Software Testing & Machine Learning

Pengyu Nie <pynie@uwaterloo.ca>

Agenda

- Background on software testing
- Machine learning for software testing

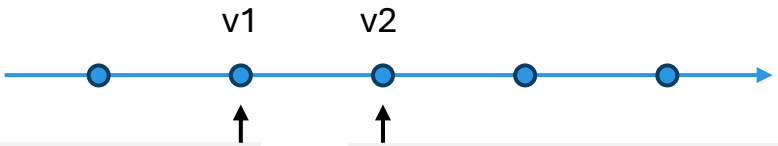
What is Testing and Why

- "The process of evaluating and verifying that a software product or application does what it's supposed to do"
- Why
  - Prevent bugs (from troubling users of software)
  - Ensure software quality
  - Improve performance
- Takes ~50% of software development time!

Source: https://www.ibm.com/topics/software-testing

Regression Testing

- Regression testing focuses on finding defects after a major code change has occurred




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

```
def add(m: float, n: float) -> float:
  return m + n
```

```
def test_add_int():
  assert add(1, 2) == 3
```

```
def test_add_int():
  assert add(1, 2) == 3
```

```
def test_add_float():
  assert add(1.0, 3.2) == 4.2
```

Arrange, Act, Assert

arrange
prepare test inputs

act
invoke code under test

assert
aka **test oracles**
check expected outcomes

name

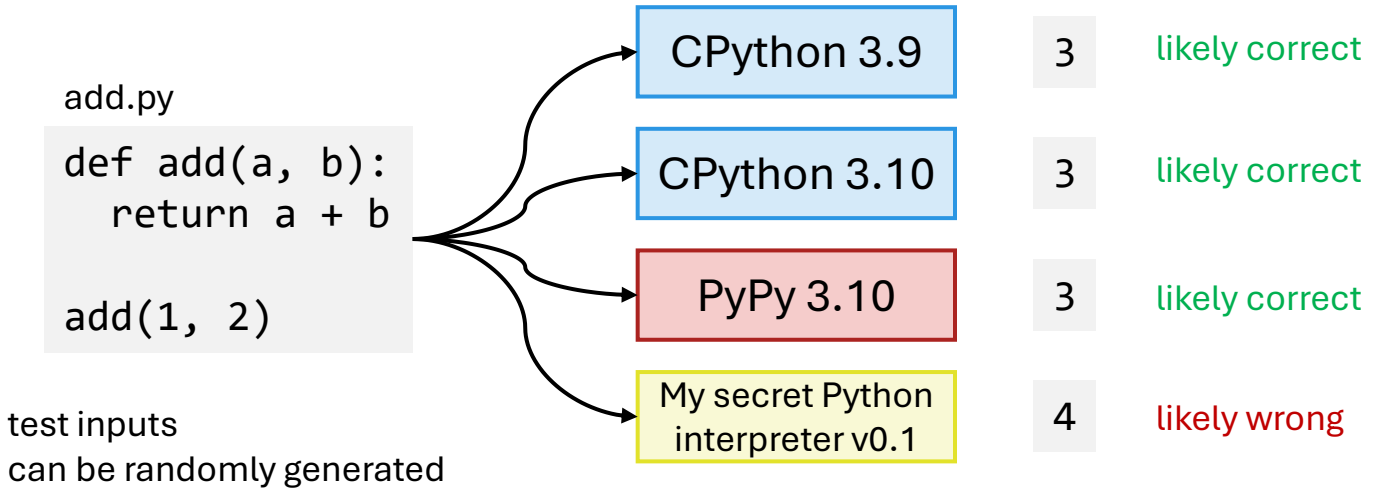
```
def test_duckduckgo_instant_answer_api_search():

  url = 'https://api.duckduckgo.com/?q=python+programming&format=json'

  response = requests.get(url)
  body = response.json()

  assert response.status_code == 200
  assert 'Python' in body['AbstractText']
```

What if we don't have the test oracles?

Source: https://automationpanda.com/2020/07/07/arrange-act-assert-a-pattern-for-writing-good-tests/

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"



Source: https://www.cs.swarthmore.edu/~bylvisa1/cs97/f13/Papers/DifferentialTestingForSoftware.pdf

Metamorphic Testing

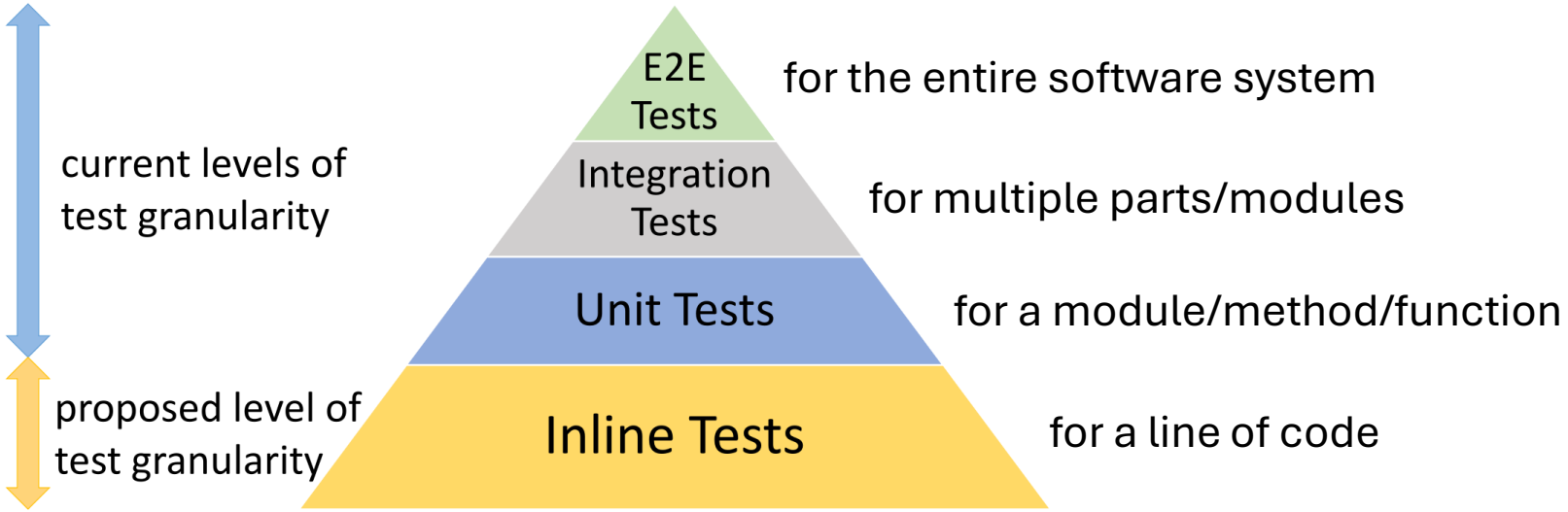
- Test oracles → metamorphic relationships
  "necessary properties of the target function or algorithm in relation to multiple inputs and their expected outputs"

```
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)
...
```

Granularities of Testing



Yu Liu, **Pengyu Nie**, Owolabi Legunsen, and Milos Gligoric. Inline Tests. In ASE'22.

Test {Selection, Minimization, Prioritization}

- What if we have a lot of tests to run?
- Regression test selection / RTS
  - Only run the subset of tests that are affected by code changes
- Test minimization
  - Remove the redundant tests which do not lead to any benefit (in terms of code coverage / mutation score)
- Test prioritization
  - Execute the important tests first (e.g., related to code changes, higher contributions to code coverage / mutation score)
    in the hope that they will find bugs early

Machine Learning for Software Testing

Learning Deep Semantics for Test Completion

work by **Pengyu Nie**, Rahul Banerjee, Junyi Jessy Li, Raymond J. Mooney, and Milos Gligoric. In ICSE'23.

Motivation: Writing Tests is Tedious

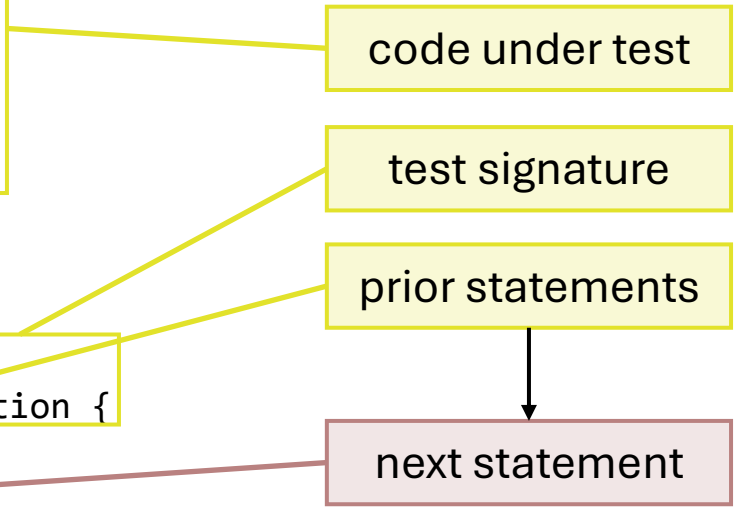
- Testing is the most frequently-used technique to ensure software correctness
- Writing tests can take a lot of manual efforts (~50% of development time)
- Automatically generated tests (e.g., by random testing) have stylistic issues and do not replace the need of manual efforts

Goal: developing ML models to assist developers in writing tests

Task: Test Completion

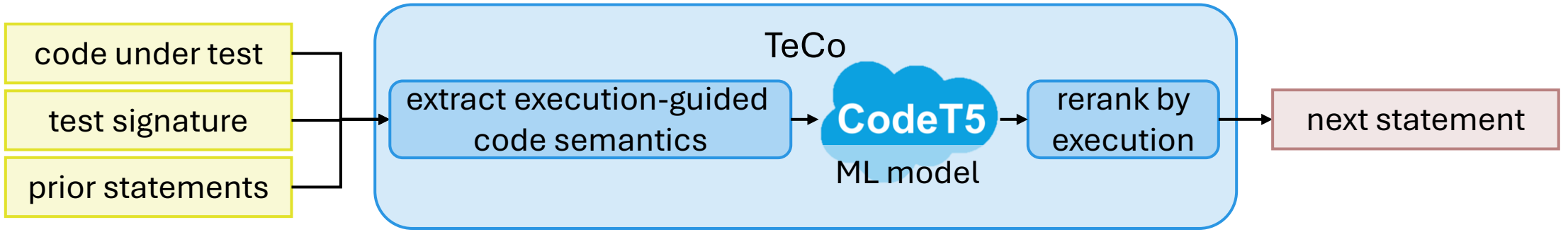
- Complete one statement at a time

```
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

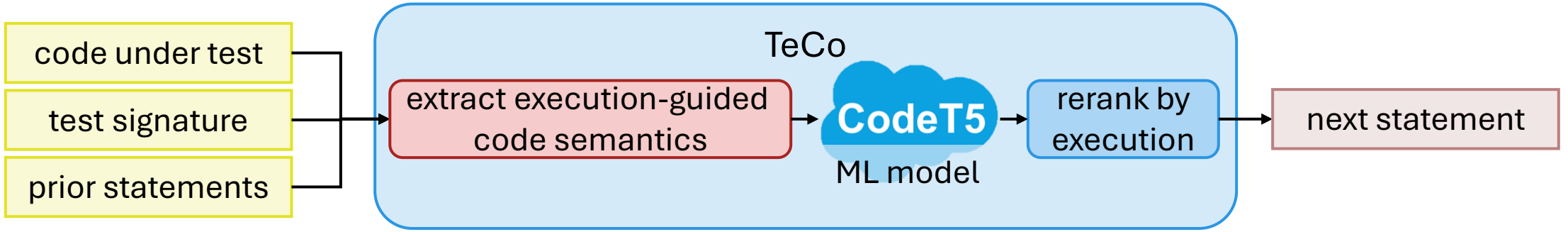
TeCo: ML + Execution for Test Completion



- 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

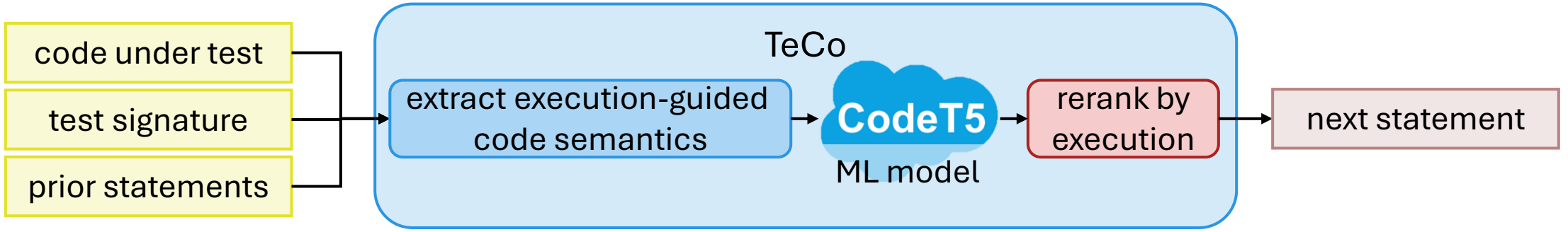
methods executed before/after the test by the testing framework

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

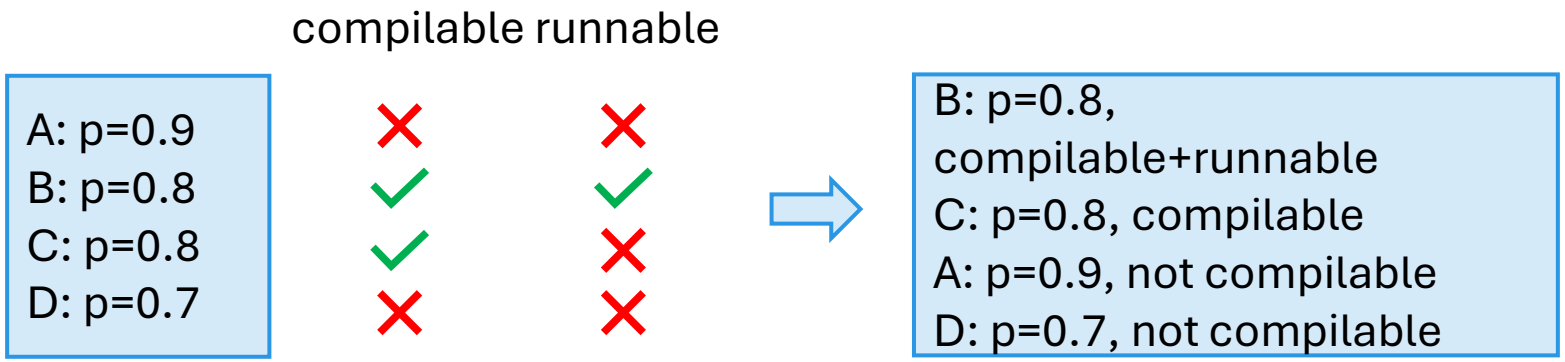
compilation error: addImage is overloaded addImage(File); addImage(Object)

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

Reranking by Execution



- Reranking: prioritize generating compilable and runnable statements



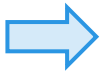
Reranking by Execution: Example

```
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

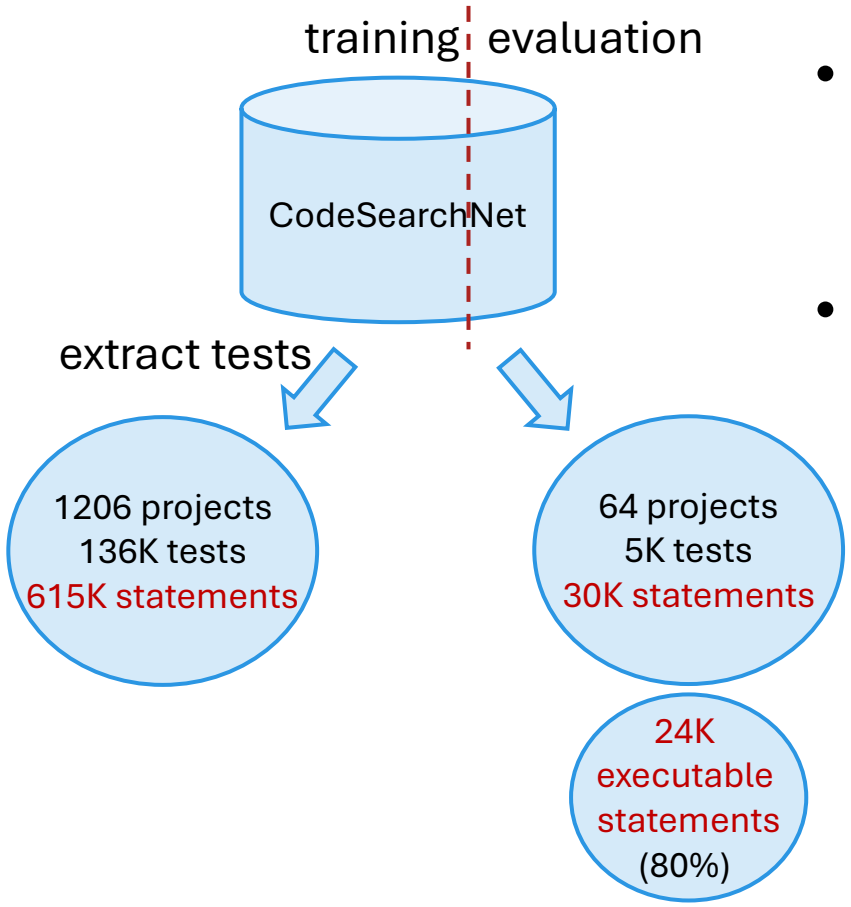
```
sut.addImage(null);
sut.addImage((File) null);
...
```





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

Evaluation: Dataset

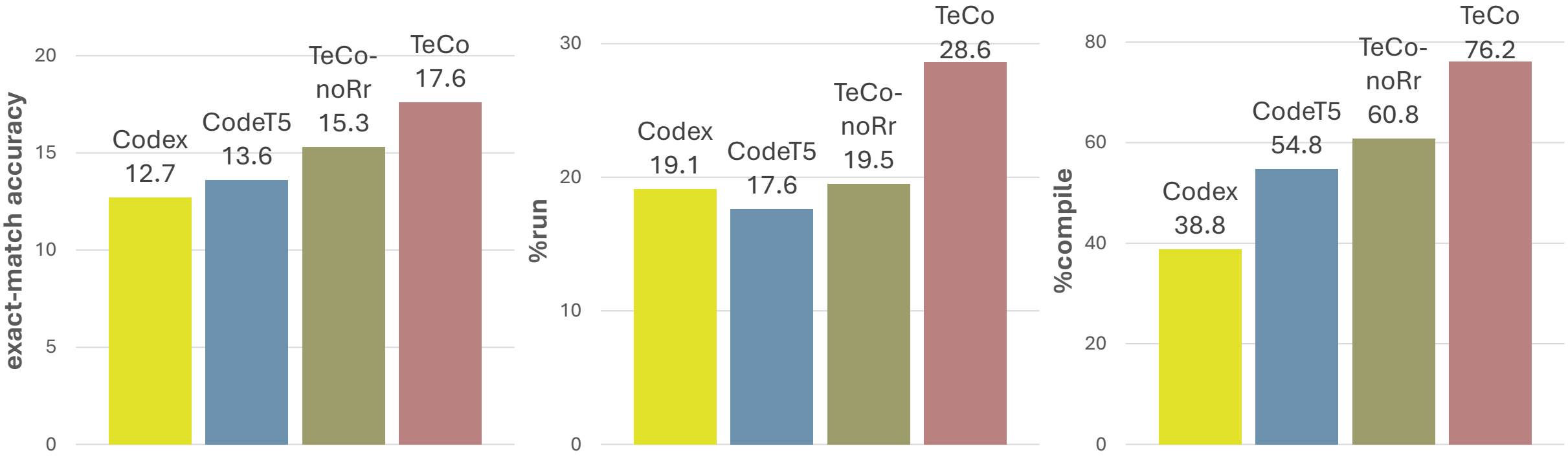


- Developer-written tests from open-source Java projects in CodeSearchNet
  - same dataset and split as used in pre-training CodeT5
- 80% of the evaluation set statements are executable
  - computing additional metrics on the executability of the output statements

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