a mini introduction to

Software Testing

Pengyu Nie <pynie@uwaterloo.ca>

Agenda

- Concepts & taxonomy
- Regression testing
- Differential & metamorphic testing
- Automated test generation

Test Case & Test Suite

code under test (CUT)

```
def add(a, b):
  return a + b
```

```
foo
```

```
bar
```

```
...
```

tests

```
def test_add():
  a = 1
  b = 2

  res = add(a, b)

  assert res == 3
```

test case
smallest check for one program behavior
aka: test method

arrange
preparing inputs

act
invoking CUT

assert / oracle
checking outputs

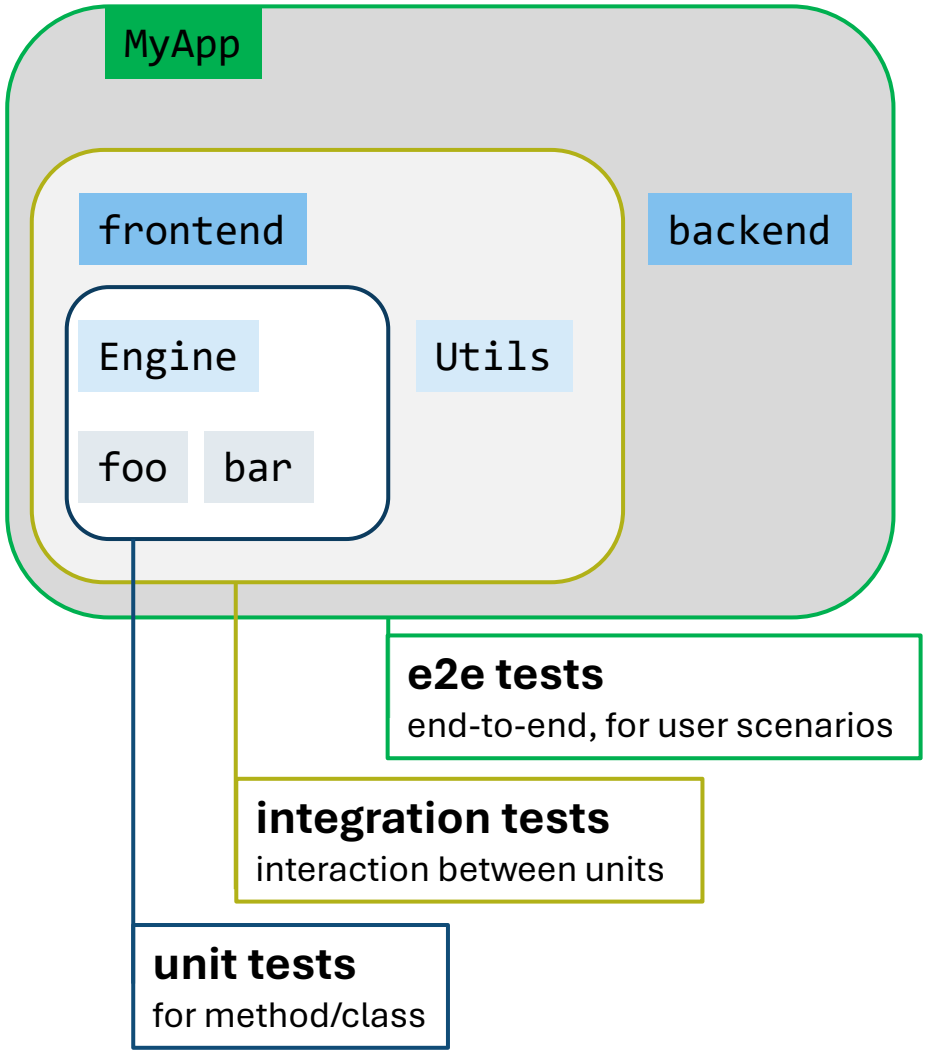
```
test_foo
```

```
test_bar_1
```

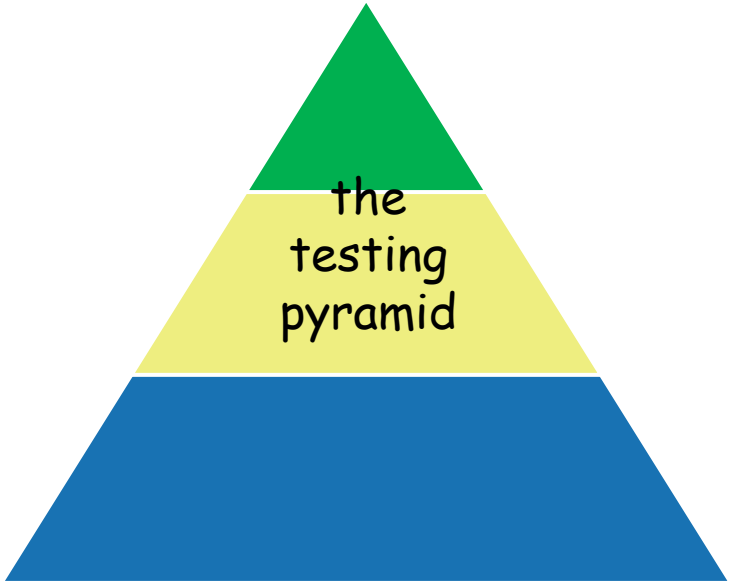
```
test_bar_2
```

test suite
a collection of test cases
aka: test class

Taxonomy of Tests



by testing granularity



Related work (of mine):
inline tests for individual code statements

Taxonomy of Tests (cont.)

by testing subjects

functional tests
for functional requirements / business logic

UI tests
for user interface

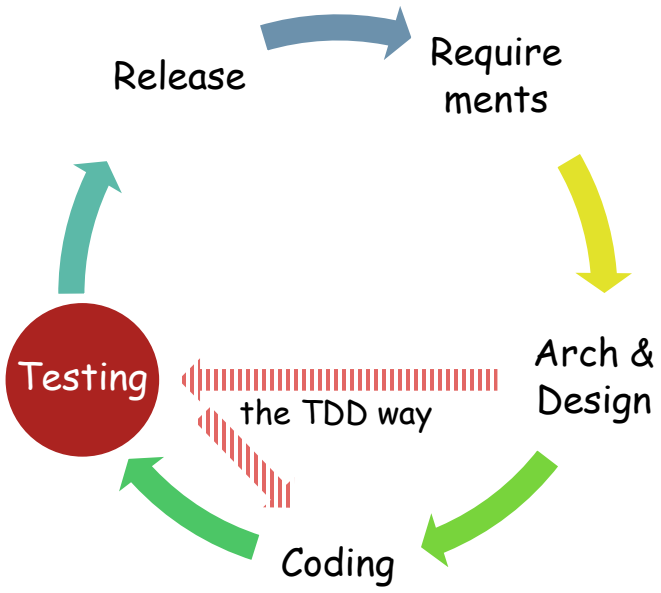
performance tests
measuring code efficiency

accessibility tests
check compliance with accessibility requirements

compatibility tests
check compatibility wrt OS/hardware versions

by testing frameworks

Why Testing



- Check compliance with software requirements
  - functional requirements
  - safety, security, privacy, etc.
- Catch defects and bugs earlier than later
  - continuous integration
  - test-driven development
- Serve as executable specifications of expected behavior
  - documentation
  - refactoring

Regression Testing

- Regression tests are executed (on CI) at every code commit
- Make sure future changes don't silently break existing functionalities

```
def apply_discount(x):
  if x >= 50:
    return 0.9 * x
  return x
```

```
def test_apply_discount_50(x):
  assert apply_discount(50) == 40
```

✓

```
def apply_discount(x):
  if 25 <= x <= 50:
    rate = 0.95
  elif x >= 50:
    rate = 0.9
  return rate * x
```

```
def test_apply_discount_50(x):
  assert apply_discount(50) == 40
```

✗

Hey, you broke the codebase!
Fix the bug before you can merge.

```
def apply_discount(x):
  if 25 <= x < 50:
    rate = 0.95
  elif x >= 50:
    rate = 0.9
  return rate * x
```

```
def test_apply_discount_50(x):
  assert apply_discount(50) == 40
```

✓

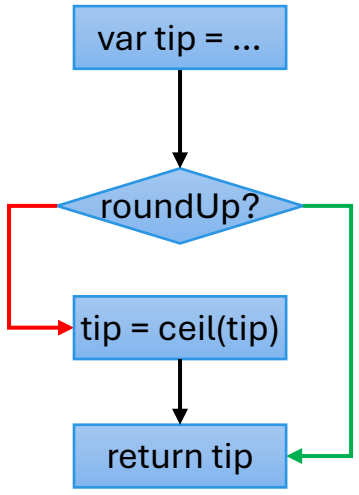
```
def test_apply_discount_25(x):
  assert apply_discount(25) == 23.75
```

✓

Don't forget to test new code

Code Coverage

- How many regression tests should we have? (test adequacy metric)
- What % of code elements is “covered” when executing the test suite?
  - line coverage 3 / 4 lines = 75%
  - branch coverage 1 / 2 branches = 50%




```
class TipCalculator {
    var amount: Double = 0.0
    var tipPercent: Double = 0.0
    var roundUp: Boolean = false

    fun calculateTip(): Double {
        var tip = tipPercent / 100 * amount
        if (roundUp) {
            tip = ceil(tip)
        }
        return tip
    }
}
```

```
@Test
fun testCalculateTip() {
    val calculator = TipCalculator()
    calculator.amount = 42.0
    calculator.tipPercent = 10.0

    val tip = calculator.calculateTip()

    assertEquals( expected: 4.2, tip,  delta: 1e-6)
}
```

The “Oracle” Problem

```
def test_add():
  a = 1
  b = 2

  res = add(a, b)

  assert res == 3
```

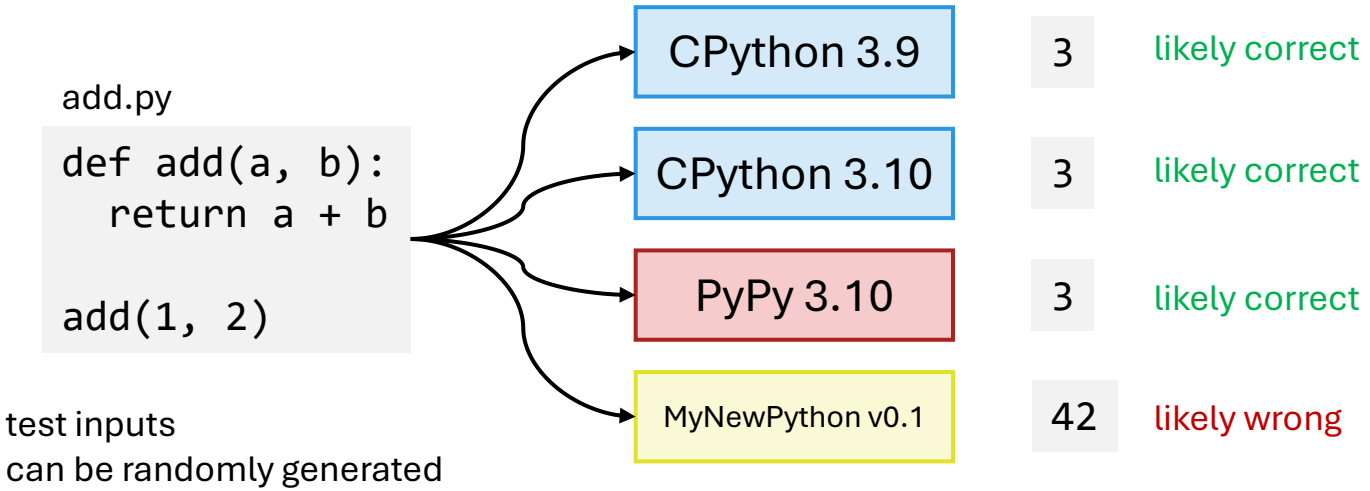
- Explicit assertions – but do we always have them?
  - underspecified requirements
  - nondeterminism
  - evolving behavior
  - large/complex objects
- Pitfalls
  - add an assertion anyways? → flaky tests or brittle tests
  - no oracle, or very weak oracle? → false confidence

Solution: test oracles w/o explicit assertion

- differential testing
- metamorphic testing
- property-based testing
- …

Differential Testing

- "If a single test is fed to several comparable programs, and one program gives a different result, a bug may have been exposed"



Metamorphic Testing

- Test oracles → metamorphic relationships
  "necessary properties of the target function or algorithm in relation to multiple inputs and their expected outputs"

```
def add(a, b):
  return a + b
```

metamorphic relationship:
$\forall a, b.\ \mathrm{add}(a, b) = \mathrm{add}(b, a)$

```
assert add(1, 2) == add(2, 1)
assert add(-3, -4) == add(-4, -3)
assert add(math.pi, math.e) == add(math.e, math.pi)
...
```

Automated Test Generation

- Writing tests can be tedious and time-consuming
  - taking ~50% of the development time
- How can we automate this?

```
def test_add():
  a = 1
  b = 2

  res = add(a, b)

  assert res == 3
```

arrange
preparing inputs

somehow generate

act
invoking CUT

enumerate all possible CUT

assert / oracle
checking outputs

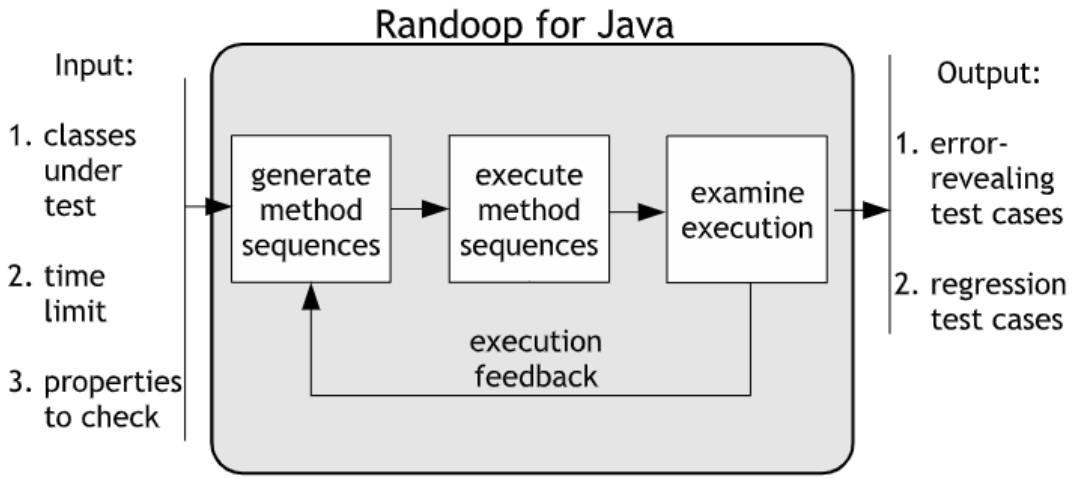
- no oracle (weak)
- assertion against current ver. (flaky/brittle)
- differential/metamorphic (not generic)
- somehow generate

Random Test Generation

- Generate random inputs for invoking the CUT
- Start from primitive types (int, float, string…)
- Assemble complex objects by invoking other methods

Randoop

Automatic unit test generation for Java




```
Object[] a = new Object[];
LinkedList ll = new LinkedList();
ll.addFirst(a);
TreeSet ts = new TreeSet(ll);
Set u = Collections.unmodifiableSet(ts);

assert u.equals(u);
```

input

oracle

Assertion fails: bug in JDK!

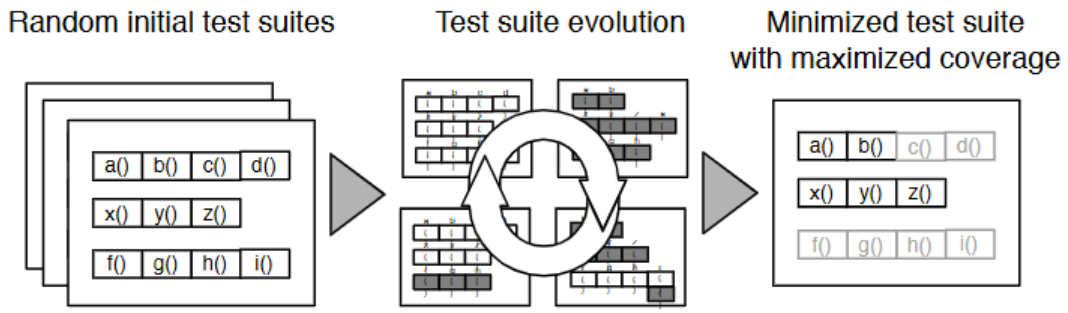
tool: https://randoop.github.io/randoop/
link to paper (ICSE'07)
Most Influcencial Paper Award at ICSE'17

Search-Based Test Generation

- Start from a random test suite
- Use genetic programming to create new test cases
- …with the goal of increasing test adequacy (code coverage, mutation score)

EVOSUITE

Automatic Test Suite Generation for Java




```
@Test(timeout = 4000)
public void testFooReturningFalse()  throws Throwable  {
    StringExample invokesFoo = new StringExample();
    boolean resultFromFoo = invokesFoo.foo("");
    assertFalse(resultFromFoo);
}
```

tool: https://www.evosuite.org/
paper1 at ISSTA'10, paper2 at QSIC'11

Machine Learning for Test Generation

- Problems of random & search-based test generation
  - not very human-readable
  - the oracle problem

no oracle (weak)
assertion against current ver. (flaky/brittle)
differential/metamorphic (not generic)
~~somehow generate~~

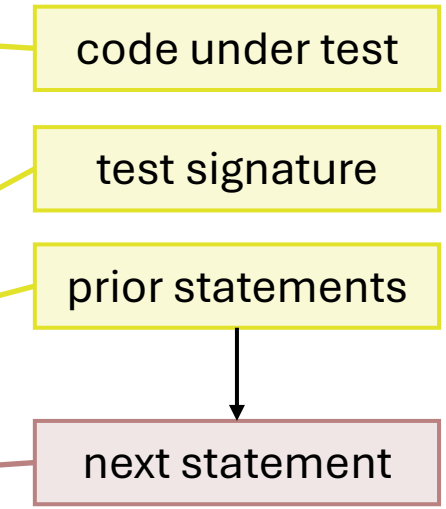
- Can we use ML models / LLMs to generate human-like tests?
  - Yes, and the research community is actively working on it!

Related work (of mine):
TeCo: ML + execution for test completion
exLong: ML + execution for generating exceptional behavior tests
(of others):
CAT-LM: pretrained model
CodaMosa: combining search-based and ML test generation
…

Task: Test Completion

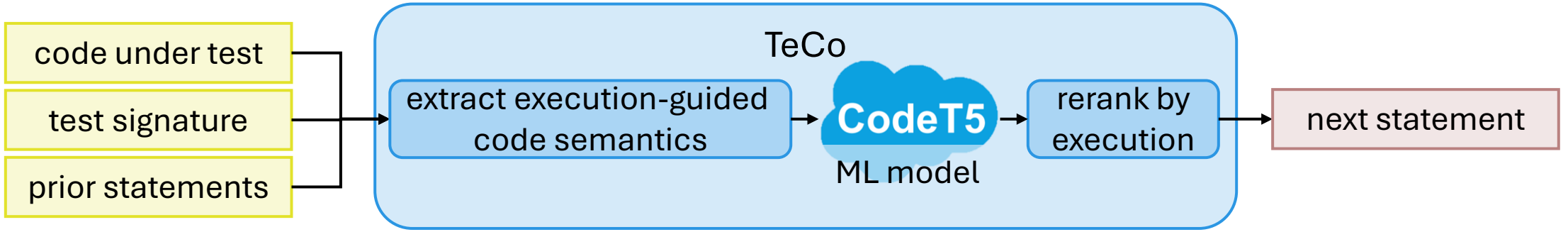
- Complete one statement at a time

```
public class GMOperation extends org.im4java.core.GMOperation {
  public GMOperation addImage(final File file) {
    if (file == null) {
      throw new IllegalArgumentException("file must be defined");
    }
    getCmdArgs().add(file.getPath());
    return this;
  }
  ...
}
public class GMOperationTest {
  @Test
  public void addImage_ThrowsException_WhenFileIsNull() throws Exception {
    exception.expect(IllegalArgumentException.class);
    ...
  }
  ...
}
```



Example from project sharneng/gm4java, GMOperationTest.java

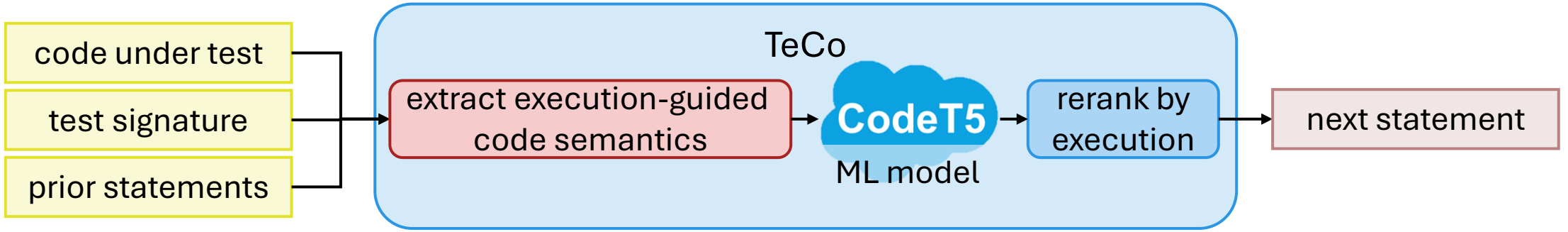
TeCo: ML + Execution for Test Completion



- Test completion can greatly benefit from reasoning about execution
  - types, program state (local and global), callable methods, etc.
  - whether the output is executable
- TeCo uses code semantics as inputs and performs reranking by test execution

Wang et al., CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In EMNLP'21.

Execution-Guided Code Semantics



- Execution results: program state after executing prior statements
  - **S1** local var types
  - **S2** absent types
  - **S3** uninitialized fields
- Execution context: code fragments relevant for predicting next statement
  - **S4** setup teardown
  - **S5** last called method
  - **S6** similar statement

Execution-Guided Code Semantics: Example

```
public class GMOperation extends org.im4java.core.GMOperation {
  public GMOperation addImage(final File file) {...}
... }

public class GMOperationTest {
  GMOperation sut;
  @Before public void setup() { ... sut = new GMOperation(); ... }
  @Test
  public void addImage_ThrowsException_WhenFileIsNull() throws Exception {
    exception.expect(IllegalArgumentException.class);
                    ?
  }
... }
```

S2 absent types

types that are required by the code under test, but are not available before executing the next statement

S4 setup teardown

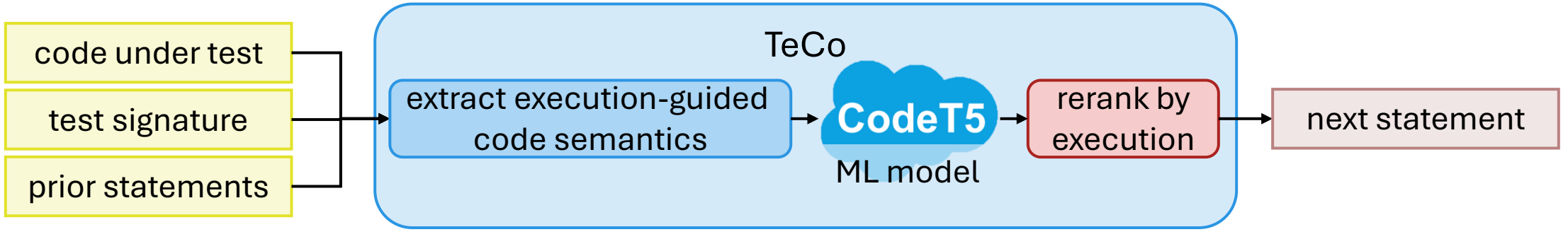
methods executed before/after the test by the testing framework

CodeT5 prediction `new GMOperation().addImage(null);` ✗

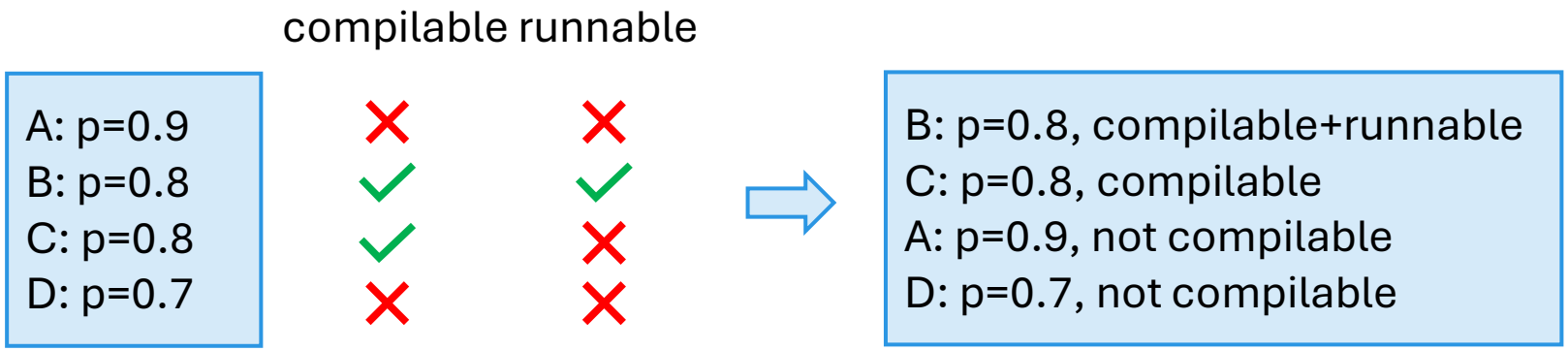
compilation error: addImage is overloaded addImage(File); addImage(Object)

TeCo prediction `sut.addImage((File) null);` ✓

Reranking by Execution



- Reranking: prioritize generating compilable and runnable statements



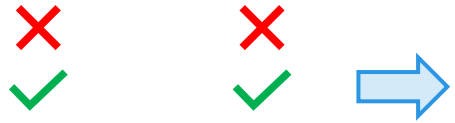
Reranking by Execution: Example

```
public class GMOperation extends org.im4java.core.GMOperation {
  public GMOperation addImage(final File file) {...}
... }

public class GMOperationTest {
  GMOperation sut;
  @Before public void setup() { ... sut = new GMOperation(); ... }
  @Test
  public void addImage_ThrowsException_WhenFileIsNull() throws Exception {
    exception.expect(IllegalArgumentException.class);
                    ?
  }
... }
```

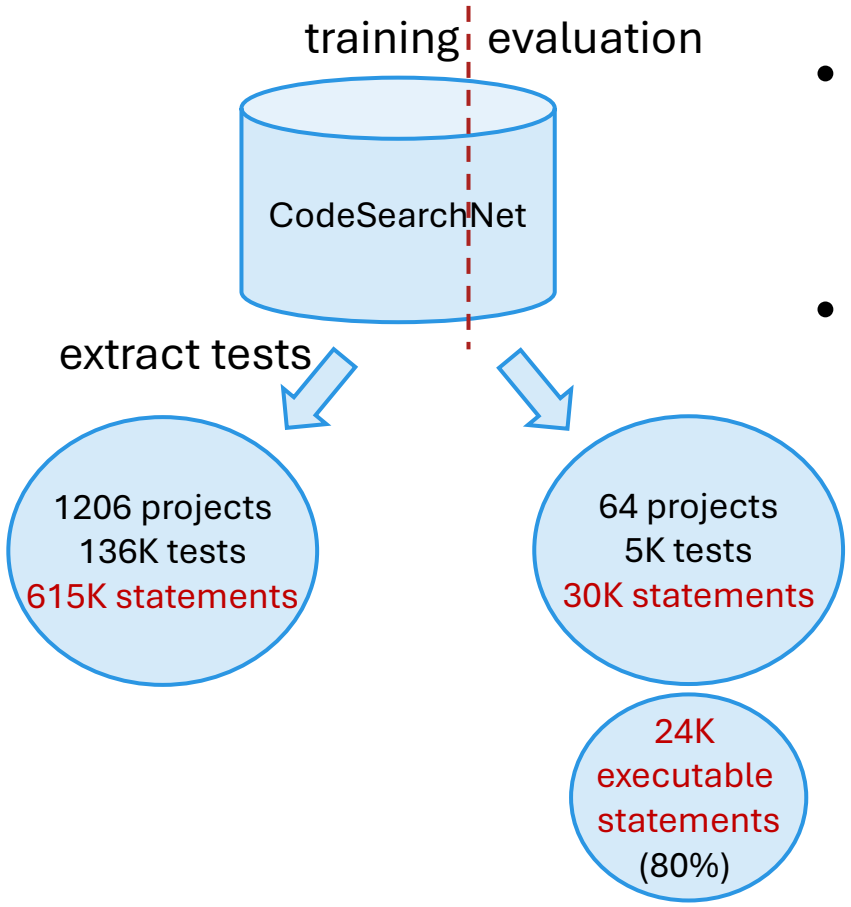
compilable runnable

```
sut.addImage(null);
sut.addImage((File) null);
...
```



```
sut.addImage((File) null);
sut.addImage(null);
...
```

Evaluation: Dataset

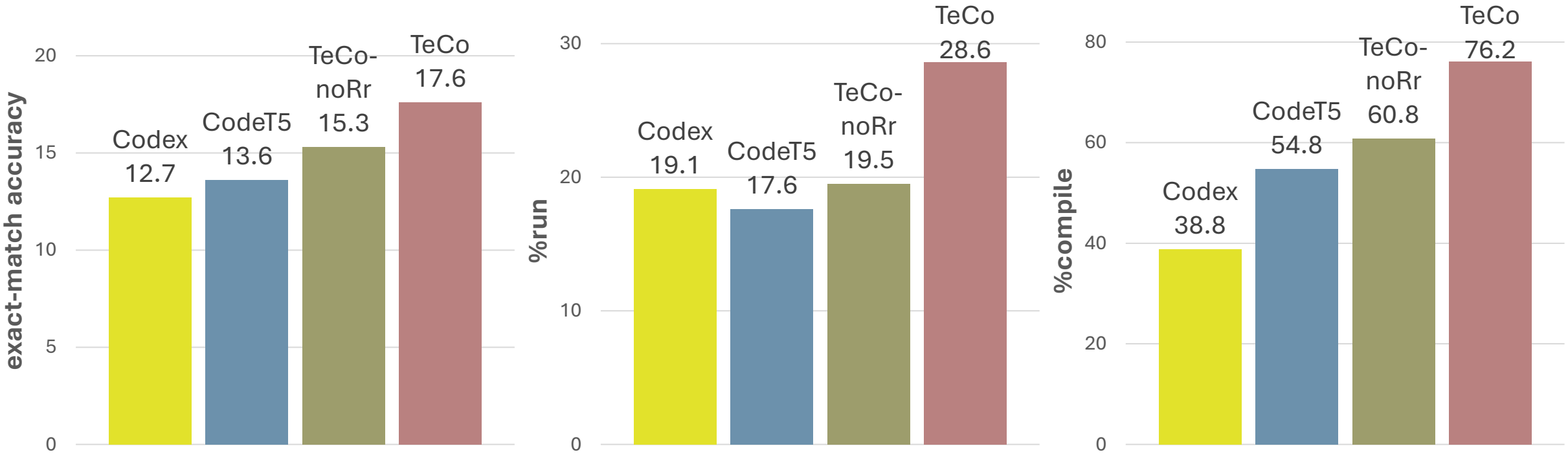


- Developer-written tests from open-source Java projects in CodeSearchNet
  - same dataset and split as used in pre-training CodeT5
- 80% of the evaluation set statements are executable
  - computing additional metrics on the executability of the output statements

Evaluation: Setup

- Metrics
  - syntax-level correctness: exact match accuracy (similarity-based metrics in paper)
  - functional correctness: %run, %compile
- Baselines
  - Codex: 175B model pre-trained on GitHub (Mar 2023)
  - CodeT5: 220M model pre-trained on CodeSearchNet, fine-tuned on our dataset
- Models
  - TeCo-noRr: code semantics + CodeT5
  - TeCo: code semantics + CodeT5 + reranking by execution
- Configurations
  - 4x Nvidia 1080Ti GPUs, Linux
  - run each experiment three times with different random seeds

Evaluation: Test Completion



TeCo improves the accuracy of test completion by 29%, and is better in generating compilable/runnable test statements

Recap

- Concepts & taxonomy
  - test case, test suite
  - unit/integration/e2e tests
- Regression testing
  - guard against future changes
  - test adequacy, code coverage
- Differential & metamorphic testing
  - the oracle problem
- Automated test generation
  - random test generation
  - search-based test generation
  - machine learning for test generation

Research Topics Not Covered

- Reducing testing cost
  - regression test selection
  - test suite minimization/reduction
  - test case prioritization
- Other automated testing approaches
  - property-based testing
  - symbolic/concolic execution
  - model checking
- Detecting and fixing flaky tests
- Fuzzing
- Test smells & maintenance

Pengyu Nie <pynie@uwaterloo.ca>