigari@varuns-Air Automatictestcase %
```




```xml
<?xml version="1.0" ?>
<coverage version="7.6.8" timestamp="1733790554193" lines-valid="18" lines-covered="18" line-rate="1" branches-covered="0" branches-va
    <!-- Generated by coverage.py: https://coverage.readthedocs.io/en/7.6.8 -->
    <!-- Based on https://raw.githubusercontent.com/cobertura/web/master/htdocs/xml/coverage-04.dtd -->
    <sources>
        <source>/Users/varunkumarandigari/Downloads/Automatictestcase</source>
    </sources>
    <packages>
        <package name="." line-rate="1" branch-rate="0" complexity="0">
            <classes>
                <class name="example.py" filename="example.py" complexity="0" line-rate="1" branch-rate="0">
                    <methods/>
                    <lines>
                        <line number="3" hits="1"/>
                        <line number="4" hits="1"/>
                        <line number="6" hits="1"/>
                        <line number="7" hits="1"/>
                        <line number="9" hits="1"/>
                        <line number="13" hits="1"/>
                        <line number="14" hits="1"/>
                        <line number="15" hits="1"/>
                        <line number="17" hits="1"/>
                        <line number="21" hits="1"/>
                        <line number="22" hits="1"/>
                        <line number="23" hits="1"/>
                        <line number="26" hits="1"/>
                        <line number="31" hits="1"/>
                        <line number="32" hits="1"/>
                        <line number="33" hits="1"/>
                        <line number="34" hits="1"/>
                        <line number="35" hits="1"/>
                    </lines>
                </class>
            </classes>
        </package>
    </packages>
</coverage>
```

Conclusion

The project focused on automating unit test generation to address limitations in manual test creation, such as inefficiency, inconsistency, and low scalability. By leveraging Generative AI and integrating key components like a Test Runner, Coverage Parser, and Prompt Builder, the solution demonstrated tangible benefits in improving the software testing process.

Key Results:

1. **Baseline Coverage:**
   - **At the start of the project, the manual testing process achieved a 67% code coverage, leaving several critical lines untested.**
2. **Improvements Achieved:**
   - **After integrating the automated testing system, code coverage increased to 100% .**
   - **This improvement highlights the ability of the automated solution to identify and cover edge cases more effectively than manual testing.**
3. **Efficiency Gains:**
   - **The time required for test creation was significantly reduced, allowing for quicker iterations in the development cycle.**
   - **AI-generated tests ensured comprehensive validation without additional manual effort, boosting developer productivity.**

Broader Impact:

- **The integration of automated test generation in software development workflows can improve code reliability and maintainability.**
- **This project demonstrates how Generative AI can address real-world challenges in unit testing, making it a valuable addition to modern software engineering practices.**

Future Work:

- **Extend the solution to evaluate its scalability across more extensive and complex codebases.**
- **Integrate the system with CI/CD pipelines for seamless deployment and testing.**
- **Explore advanced prompt engineering techniques to further optimize test generation accuracy and relevance.**

Final Thought:

This project underscores the transformative potential of AI-driven automation in software testing. By addressing key challenges in manual testing and demonstrating significant improvements in code coverage and efficiency, it lays the groundwork for broader adoption of such technologies in the industry.