How does LLM-based test generation for Java libraries perform when full source code is available?

Evaluating LLM-based test generation for libraries across code representations

EEMCS, Delft University of Technology, The Netherlands

Author
Cao Minh Nguyen
c.m.nguyen-1@student.tudelft.nl

Supervisors
Cathrine Paulsen
Sebastian Proksch

1 Background & Motivation

Problem

- **Dependency Risks:** Software relies heavily on open-source libraries requiring frequent updates.
- **Breaking Changes:** Updates can introduce bugs or breaking behavioural changes.
- **Test Scarcity:** Developers need tests to ensure updates remain compatible, but library tests are often incomplete & tools like Dependabot suggest updates without verifying compatibility.

Knowledge Gap

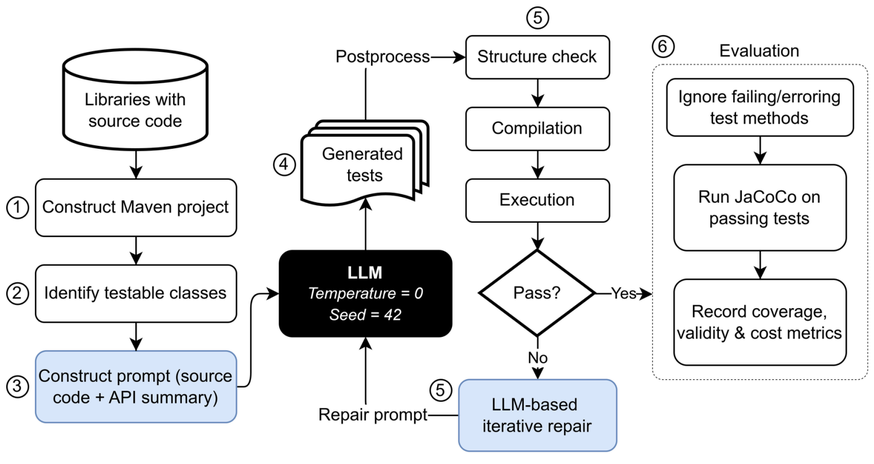
- **Traditional Tools:** Automated test generation tools (e.g., EvoSuite) generate tests that are hard to read and integrate into real workflows.
- **Benchmark Focus:** Existing LLM-based test generation studies evaluate on benchmarks datasets or GitHub projects rather than real-world released libraries.
- **Gap:** It remains unclear whether LLMs can generate effective tests in real-world dependency validation settings.

Opportunity

- **LLMs Potential:** LLMs can generate effective and readable tests from source code, improving usability for developers.
- **Study Purpose:** Evaluates LLM-based test generation for released Java libraries on Maven Central to assess its usefulness for realistic dependency validation workflows.

2 Methodology

Figure 1. Overview of Python pipeline for LLM-based test generation and evaluation



- Generates, validates, and evaluates LLM-generated JUnit tests for **34 Java libraries** obtained from Maven Central.
- Generates tests using a locally hosted Qwen2.5-Coder-7B model (6k context window).
- Implements a **generation → validation → repair loop** using Maven compilation and execution feedback (up to 2 repair iterations).
- Effectiveness is assessed using code coverage (JaCoCo), compilation and runtime validity, compilation error categories, and computational cost.
- Compares results against **EvoSuite baseline** in terms of coverage.

3 Results

RQ1: How does the coverage achieved by LLM-generated tests compare with EvoSuite-generated tests?

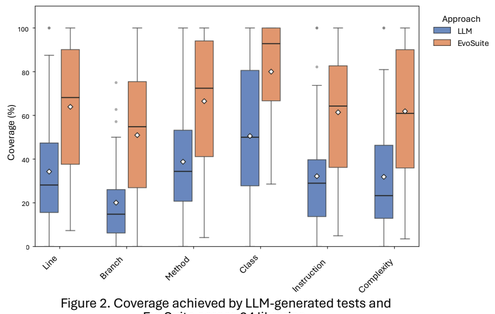


Figure 2. Coverage achieved by LLM-generated tests and EvoSuite across 34 libraries.

- Median line coverage: 68.2% (EvoSuite) vs. 28.1% (LLM).
- Median branch coverage: 54.8% (EvoSuite) vs. 14.8% (LLM).
- Low branch coverage: LLM-generated tests explored relatively few execution paths.

⭐ **Key Finding:** Tests generated by the local Qwen2.5-Coder-7B model achieved substantially lower coverage than EvoSuite, motivating further analysis of test validity and compilation failures (RQ2).

RQ2: What proportion of LLM-generated tests compile and execute successfully, and what are the main causes of invalid tests?

- Only **56.3%** of generated test classes compiled (197/350 CUTs).
- Among compiled tests, 78.2% of test methods passed.
- Most frequent compilation failure: *cannot find symbol* (33.6% of failed test classes).

⭐ **Key Finding:** Compilation failures, particularly unresolved symbols, were the primary reason for the low coverage achieved by the local LLM.

RQ3: How does iterative repair affect the validity, coverage, and computational cost of LLM-generated tests?

Metric	0	1	2	3
Test validity				
Compilation success rate (%)	43.93	52.80	58.41	57.94
Runtime success rate (%)	73.32	76.60	77.57	78.58
Coverage				
Median line coverage (%)	22.30	30.32	34.57	33.40
Median branch coverage (%)	12.84	19.63	22.53	19.50
Computation cost per CUT				
Tokens/class	2488.6	5145.3	6904.2	8900.5
Pipeline runtime/class (s)	23.05	41.87	55.63	68.98

- Iterative repair improved both test validity and coverage up to 2 attempts.
- Two repair attempts improved both line and branch coverage, but increased token usage (2.8×) and runtime (2.4×).
- A third repair attempt provided diminishing returns in coverage.

⭐ **Key Finding:** Two repair attempts provided the best trade-off between coverage, validity, and computational cost. Repaired output can discard previously valid tests.

RQ4: How does pipeline performance differ between a local open-source LLM and a more capable cloud-hosted LLM?

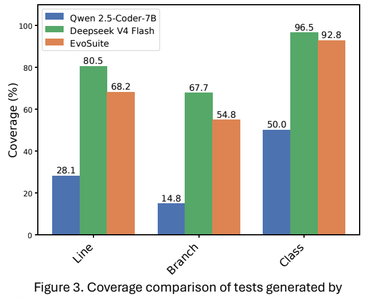


Figure 3. Coverage comparison of tests generated by DeepSeek (cloud LLM), Qwen2.5 (local LLM), and EvoSuite

- **DeepSeek V4 (Cloud) vs. Qwen2.5 (Local):** Cloud model significantly outperformed the local model across all coverage metrics.
- **Higher Validity:** Deepseek achieved **89.4%** compilation success vs. Qwen2.5's **56.3%**, yielding higher passing rates and more executed tests.

⭐ **Key Finding:** Stronger LLMs can surpass EvoSuite in coverage, but at higher computational cost than smaller local models.

4 Discussion

- **API grounding is a key limitation:** Small LLMs frequently hallucinate invalid symbols even with API context → future research on improved API-aware generation strategies.
- **Compilation failures dominate coverage loss:** Most reduced coverage is due to non-compiling tests, motivating method-level test generation to better handle compilation failures.
- **Iterative repair is essential:** Repair should be included in production systems to improve test validity, although more repair iterations is not always beneficial.
- **Model capability is critical:** Strong models can surpass EvoSuite in coverage → potential for real-world use in dependency updates validation.
- **Clear cost–performance trade-off:** More capable models improve test quality but require significantly higher token cost and generation time.

5 Conclusion

- Evaluate LLM-based test generation for released Java libraries rather than benchmarks, targeting dependency updates validation.
- LLM-based test generation is strongly limited by **API grounding issues**, causing compilation failures.
- **Iterative repair improves results but with diminishing returns**, making it a supporting rather than primary solution.
- **Model capability** is the key determinant of test quality.

Future Direction

- Method-level generation & hybrid LLM + EvoSuite approaches are promising directions for improving test validity and coverage.

👑 **Key Takeaway**

- **LLM-based test generation from source code is promising for real-world dependency update validation when combined with sufficiently capable models and iterative repair mechanisms, demonstrating strong potential to complement existing SBST approaches and improve automated testing quality in practice.**