

Lecture 11 – LLM-based Software Testing

AAA705: Software Testing and Quality Assurance

Jihyeok Park



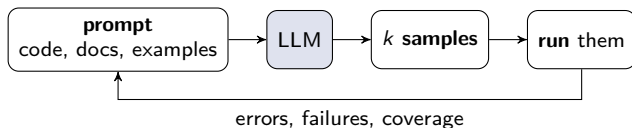
2026 Fall

```
1  int mid(int x, int y, int z) {  
2      int m = z;  
3      if (y < z) {  
4          if (x < y)      m = y;  
5          else if (x < z) m = y; // bug  
6      } else {  
7          if (x > y)      m = y;  
8          else if (x > z) m = x;  
9      }  
10     return m;  
11 }
```

Last week: a model wrote $m = x$.
The **tests** judged it.

Today: the model writes the **tests**.

Who judges those?



- Trained on all public code: it writes what code like this **usually** looks like
- It can also fill a blank: <MASK> (last week's AlphaRepair)
- It does **not** run anything. Every tool today adds the run

1. Unit Test Generation
2. LLM-based Fuzzing
3. The Oracle Problem, Again
4. Agents, and Evaluating Them

1. Unit Test Generation
2. LLM-based Fuzzing
3. The Oracle Problem, Again
4. Agents, and Evaluating Them

A model alone writes plausible tests → give it the function's documentation.

TestPilot¹: no training, no example tests. Four things about the function:

```
let delta = require('quill-delta');
// signature:  concat(other)
// doc:        Returns a new Delta of this and other combined
// usage:      delta.concat(other)
// body:       concat(other) { const delta = new Delta(...)
describe('test quill-delta', function() {
  it('test concat', function(done) {
```

- All four help. The **body** and the **doc comment** help most; usage examples are not essential

¹[TSE'23] M. Schäfer, S. Nadi, A. Eghbali, F. Tip. "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation."

The model's test asserts a wrong number. Run it, append the error, ask again:

```
let delta3 = delta1.concat(delta2);
assert.equal(delta3.ops.length, 6);    // fails
// the test above fails with the following error:
//   expected 5 to equal 6
// fixed test:
let delta3 = delta1.concat(delta2);
assert.equal(delta3.ops.length, 5);    // passes
```

- One retry per test. It rescues about one failing test in six
- On 25 npm packages: higher coverage than the search-based baseline (Nessie)
- The model **copied the observed value**. Is 5 right? Nobody asked

TestPilot counted coverage → what about the tests that never ran?

Yuan et al.²: 1,000 Java tests written by ChatGPT.

does not compile	58%
compiles, fails	17%
passes	25%

- The passing ones match **manual** tests in coverage
- **ChatTester**: feed the compile error back, up to three rounds → 73% compile

²[FSE'24] Z. Yuan, M. Liu, S. Ding, K. Wang, Y. Chen, X. Peng, Y. Lou. "Evaluating and Improving ChatGPT for Unit Test Generation."

```
StringBuilder("Hello World").setCharAt(6, c):
```

```
// prompt: class + method body
sb.setCharAt(6, '-');
assertEquals("Hello-World",
            sb.toString());
// actual: "Hello -orld"
```

wrong assertion

```
// 1. "Infer the intention"
// -> "set the character at the
//     specified index"
// 2. generate with the intention
sb.setCharAt(6, 'J');
assertEquals("Hello Jorld",
            sb.toString());
```

passes

- ChatTester repairs **compile** errors only. A **failing** assertion is left alone
- `mid(2, 1, 3)`: expected 2, got 1. Is the test wrong, or the program?
The tool cannot know

Both studies used small methods → try a real project.

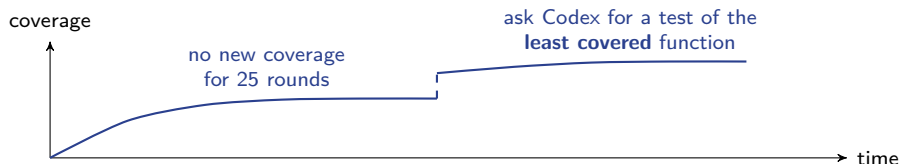
Siddiq et al.³: the same models on HumanEval (toy functions) and SF110 (real projects).

line coverage	toy functions	real projects
Codex	87%	2%
GPT-3.5	69%	1%
EvoSuite (search)	96%	28%

- In a real project the test needs **dependencies, fixtures, a build** – none of it is in the prompt. Most tests do not compile
- Unknown symbols are the top error: the model **invents** types

³[EASE'24] M. L. Siddiq, J. C. S. Santos, R. H. Tanvir, N. Ulfat, F. Al Rifat, V. C. Lopes. "Using Large Language Models to Generate JUnit Tests: An Empirical Study."

Search-based tools run as they go, but get stuck → ask the model for a seed.



- **CodaMosa**⁴ = Pygmin's search + this one extra move
- The model's test is turned into a **seed**; mutation takes over again

⁴[ICSE'23] C. Lemieux, J. P. Inala, S. K. Lahiri, S. Sen. "CodaMosa: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models."

`bump_version(version, pos=2)` expects a string like `"1.2.3"`.

```
# search alone: random strings, instant exceptions
bump_version('a\t!sUo~AU');    bump_version(None)

# Codex, asked for a test of bump_version
def test_bump_version():
    assert bump_version('0.0.0') == '1.0.0'    # wrong value, right shape

# the search mutates the seed and reaches new branches
bump_version('0.0.0', -1);    bump_version('0.0.0', -2687)
```

- The model knows what a version string **looks like**. The search does the rest
- 486 Python modules: higher coverage on 173, lower on 10

Research counts coverage → at Meta, every accepted test must be an improvement.

TestGen-LLM⁵ extends **existing** test classes. The whole prompt:

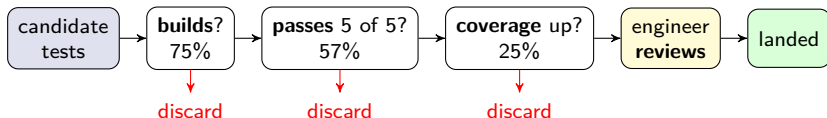
Here is a Kotlin unit test class and the class that it tests:

```
{existing_test_class}  
{class_under_test}
```

Write an extended version of the test class that includes additional unit tests that will increase the test coverage of the class under test.

- Nothing clever in the prompt. The work is in what happens **after**

⁵[FSE'24] N. Alshahwan, J. Chheda, A. Finogenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, E. Wang. "Automated Unit Test Improvement Using Large Language Models at Meta."



- Of the test classes tried, a quarter got a test that survived all three filters. Reviewers accepted **73%** of those
- What the filters miss, the reviewer catches: a test of a getter, a test with **no assertion**
- The filters are this course's oracles. The model only supplies **candidates**

1. Unit Test Generation
2. LLM-based Fuzzing
3. The Oracle Problem, Again
4. Agents, and Evaluating Them

PyTorch inputs must respect tensor shapes → the model learned that grammar.

TitanFuzz⁶ asks Codex, one API at a time:

```
"""
Task 1: Import TensorFlow 2.10.0
Task 2: Generate input data
Task 3: Call the API tf.nn.conv2d(input, filters, strides, padding)
"""

import tensorflow as tf
# Codex continues:
input  = tf.Variable(tf.random.normal([1, 10, 10, 1]))
filter = tf.Variable(tf.random.normal([3, 3, 1, 1]))
op      = tf.nn.conv2d(input, filter, strides=[1, 1, 1, 1],
                        padding='VALID')
```

The shapes match. No grammar, no dictionary, no seed corpus.

⁶[ISSTA'23] Y. Deng, C. S. Xia, H. Peng, C. Yang, L. Zhang. "Large Language Models Are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models."

Mask a piece of a seed program. An infilling model (InCoder) fills it:

```
# seed
A = torch.rand(50, 50); B = torch.clone(A); C = torch.mm(A, B)

C = torch.mm(<SPAN>)           # replace the arguments
C = torch.<SPAN>(A, B)          # replace the method
<SPAN>; C = torch.mm(A, B)     # replace the prefix
```

- Keep programs that are **valid** and **different**: more unique API calls, deeper dataflow, fewer repeats
- The fuzzing loop is the usual one. Only the mutator changed

Run every program on CPU and on GPU. Compare:

```
x = torch.randn(10, 10).log()           # some entries are NaN
y = torch.histc(x, bins=10, min=0, max=1)
# CPU: [48, 0, 0, ...]    -- counts every NaN in bin 0
# GPU: [ 2, 0, 0, ...]    -- does not
```

- Oracle: CPU vs. GPU disagree, or a crash
- 65 bugs, 41 previously unknown. The model made the inputs; the oracle is from the oracles lecture

Models write typical code; bugs hide in atypical code → show it old bugs.

FuzzGPT⁷ mines bug-triggering snippets from issue trackers and puts them in the prompt:

```
# API: torch.Tensor.apply_      Bug: apply_ fails on ...
x = torch.randn(3, 3);  x.apply_(lambda a: a + 1)
# API: torch.gather          Bug:          <- the model continues
```

A bug it found, confirmed as a security issue by Google:

```
x = np.arange(10);  x_copy = tf.experimental.numpy.copy(x)
x[3] = 42           # on CPU, x_copy[3] becomes 42 too: shared memory
```

⁷[ICSE'24] Y. Deng, C. S. Xia, C. Yang, S. D. Zhang, S. Yang, L. Zhang. "Large Language Models are Edge-Case Generators: Crafting Unusual Programs for Fuzzing Deep Learning Libraries."

TitanFuzz is built for one library → let the model write the prompt from the docs.

Fuzz4All⁸: give it the documentation of the target. GPT-4 distills it into a short prompt. Then loop with one of three instructions:

generate new	“create a program which uses complex SMT2 logic”
mutate	“create a mutated program that modifies the previous generation”
keep semantics	“create a semantically equivalent program to the previous generation”

Targets: GCC, Clang, Z3, CVC5, Go, javac, Qiskit – 6 languages, same fuzzer.

⁸[ICSE'24] C. S. Xia, M. Paltenghi, J. L. Tian, M. Pradel, L. Zhang. “Fuzz4All: Universal Fuzzing with Large Language Models.”

```
(declare-const x1 Real) (assert (! (= x1 1))) (check-sat)
                        | mutate
(declare-const x1 Int)  (assert (! (= x1 1))) (check-sat) (get-model)
                        | keep semantics
(declare-const x1 Int)  (assert (! (= x1 1) :named a)) (check-sat)
                        -> CVC5 crashes (fixed)
```

- 98 bugs across the seven systems, 64 previously unknown
- It makes **43% fewer** inputs than a classic fuzzer in 24 hours, and covers **37% more** code: each input is worth more

A protocol's grammar exists only as RFC prose → the model has read the RFC.

ChatAFL⁹ asks once, before fuzzing starts:

For the RTSP protocol, all of client request grammar is:

```
(example) PLAY: {PLAY <Value> RTSP/1.0\r\n, CSeq: <Value>\r\n,  
                Session: <Value>\r\n, Range: <Value>\r\n, \r\n}  
-> DESCRIBE: {...}  SETUP: {...}  PAUSE: {...}  TEARDOWN: {...}
```

- AFLNet then mutates only the <Value> parts: the message stays well-formed
- Asked five times, majority vote – the model is not always right

⁹[NDSS'24] R. Meng, M. Mirchev, M. Böhme, A. Roychoudhury. "Large Language Model Guided Protocol Fuzzing."

A Live555 use-after-free needs SETUP → PLAY → PAUSE → PLAY. No recorded seed contained PAUSE.

Here is one sequence of client requests: DESCRIBE... SETUP... PLAY... Please add the PAUSE and TEARDOWN requests in the accurate locations.

- The model is called **twice**: to enrich the seeds at the start, and for “what message comes next?” when coverage stalls
- Everything else runs at AFLNet speed. 9 new vulnerabilities in Live555, ProFTPD, Kamailio

1. Unit Test Generation
2. LLM-based Fuzzing
3. The Oracle Problem, Again
4. Agents, and Evaluating Them

Inputs were the easy half → the assertion needs the intended behavior.

TOGA¹⁰: given a test prefix and the method's signature and docstring – **not** its body – pick the oracle.

```
Stack s = new Stack(); s.push(2); s.pop();  
boolean empty = s.isEmpty();
```

- 1 Will this prefix **throw**? A classifier says no
- 2 Rank the candidates: `assertTrue(empty)`, `assertFalse(empty)`, `assertNull(...)`, `assertEquals(0 | 1 | -1 | a local value, ...)`
- 3 Highest score: `assertTrue(empty)`. If `pop` is a no-op, this **catches** it

¹⁰[\[ICSE'22\]](#) E. Dinella, G. Ryan, T. Mytkowicz, S. K. Lahiri. "TOGA: A Neural Method for Test Oracle Generation."

```
int m = mid(2, 1, 3);
```

candidates	from
<code>assertEquals(0 1 -1, m)</code>	common constants
<code>assertEquals(2 1 3, m)</code>	values in the prefix

- The right one is in the list. Will the ranker pick 2 over 1 and 3? It has never seen the body
- Reported: 57 Defects4J bugs found, 30 by no other tool

57 bugs → two groups re-ran the experiment.

TOGA's prefixes were generated by EvoSuite on the **fixed** program, then kept if they fail on the buggy one.

prefixes generated on	bugs found
the fixed version (TOGA's setup)	104
the buggy version (the real situation)	64

- Liu et al.¹¹: a prefix made on the fixed version already **knows** where the bug is
- Hossain et al.¹²: of the 57 bugs, **38** were caught because the prefix threw an exception – no oracle needed

¹¹[ISSTA'23] Z. Liu, K. Liu, X. Xia, X. Yang. "Towards More Realistic Evaluation for Neural Test Oracle Generation."

¹²[FSE'23] S. B. Hossain, A. Filieri, M. B. Dwyer, S. Elbaum, W. Visser. "Neural-Based Test Oracle Generation: A Large-Scale Evaluation and Lessons Learned."

oracle	bugs found	precision
TOGA	64	0.38%
“expect no exception”	44	0.77%

- 0.38%: a developer reads **260** failing tests to find one real bug
- The trivial oracle finds most of the same bugs with twice the precision
- `mid(2, 1, 3)` never throws. Only an **assertion** can catch it – and that is where TOGA is weakest
- Same lesson as “55 of 105”: measure an oracle by its **false alarms**

Where does the expected value come from? → sometimes a human wrote it down.

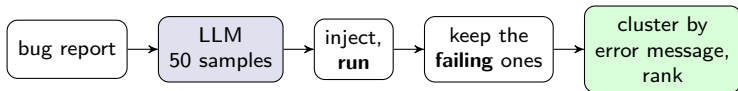
A real report (Commons Math): *“In MathUtils, some equals methods return true if both arguments are NaN. This contradicts the IEEE standard.”*

LIBRO¹³ puts the report in the prompt and asks for a test:

```
public void testEquals() {  
    assertFalse(MathUtils.equals(Double.NaN, Double.NaN));  
    assertFalse(MathUtils.equals(Float.NaN, Float.NaN));  
}
```

The expected value came from a sentence. Earlier tools could reproduce only **crashes**.

¹³[ICSE'23] S. Kang, J. Yoon, S. Yoo. “Large Language Models Are Few-Shot Testers: Exploring LLM-based General Bug Reproduction.”



- A test that passes reproduced nothing. Show nothing unless several samples **fail the same way**
- Rank first the tests whose failure mentions values from the report
- Defects4J: a reproducing test for 251 of 750 bugs. On 31 reports written **after** the model's training cutoff: the same rate
- “mid(2, 1, 3) returns 1, expected 2” → `assertEquals(2, mid(2, 1, 3))`

No bug report, no human → let the model write the reference.

Ask ChatGPT whether these two differ:

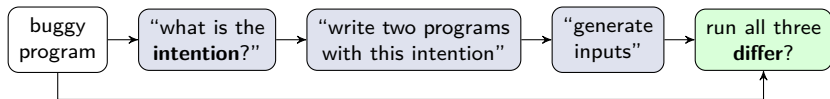
```
def gcd(a, b):  
    if b == 0: return a  
    return gcd(a, a % b)
```

```
def gcd(a, b):  
    if b == 0: return a  
    return gcd(b, a % b)
```

“No. Both implement the same GCD algorithm.” But gcd(12, 20) gives 12 and 4.

- Asked directly for a failing test of a buggy program, ChatGPT succeeds 29% of the time¹⁴

¹⁴[ASE'23] T.-O. Li, W. Zong, Y. Wang, H. Tian, Y. Wang, S.-C. Cheung, J. Kramer. “Nuances Are the Key: Unlocking ChatGPT to Find Failure-Inducing Tests with Differential Prompting.”



input	reference 1	reference 2	buggy
(12, 20)	4	4	12

→ `assert gcd(12, 20) == 4`

- Differential testing. The second implementation is written by the model – from the **intention**, not from the buggy tokens
- One of its benchmark problems is “median of three”: it finds `mid(2, 1, 3)` every time

1. Unit Test Generation
2. LLM-based Fuzzing
3. The Oracle Problem, Again
4. Agents, and Evaluating Them

Last week RepairAgent chose its tools → what should the tools look like?

SWE-agent¹⁵: not a shell. A few commands with compact output:

```
> open solvers/diophantine.py 400
[File: solvers/diophantine.py (3093 lines)]   (400 lines above)
401: elif total_degree == 3:
402:     diop_type = "cubic_thue"
...                                           (2684 lines below)
> edit 404:407
elif (total_degree > 3 and len(set(k.exp[k.is_Pow])) == 1):
end_of_edit
[lintor] E999 SyntaxError -> edit rejected, file unchanged
```

Search returns at most 50 hits; the viewer shows 100 lines; a bad edit is undone.

¹⁵[\[NeurIPS'24\]](#) J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, O. Press. “SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering.”

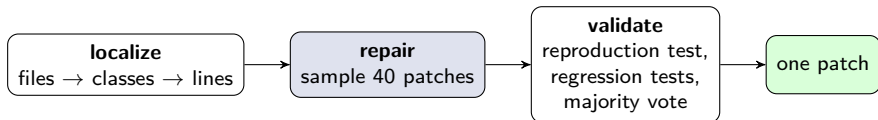
Same model (GPT-4 Turbo), SWE-bench Lite:

interface	resolved
plain Linux shell	11.0%
SWE-agent	18.0%
without the linter	15.0%
viewer shows the whole file	12.7%
viewer shows 30 lines	14.3%

- Too little context and too much context both hurt
- The model did not change. The **tools** did

Agents take 30–40 turns and lose track → a fixed three-step pipeline.

Agentless¹⁶: the model never decides what to do next.



- The model also writes a test that **reproduces** the issue; patches that make it pass survive
- Last week's loop, localize → generate → validate, with an LLM in each box

¹⁶[\[arXiv'24\]](#) C. S. Xia, Y. Deng, S. Dunn, L. Zhang. "Agentless: Demystifying LLM-based Software Engineering Agents."

how the final patch is chosen	resolved
majority vote only	25.7%
+ must pass the regression tests	27.0%
+ must pass the generated reproduction test	32.0%

- Best open-source result at the time, at \$0.70 per issue
- The gain came from **judging** candidates better, not from generating them better
- The judge is a test the model wrote. Who judges that?

Every number today came from a model that changes → can anyone repeat it?

Sallou et al.¹⁷ asked ChatGPT for tests of the same class, ten times, on different days:

Defects4J Math-5	as is	reciprocal renamed
branch coverage	70%, spread 28 points	tests do not compile
failing tests	1–2 of 5–7	–

- Rename one method and the model **invents** members that do not exist. It had seen this class before

¹⁷[ICSE-NIER'24] J. Sallou, T. Durieux, A. Panichella. "Breaking the Silence: The Threats of Using LLMs in Software Engineering."

- The **model version** and the date of the queries
- The exact **prompts**
- **Repeated runs** and their spread, not one number
- Results on **transformed** or unseen code: renamed, from outside GitHub, written after the training cutoff

The fuzzer-evaluation rules, again. Last week's SWE-bench+ found the answers inside the issues; the same check applies here.

	the model replaced	it did not replace
inputs	the grammar, the seeds, the mutator	running the program
unit tests	writing the test	build, run, cover, review
oracles	drafting the assertion	checking the assertion
agents	choosing the next step	the tests that judge the patch

- The model made **generating** cheap. Every honest evaluation today found the **false alarms**
- The work moved: from writing tests to **judging** generated ones. That judging is this course

1. Unit Test Generation
2. LLM-based Fuzzing
3. The Oracle Problem, Again
4. Agents, and Evaluating Them

- Course Review

Jihyeok Park

`jihyeok_park@korea.ac.kr`

`https://plrg.korea.ac.kr`