Context-First Prompting and Feedback-Driven Repair for Unit Test Generation with GPT-4o-mini

Arjun Raj ▪ Manindra de Mel ▪ Nathan Woodburn

School of Computing, Australian National University

COMP4130 ▪ Assignment 4 ▪ 2026

Problem statement

LLMs treat unit-test generation as creative writing, not coverage.

- Default GPT-style prompts return plausible JUnit code that *compiles* but exercises only the focal method's happy path.
- Branch conditions where Defects4J faults live remain untested.
- No feedback loop $\Rightarrow$ failing output discarded, not learned from.

Goal. Generate branch-targeted, fault-revealing JUnit 4 tests on three Defects4J subjects using GPT-4o-mini (mandated): Commons Codec 18, Collections 27, Compress 45; focal classes `StringUtils`, `MultiValueMap`, `TarUtils`.

Targets: high line/branch coverage, high pass rate, and *identify* the seeded bug in every focal class.

Why default LLM prompting misses the bug

Example: `StringUtils.equals` (Codec 18). Fault path requires *three* conditions at once:

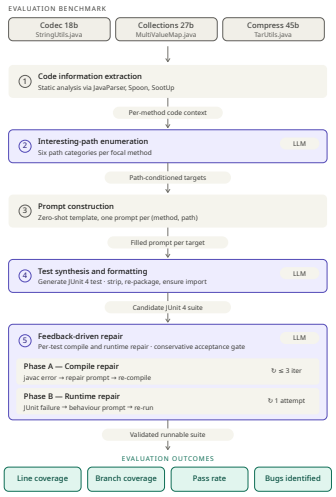
1. at least one argument is a non-`String` `CharSequence` (e.g. `StringBuilder`);
2. the two arguments have different lengths;
3. delegates to `CharSequenceUtils.regionMatches` with a `Math.max` length (should be `Math.min`).

An unguided prompt has no reason to pick those exact inputs.

- Three families of missing knowledge: *syntactic*, *control/data-flow*, *dependency*.
- Same gap on `MultiValueMap.deserialize` and `TarUtils.formatLongOctalOrBinaryBytes`.

Leverage: structured context + enumerated targets, not example tests.

Five-stage, context-first pipeline



1. **Extractor:** 28 fields per focal method (source, fields, constructors, predicates, throws, Jimple IR).
2. **Path inventory:** one (focal, target path) per LLM call; edge-class taxonomy.
3. **Zero-shot prompt:** `@persona` / `@terminology` / `@instruction` directive template.
4. **Generation driver:** GPT-4o-mini, formatter, per-path `.java` file.
5. **Feedback loop:** compile-repair (≤ 3 iter.) plus behaviour-guided refinement.

Stage 1–2: extraction and target enumeration

Code-information extractor (Task 1).

- **Syntactic:** signature, source, javadoc, fields, constructors, imports.
- **Control/data-flow:** branch hints, exact predicates, loops, throws, side effects, Jimple IR.
- **Dependency:** helper calls, sibling methods on the focal class.
- Tools: JavaParser + Spoon + SootUp; one CSV row per non-constructor method.

Fuzzing-inspired path inventory (Task 2).

- Each path: stable ID, taxonomy label, predicate, input shape, expected outcome.
- Edge inclusion: **229 boundary, 104 null, 108 empty/minimal, 72 exceptional**.
- **513 / 684 generation units = 75%** target an explicit edge condition.

Stage 3–5: prompt, generate, feedback

Zero-shot directive prompt (Task 3).

- `@persona` (senior Java engineer, JUnit 4) + `@terminology` (every placeholder) + `@instruction` (12 rules: JUnit 4, JDK 8, try/catch, must start with `package`, ...).
- *No exemplars*; all leverage carried by structured context placeholders.

Generation + two-phase repair (Tasks 4–5).

- Hash-named test file per (focal FQN, Path ID); overload-safe, collision-free.
- **Phase A** – compile repair: feed verbatim `javac` diagnostics back, ≤ 3 iterations.
- **Phase B** – behaviour repair: runtime failure + coverage gap drive next prompt; accept only if compiles *and* passes.

Experimental setup

Subjects (Defects4J buggy versions).

- Commons Codec 18, focal `StringUtils`
- Commons Collections 27, focal `MultiValueMap`
- Commons Compress 45, focal `TarUtils`

Pipeline configuration.

- Model: **GPT-4o-mini** (held fixed)
- Test framework: JUnit 4, JDK 8
- Coverage: JaCoCo 0.8.12 (focal-class scoped)

Output of one generation sweep.

- **684** parseable JUnit 4 test files
- **669** runnable after compile-repair (97.8%)
- 15 unrecoverable on `MultiValueMap.deserialize`
- **75%** of units target edge conditions

Results: pass rate and coverage

Pass rate (per focal class).

Class	Run.	Pass	Rate
`StringUtils`	207	152	73.4%
`MultiValueMap`	220	154	70.0%
`TarUtils`	242	117	48.3%
Total	**669**	**423**	**63.2%**

Coverage (JaCoCo, cov/total, %).

Class	Line	Branch
`StringUtils`	38/39 (97.4)	19/20 (95.0)
`MultiValueMap`	81/88 (92.0)	38/44 (86.4)
`TarUtils`	146/155 (94.2)	99/102 (97.1)
Aggregate	**265/282 (94.0)**	**156/166 (94.0)**

Weighted aggregate of **94.0% line / 94.0% branch** across the three focal classes; pass rate is bounded by model accuracy on edge inputs, lowest on `TarUtils` byte-layout methods.

Bug identification + conclusion

All three seeded Defects4J faults exposed (3/3).

Defect	Evidence (failing test, signal)
`StringUtils.equals` (Codec 18)	`...PATH_017_Test`: `AssertionError` on non-String `CharSequence`, differing lengths.
`MultiValueMap.deserialize` (Coll. 27)	`...readObject_PATH_006_Test`: `ClassCastException` `ArrayList`→`HashSet`.
`TarUtils.formatLongOctalOrBinaryBytes` (Compress 45)	`...PATH_006_Test`: `ArrayIndexOutOfBoundsException`: 9 on small buffer.

Worked. Structured context + enumerated targets + repair loop ⇒ 94.0% line / 94.0% branch and 3/3 bug identification on small hosted model.

Didn't. 15 `MultiValueMap.deserialize` compile failures escaped the three-iteration budget (generics, visibility).

Next. Data-flow paths, larger eval, model-substitution robustness (open-source 7B study in appendix).