Pynguin-MVP – Automated Unit Test Generation for Python

Inspired by the research paper 'Pynguin: Automated Unit Test Generation for Python'.
This project implements a lightweight version that analyzes Python modules, generates random test cases, executes them, measures coverage, and exports pytest files.

1. Introduction

Goal: Automatically generate Python unit tests that achieve high code coverage.
Motivation: Reduce manual testing effort and improve software reliability.

2. Implementation Overview

Language & Tools: Python 3.11, pytest, coverage.py
Key modules:
• analysis.py – discovers functions and type hints
• ir.py – internal representation of tests
• generators.py – creates random arguments
• exec_cov.py – runs tests and collects coverage
• search_random.py – generation loop
• exporter_pytest.py – pytest file writer
• cli.py – command line interface

3. Demo Workflow

1. Target module: triangle.py
2. Run: pynguin-mvp --project-path . --module-name triangle --iters 200
3. Run pytest on generated tests
4. View out/coverage.json

4. Results Summary

Functions discovered: 1
Tests kept: 1–2
Lines covered: ≈ 40 %
Assertions: auto-generated
Status: All tests passed.

5. Screenshots

Add your screenshots here (showing CLI runs, test file, coverage output, etc).

6. Challenges

• Coverage path mismatch under WSL
• Dynamic execution issues with coverage.py
• Fix attempts included tracer filters and coverage.py integration

7. Conclusion

Pynguin-MVP demonstrates the core concepts of automated test generation as described in the Pynguin paper.
Future work: improve coverage tracking, add branch coverage and mutation-based assertions.

8. References

• Lukasczyk et al., 'Pynguin: Automated Unit Test Generation for Python', ICSE 2021
• Pynguin GitHub – https://github.com/se2p/pynguin
• Python docs: inspect, coverage, trace, pytest