Automated Test Case Generation using AI

1. Introduction

Context: Software testing is essential for ensuring software quality.

Problem: Manual testing is time-consuming, error-prone, and often incomplete.

Research Question: To what extent can AI automate or enhance test case generation?

Goal: Build a prototype and compare traditional vs. AI-based testing approaches.

2. Background & Fundamentals

2.1 Basics of Software Testing

- Types of testing: Unit, Integration, System, Regression
- Test design techniques: Equivalence partitioning, Boundary value analysis, etc.

2.2 Challenges in Manual Testing

- Time-consuming
- Limited test coverage
- Human error

2.3 Artificial Intelligence in Testing Automation

- Machine Learning (ML): Learns from historical bugs and code changes
- Natural Language Processing (NLP): Generates test cases from descriptions
- Large Language Models (LLMs): ChatGPT, CodeT5, StarCoder

3. Related Work

Manual Tools:

- unittest, pytest, JUnit

AI-Based Tools:

- Pynguin Automated unit test generation
- EvoSuite Genetic test generation for Java
- OpenAI ChatGPT / Code Interpreter
- Hugging Face / Transformer Models: CodeT5, StarCoder, CodeGen

Academic Research:

- Comparative studies: AI vs. manual testing
- Metrics: Bug detection, test quality, developer experience

4. Case Study

4.1 Setup

- Language: Python
- Tools: pytest, Pynguin, OpenAI API,Hugging Face models

4.2 Codebase

- Small open-source application

4.3 Test Generation

- Manual: Traditional unit tests with pytest
- Pynguin: Auto-generated white-box tests
- ChatGPT / LLMs: Prompt-based generation

  Transformer-based tests using CodeT5 / StarCoder

4.4 Implementation

4.4.1 Manual Test Cases: Structure and examples

4.4.2 Pynguin: Usage, test output, customization

4.4.3 ChatGPT & Transformers: Prompts, examples, evaluation

5. Evaluation & Comparison

- Code Coverage: Use coverage.py
- Bug Detection: Insert sample bugs
- Effort & Time: Manual vs. AI-based
- Code Quality: Readability, maintainability
- Limitations: Constraints of AI testing

6. Discussion

- Interpretation of results
- Practical implications
- When AI-based testing is usefuland when its not

7. Conclusion & Future Work

- Summary of findings
- Answer to the research question
- Future directions: UI testing, integration testing, better prompts, model tuning

Appendix / Tools Overview

Manual Tools: pytest, unittest

AI Tools: Pynguin, ChatGPT, Transformer Models (CodeT5, StarCoder)

Measurement: coverage.py, test count, bug detection, time, quality