Exercise 2: Automated Testing & Coverage

Name: Lavanika Srinivasaraghavan
Course: CS 520 - Fall 2025

GitHub Repository:
https://github.com/lava-nika/CS520-Exercise2-Automated-Testing-and-Coverage

---

Part 1: Baseline Coverage Analysis

This is the baseline code coverage analysis for all 180 generated solutions from Exercise 1. Coverage was measured using Python's `coverage.py` tool and executing HumanEval tests against each solution.

Setup

- **Tool**: coverage.py with branch analysis enabled (`--branch` flag)
- **Test Suite**: HumanEval benchmark tests
- **Solutions Analyzed**: 180 solutions across 10 problems
  - 2 Models: Claude Sonnet 4.5, GPT-5
  - 3 Strategies: Chain-of-Thought (CoT), Self-Debug, Test-Driven Specification (TDS)
  - 3 Samples per strategy × 2 models × 10 problems = 180 total solutions

---

Baseline Coverage Table

Note:

1. Problem 8 - All 18 solutions failed due to numerical precision issues (required 1e-10 tolerance, used 1e-6). So high coverage does not guarantee correctness.
2. Function naming errors like in problem 1 are not detectable by coverage alone.

Problem	HumanEval ID	Function Name	Tests Passed	Line Coverage	Branch Coverage	Notes
problem1	HumanEval/0	has_close_elements	16/18	85.8%	83.8%	2 failures: function naming errors
problem2	HumanEval/1	separate_paren_groups	18/18	88.2%	81.8%	All passed; good coverage
problem3	HumanEval/31	is_prime	18/18	86.4%	84.6%	All passed; good coverage
problem4	HumanEval/10	make_palindrome	18/18	87.4%	83.1%	All passed; good coverage
problem5	HumanEval/54	same_chars	18/18	93.4%	61.9%	All passed; low branch coverage
problem6	HumanEval/61	correct_bracketing	18/18	91.3%	86.6%	All passed; strong coverage
problem7	HumanEval/108	count_nums	18/18	97.6%	96.5%	All passed; excellent coverage
problem8	HumanEval/32	find_zero	0/18	0.0%	N/A	All failed: precision issues
problem9	HumanEval/105	by_length	18/18	97.8%	91.5%	All passed; excellent coverage
problem10	HumanEval/163	generate_integers	18/18	92.2%	78.1%	All passed; good coverage
Average	—	—	160/180 (88.9%)	91.2%	83.1%	—

Part 2: LLM-Assisted Test Generation & Coverage Improvement

I analyzed the 10 HumanEval problems and chose 2 problems using the following criteria:

1. Largest coverage gaps to maximize improvement
2. Algorithmic complexity
3. Diverse complexity types like exception handling and loops

Problem 10: `by_length` (HumanEval/105)

- Baseline coverage = 92.2% line, 78.1% branch - largest gap of 21.9%.
- Why: 20-line solution with try-except block, filtering, sorting and mapping

Problem 4: `make_palindrome` (HumanEval/10)

- Baseline coverage = 87.4% line, 83.1% branch - (16.9% gap)
- Why: Algorithm with helper function, while loop and edge cases.

Results:

- **Problem 10** (by_length): 78.1% –> 100% branch coverage (+21.9%) (Largest improvement gap)
- **Problem 4** (make_palindrome): 83.1% –> 100% branch coverage (+16.9%)
- **Total tests generated**: 14 (7 per problem)
- **Convergence**: Immediate (100% achieved in iteration 1)
- **LLM used**: Gemini 2.5 Flash

Prompts used to generate tests

Problem 10 prompt

```
Improve branch coverage for this integer filtering and mapping function.

Current baseline: 92.2% line, 78.1% branch

Function: Filter integers 1-9 from array, sort descending, map to word names.
Implementation: Uses dictionary lookup with try-except for invalid integers.

Generate 5-7 pytest tests covering:
- Empty array edge case
- Valid range only: [1,2,9] etc
- Out of range: negatives, zero, 10+, floats if applicable
- Duplicates: [1,1,2,2]
- Mixed valid/invalid: [1, -5, 100, 5]
- Exception path: values causing KeyError in dict lookup
- Large arrays, all invalid arrays

Focus on try-except block coverage and edge cases.
```

Problem 4 prompt

```
Improve branch coverage for this palindrome construction function.

Current baseline: 87.4% line, 83.1% branch

Function: Find shortest palindrome beginning with input string.
Algorithm: Find longest palindrome suffix, append reversed prefix.
Implementation: Uses while loop with is_palindrome helper.

Generate 5-7 pytest tests covering:
- Empty string edge case
- Already palindrome: 'a', 'racecar'
```

```
- Needs construction: 'cat' → 'catac', 'ab' → 'aba'
- Different lengths: 1, 2, 3, 5, 10+ chars
- While loop iterations: 0 (already palindrome), 1, multiple
- Edge: string where suffix finding requires full traversal

Focus on while loop branch coverage and is_palindrome calls.
```

Before/after coverage numbers

Problem 10: by_length

Baseline Coverage (78.1% branch coverage)

```python
def by_length(arr):
    dic = {1: "One", 2: "Two", ..., 9: "Nine"}
    sorted_arr = sorted(arr, reverse=True)
    new_arr = []
    for var in sorted_arr:
        try:
            new_arr.append(dic[var])  # Exception path was not fully covered
        except:
            pass  # this branch not utilized
    return new_arr
```

Uncovered branches were:

- Exception path with out-of-range values (0, 10+, negatives)
- All-invalid array inputs
- Large arrays with exception handling

Problem 10 had the largest improvement gap because of all these above reasons.

After LLM Tests (100% branch coverage)

7 Tests generated by Gemini 2.5 Flash:

1. `test_empty_array`: Empty input `[]`
2. `test_valid_range_and_sorting_verification`: Valid integers with sort validation
3. `test_valid_range_with_duplicates`: Duplicate values `[9, 9, 1, 1]`
4. `test_out_of_range_positives_only`: `[10, 500, 100]` -> triggers exception path
5. `test_out_of_range_negatives_and_zero`: `[0, -1, -50]` -> tests boundary
6. `test_mixed_valid_and_invalid_numbers`: `[2, 10, 1, 14, 9, 0, 4]` -> both paths
7. `test_maximum_range_edge_cases`: Boundary values 1 and 9

Coverage improvement:

- Line: 92.2% –> 100% (+7.8%)
- Branch: 78.1% –> 100% (+21.9%)

- And all tests passed.

Problem 4: make_palindrome

Baseline Coverage (83.1% branch coverage)

```
def make_palindrome(string: str) -> str:
    if not string:  # Not always tested
        return ''
    beginning_of_suffix = 0
    while not is_palindrome(string[beginning_of_suffix:]):  # 0 iterations
not covered
        beginning_of_suffix += 1
    return string + string[:beginning_of_suffix][::-1]
```

Uncovered branches were:

- Empty string early return
- While loop with 0 iterations (already palindrome)
- Maximum iterations (full string traversal)

After LLM Tests (100% branch coverage)

7 Tests Generated by Gemini 2.5 Flash:

1. `test_empty_string`: `''` -> tests early return
2. `test_single_character_zero_iterations`: `'a'` -> while loop = 0 iterations
3. `test_already_palindrome_zero_iterations`: `'racecar'` -> already palindrome
4. `test_one_iteration_required`: `'ab'` -> exactly 1 iteration
5. `test_multiple_iterations_no_suffix`: `'cat'`, `'abc'` -> multiple iterations
6. `test_long_string_with_internal_palindrome`: 10+ chars
7. `test_partial_palindromic_suffix`: Tests complex suffix finding

Coverage improvement:

- Line: 87.4% –> 100% (+12.6%)
- Branch: 83.1% –> 100% (+16.9%)
- And all tests passed.

Redundancy

For **problem 10**, there was no duplicate test logic. Each test targeted distinct scenarios and the 7 tests helped achieve 100% coverage.

For **problem 4**, there was minimal redundancy. There was a clear iteration count progression (0, 1, multiple) and different string lengths were tested. The 7 tests helped achieve 100% coverage.

Note on convergence:

Both the problems converged at 100% in iteration 1 itself. This was because HumanEval tests already covered major code paths, the prompts to the LLM were effective in terms of exception paths, iterations, etc. and Gemini 2.5 Flash understood all edge cases well.

Part 3: Fault Detection Check

For each problem, I created 4 bugs representing common programming errors.

1. Off-by-one errors: Boundary condition mistakes
2. Wrong logic: Incorrect algorithm implementation
3. Missing error handling: Exception handling omitted
4. Boundary violations: Wrong range/bounds checks

Baseline tests: - HumanEval tests only (5 assertions each) which validate core functionality

Improved Tests: - Baseline + 7 LLM-generated tests (12 assertions each) which target branch coverage and edge cases

---

Problem 10: by_length

Bug 1: Off-by-one Error (Range 1-8 instead of 1-9)

Bug Code:

```
for var in sorted_arr:
    try:
        if 1 <= var <= 8:  # BUG: Should be <= 9
            new_arr.append(dic[var])
    except:
        pass
```

Detection:

- Baseline: CAUGHT (HumanEval test includes 9)
- Improved: CAUGHT by 3 tests
    - `test_maximum_range_edge_cases`
    - `test_mixed_valid_and_invalid_numbers`
    - `test_valid_range_with_duplicates`

Both caught it, but improved tests provided more specific failure information.

Bug 2: Wrong sort order (Missing reverse=True)

Bug Code:

```
sorted_arr = sorted(arr)  # BUG: Missing reverse=True
```

Detection:

- Baseline: CAUGHT (expected descending order)
- Improved: CAUGHT by 4 tests including `test_valid_range_and_sorting_verification`

Improved tests explicitly validate sort order with multiple test cases.

Bug 3: Missing exception handling

Bug Code:

```
for var in sorted_arr:
    new_arr.append(dic[var])  # BUG: No try-except
```

Detection:

- Baseline: CAUGHT (KeyError exception)
- Improved: CAUGHT by 3 tests
  - `test_out_of_range_positives_only`
  - `test_out_of_range_negatives_and_zero`
  - `test_mixed_valid_and_invalid_numbers`

Improved tests explored all exception paths with out-of-range tests.

Bug 4: Wrong boundary

Bug Code:

```
dic = {
    0: "Zero",  # BUG: Should start at 1
    1: "One",
    # ... rest of dictionary
}
```

Detection:

- Baseline: **MISSED** (no test includes 0)
- Improved: CAUGHT by 2 tests
  - `test_out_of_range_negatives_and_zero`: Expects [] for [0, -1, -50], got ['Zero']
  - `test_mixed_valid_and_invalid_numbers`: Input includes 0

Baseline tests never include 0 in inputs, so this boundary bug is undetected. LLM-generated tests explicitly test the 0 boundary case. In other words, boundary errors cause incorrect behavior with edge case inputs (0, 10, negatives) that were not covered by basic tests.

Problem 4: make_palindrome - Fault detection

Bug 1: Off-by-one in while loop

Bug Code:

```
beginning_of_suffix = 1  # BUG: Should start at 0
```

Detection:

- Baseline: CAUGHT
- Improved: CAUGHT by 2 tests
    - `test_already_palindrome_zero_iterations`
    - `test_single_character_zero_iterations`

Bug 2: Wrong slice indexing

Bug Code:

```
while not is_palindrome(string[beginning_of_suffix+1:]):  # BUG: +1 offset
```

Detection:

- Baseline: CAUGHT
- Improved: CAUGHT by 5 tests covering different iteration counts

---

Bug 3: Missing string reversal

Bug Code:

```
return string + string[:beginning_of_suffix]  # BUG: Missing [::-1]
```

Detection:

- Baseline: CAUGHT
- Improved: CAUGHT by 3 tests with varying string lengths

Bug 4: Broken helper function

Bug Code:

```python
def is_palindrome(string: str) -> bool:
    return string == string  # BUG: Should be string[::-1]
```

Detection:

- Baseline: CAUGHT
- Improved: CAUGHT by 4 tests

Note: The advantage of improved tests is mainly in boundary condition testing; all other bug types were caught by both baseline and improved tests.

Linking coverage <–> fault detection

Problem 10

Branch Coverage: - Baseline: 78.1%; Improved: 100% (+21.9%)

Fault Detection: - Baseline: 75% (3/4 bugs); Improved: 100% (4/4 bugs)

Correlation: The 21.9% coverage gap included exception handling paths, boundary checks (0, 10) and the uncovered "0 boundary" branch contained bug 4.

So branch 1 uncovered the else path that exposed bug 4. Achieving 100% coverage required tests that also caught the boundary bug.

Problem 4

Branch Coverage: - Baseline: 83.1%; Improved: 100% (+16.9%)

Fault Detection: - Baseline: 100% (4/4 bugs); Improved: 100% (4/4 bugs)

Correlation: Despite coverage gap, both achieved 100% fault detection.

Problem 4's baseline tests were already pretty comprehensive. It has fewer difficult edge cases than problem 10's exception handling.

Conclusion: coverage and fault detection have a strong positive correlation. 100% branch coverage required tests that caught bugs. Branch-focused generation naturally tests edge cases.

GitHub Repository:
https://github.com/lava-nika/CS520-Exercise2-Automated-Testing-and-Coverage

---