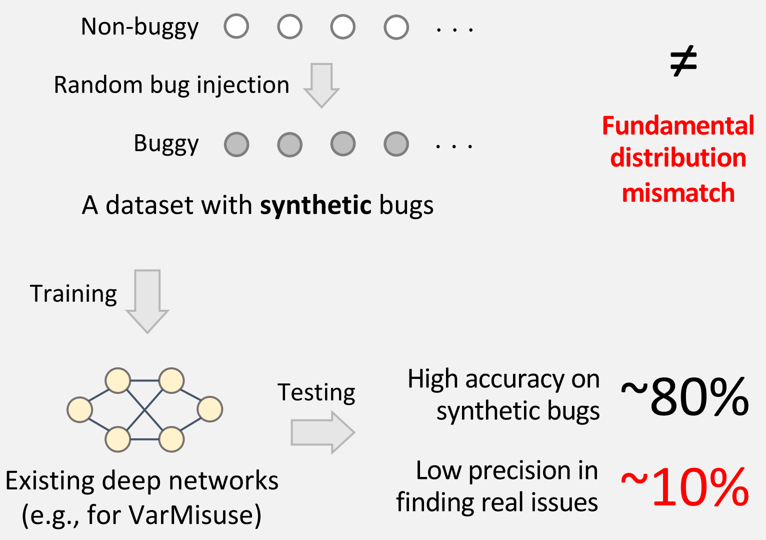
Learning to Find Naming Issues with Big Code and Small Supervision

Jingxuan He, Cheng-Chun Lee, Veselin Raychev, Martin Vechev

SRILAB ETH zürich DEEPCODE by snyk EPFL

PLDI 2021

Existing Scheme of Learning a bug detector



Non-buggy ○ ○ ○ ○ . . .

Random bug injection

Buggy ● ● ● ● . . .

A dataset with **synthetic** bugs

Training

Testing

Existing deep networks (e.g., for VarMisuse)

High accuracy on synthetic bugs ~80%

Low precision in finding real issues ~10%

≠

Fundamental distribution mismatch

A **large labeled** dataset with **real** bugs (does not exist in practice)

Our new learning method:

unsupervised anomaly detection
+
supervised learning from small manually labeled data

(no distribution mismatch)

Instantiation: finding naming issues

Namer: combining two learning schemes

Big code (unlabeled)

Pattern mining

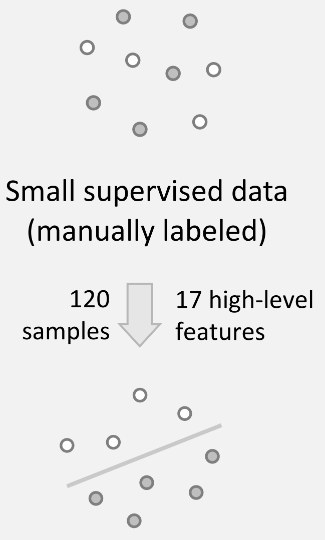
Condition

```
NumArgs(2) 0  Call 0  AttributeLoad 0  NameLoad 0  NumST(1) 0  TestCase 0  self
NumArgs(2) 0  Call 0  AttributeLoad 1  Attr 0  NumST(2) 0  TestCase 0  assert
NumArgs(2) 0  Call 2  Num 0  NumST(1) 0  NUM
```

Deduction

```
NumArgs(2) 0  Call 0  AttributeLoad 1  Attr 0  NumST(2) 1  TestCase 0  Equal
```

Name patterns: capture common naming idioms



Small supervised data (manually labeled)

120 samples

17 high-level features

Defect classifier

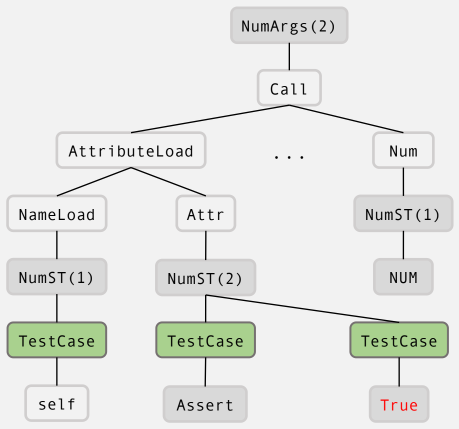
Check violation of name patterns

Query defect classifier

Pipeline of Namer

```
class TestPicture(TestCase):
  ...
  def test_angle_picture(self):
    rotated_picture_name = "IMG_2259.jpg"
    for picture in self.slide.pictures:
      if picture.relative_path \\
          == rotated_picture_name:
        picture = self.slide.pictures[0]
        self.assertTrue(picture.rotate_angle, 90)
        break
```

Parsing
Transformations
Static analyses



Extract name paths

```
NumArgs(2) 0  Call 0  AttributeLoad 0  NameLoad 0  NumST(1) 0  TestCase 0  self
NumArgs(2) 0  Call 0  AttributeLoad 1  Attr 0  NumST(2) 0  TestCase 0  assert
NumArgs(2) 0  Call 0  AttributeLoad 1  Attr 0  NumST(2) 1  TestCase 0  True
NumArgs(2) 0  Call 2  Num 0  NumST(1) 0  NUM
...
```

issue: `assertTrue`

fix: `assertEqual`

Evaluating Namer

GitHub

~33k Python repos*
~1 million source files
(deduplicated)

Pattern mining & matching

~65k patterns mined

~500k violations triggered

~90% of repos and files have violations

Run classifier on 300 violations

5 semantic defects

89 code quality issues

40 false positives

70% precision

Precision comparison

Namer: 70%

w/o classifier : 46%

w/o analyses : 59%

w/o both: 40%

Classifier and analyses are important

Examples

Semantic defects:

```
self.assertEquals(3, val)
for i in xrange(10)
```

Code quality issues:

```
num_or_process = 3
def evolve(..., **args):
```

>86% chance accepted by professional developers at coding time in an IDE

* We also evaluated Namer on a large Java dataset. See paper.