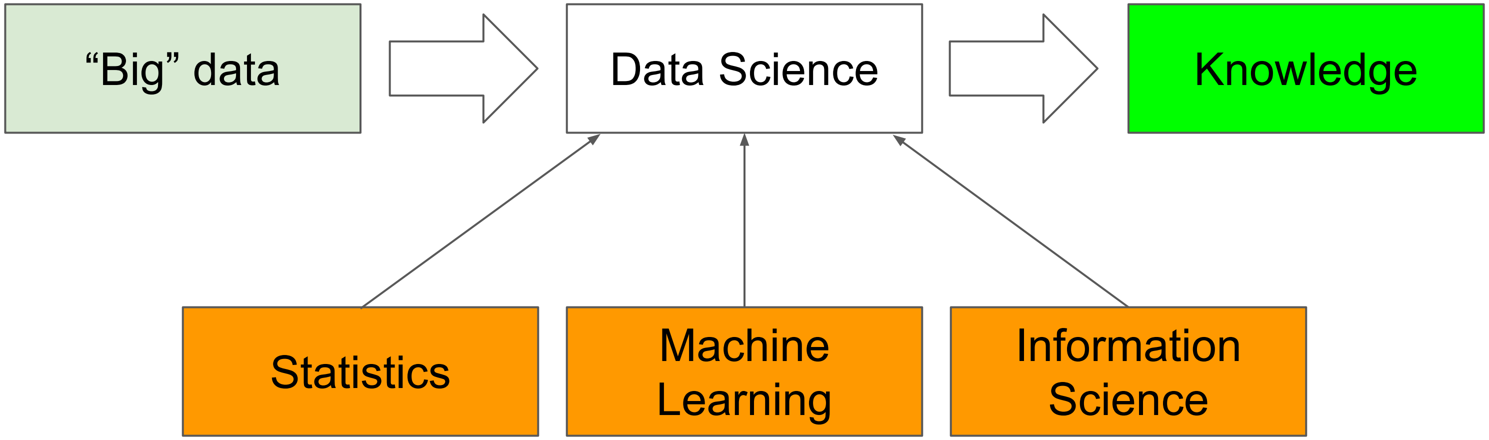
Thesis Defence

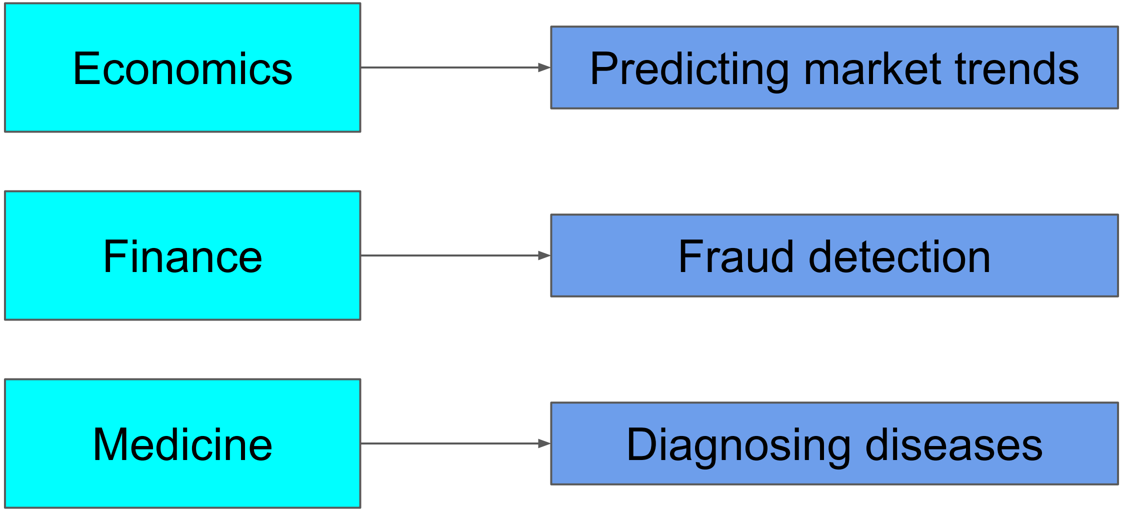
Test Case Generation and Fault Localization for Data Science Programs

Mohammad Rezaalipour

USI Università della Svizzera italiana
Lugano, Switzerland

June 13, 2024

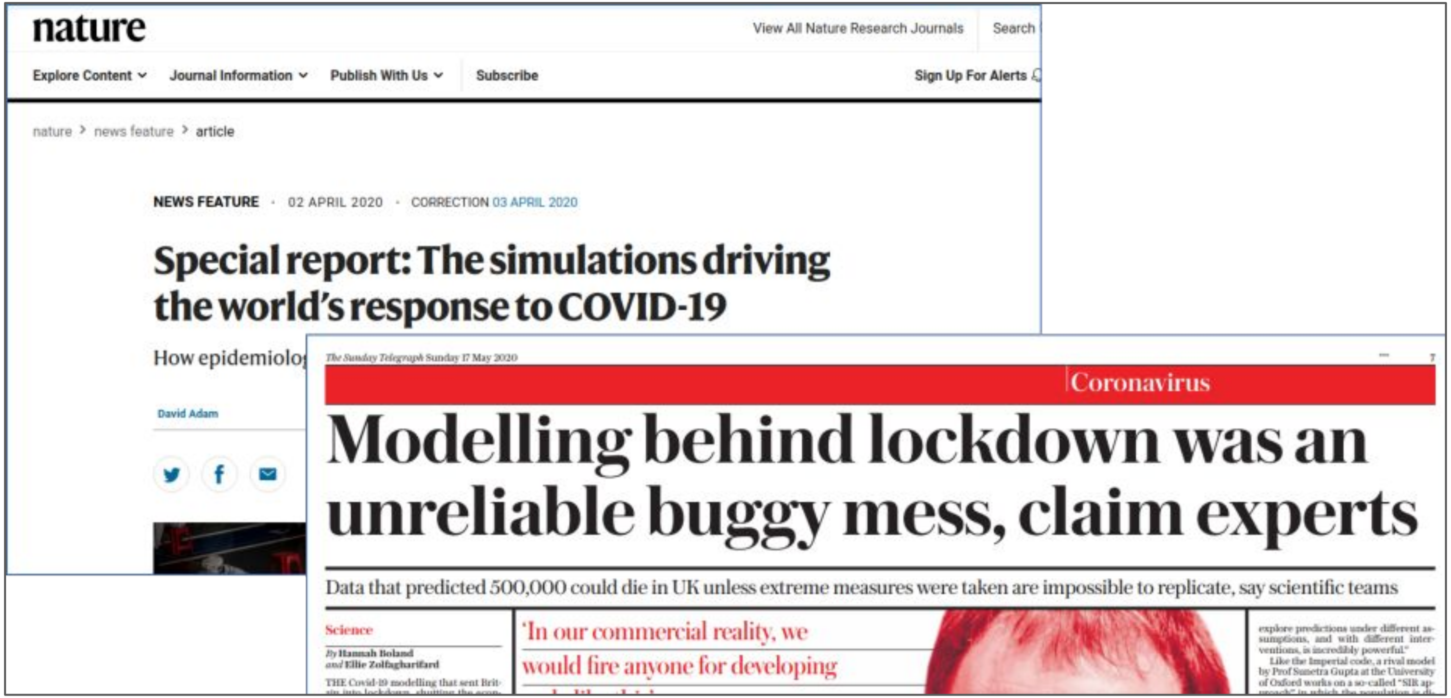

"Big" data
Data Science
Knowledge
Statistics
Machine Learning
Information Science


Economics
Predicting market trends
Finance
Fraud detection
Medicine
Diagnosing diseases

Neural Networks implemented as programs


```
def dense_block(
   x, nb_layers, nb_channels, growth_rate,
   dropout_rate=None, bottleneck=False,
):

   x_list = [x]
   for i in range(nb_layers):
       cb = convolution_block(x, growth_rate,
                              Dropout_rate,
                              bottleneck)
       x_list.append(cb)
       x = Concatenate(axis=-1)(x_list)
       nb_channels += growth_rate
   return x, nb_channels
```

Written by domain experts who may not be professional programmers



Coronavirus pandemic simulation has been criticized as being unreliable and buggy

An Empirical Study on Program Failures of Deep Learning Jobs

Ru Zhang
Microsoft Research
v-ruzha@microsoft.com

Wencong Xiao*
Alibaba Group
wencong.xwc@alibaba-inc.com

Hongyu Zhang
The University of Newcastle
hongyu.zhang@newcastle.edu.au

Yu Liu
Microsoft Research
v-yucli@microsoft.com

Haoxiang Lin†
Microsoft Research
haoxlin@microsoft.com

Mao Yang
Microsoft Research
maoyang@microsoft.com

ABSTRACT

Deep learning has made significant achievements in many application areas. To train and test models more efficiently, enterprise developers submit and run their deep learning programs on a shared, multi-tenant platform. However, some of the programs fail after a long execution time due to code/script defects, which reduces the

ACM Reference Format:
Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. 2020. An Empirical Study on Program Failures of Deep Learning Jobs. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380362

"A noticeable percentage [...] threw runtime exceptions due to code or script defects [...]"

An empirical study on TensorFlow program bugs (Zhang et al. 2018)

A Comprehensive Study on Deep Learning Bug Characteristics (Islam et al. 2019)

Taxonomy of Real Faults in Deep Learning Systems (Humbatova et al. 2019)

Repairing deep neural networks: Fix patterns and challenges (Islam et al. 2020)

Bugs with characteristics different from traditional software

Thesis statement

Understanding the capabilities and limitations of standard test generation and fault localization techniques on data science programs implemented in dynamic languages such as Python informs the development of new techniques that can be more effective.

Contributions

Part 1

- Test generation approach for neural network programs
- Test generation tool
- Curated dataset of neural network of bugs

Part 2

- Empirical study of fault localization in Python programs
- Fault localization tool

Part 1

- Test generation approach for neural network programs
- Test generation tool
- Curated dataset of neural network of bugs

Part 2

- Empirical study of fault localization in Python programs
- Fault localization tool

Generating tests for Python NN programs

Domain specific parameter types

```
def DenseNet(input_shape=None, dense_blocks=3, dense_layers=-1,
             growth_rate=12, nb_classes=None, dropout_rate=None,
             bottleneck=False, compression=1.0, weight_decay=1e-4,
             depth=40):
```

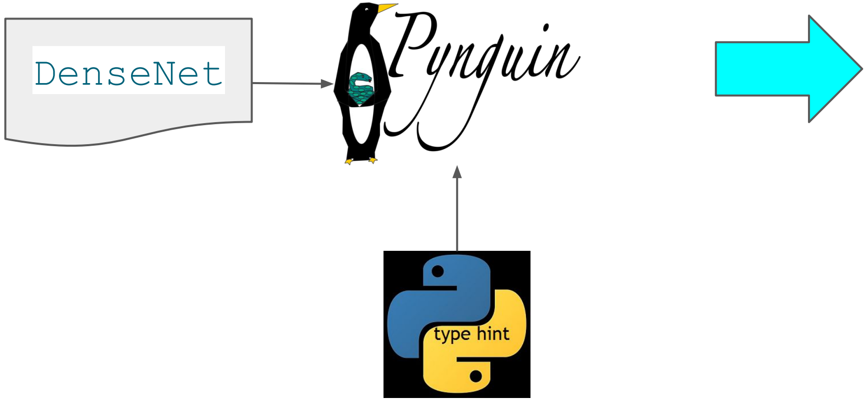
https://github.com/cmasch/densenet/

```python
def DenseNet(input_shape=None, dense_blocks=3, dense_layers=-1,
             growth_rate=12, nb_classes=None, dropout_rate=None,
             bottleneck=False, compression=1.0, weight_decay=1e-4,
             depth=40):

   if nb_classes==None:
       raise Exception('Please define number of classes.')

   if compression <=0.0 or compression > 1.0:
       raise Exception('Compression must be between 0.0 and 1.0.')

   if type(dense_layers) is list:
       if len(dense_layers) != dense_blocks:
           raise AssertionError('Dense blocks must be the same as layers')
   elif dense_layers == -1:
       dense_layers = (depth - 4)/3

    ...
```

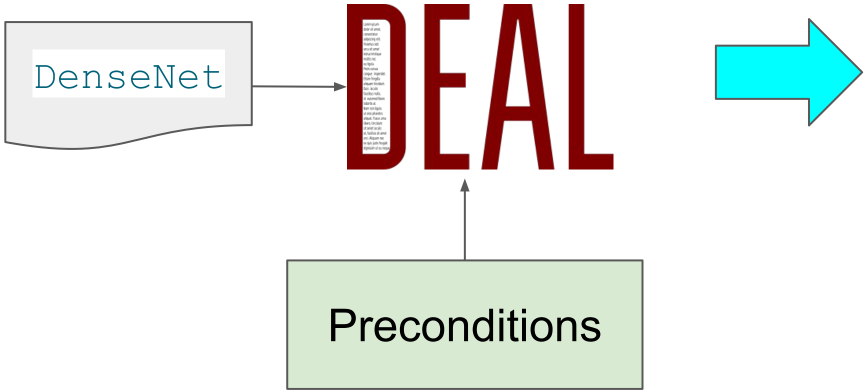
Bug (points to `dense_layers = (depth - 4)/3`)

https://github.com/cmasch/densenet/

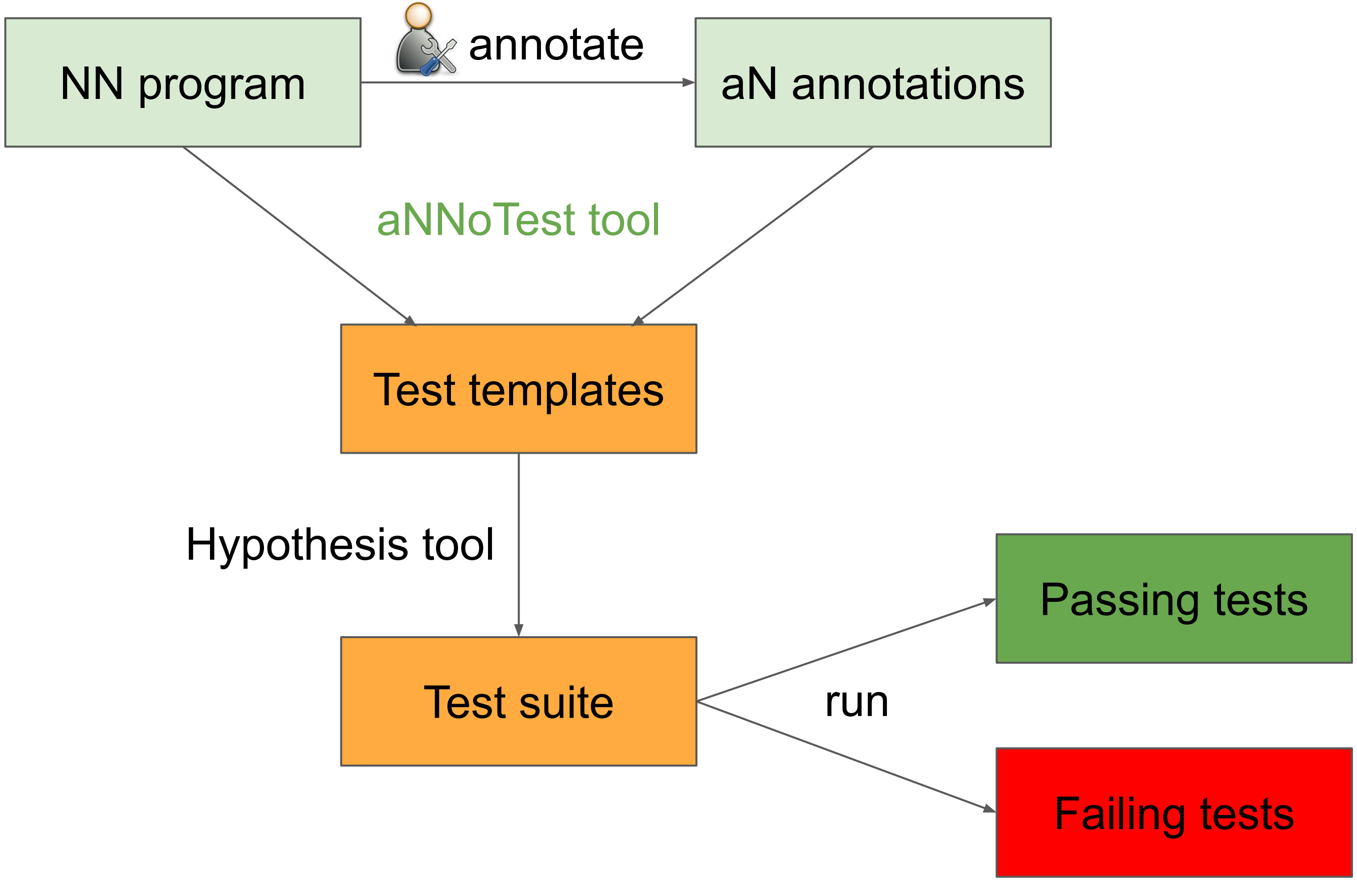
8 tests
all invalid
none triggering the failure



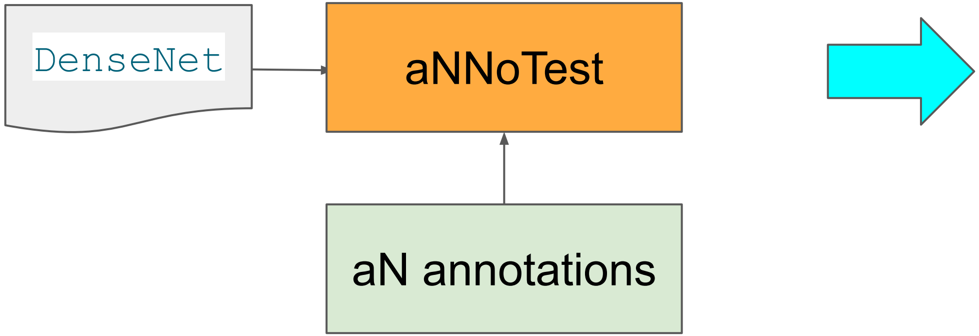
5 tests
4 invalid
none triggering (any) failure



No tests


NN program
annotate
aN annotations
aNNoTest tool
Test templates
Hypothesis tool
Test suite
run
Passing tests
Failing tests

```
@arg(input_shape): tuples(ints(min=20, max=70),
                          ints(min=20, max=70),
                          ints(min=1, max=3))

@arg(dense_blocks): ints(min=2, max=5)

...

def DenseNet(input_shape=None, dense_blocks=3, dense_layers=-1,
             growth_rate=12, nb_classes=None, dropout_rate=None,
             bottleneck=False, compression=1.0, weight_decay=1e-4,
             depth=40):
```



36 tests
all valid
one triggering the failure

Experimental Evaluation of aNNoTest

Precision: Does aNNoTest generate tests that expose bugs with few false positives (invalid tests)?

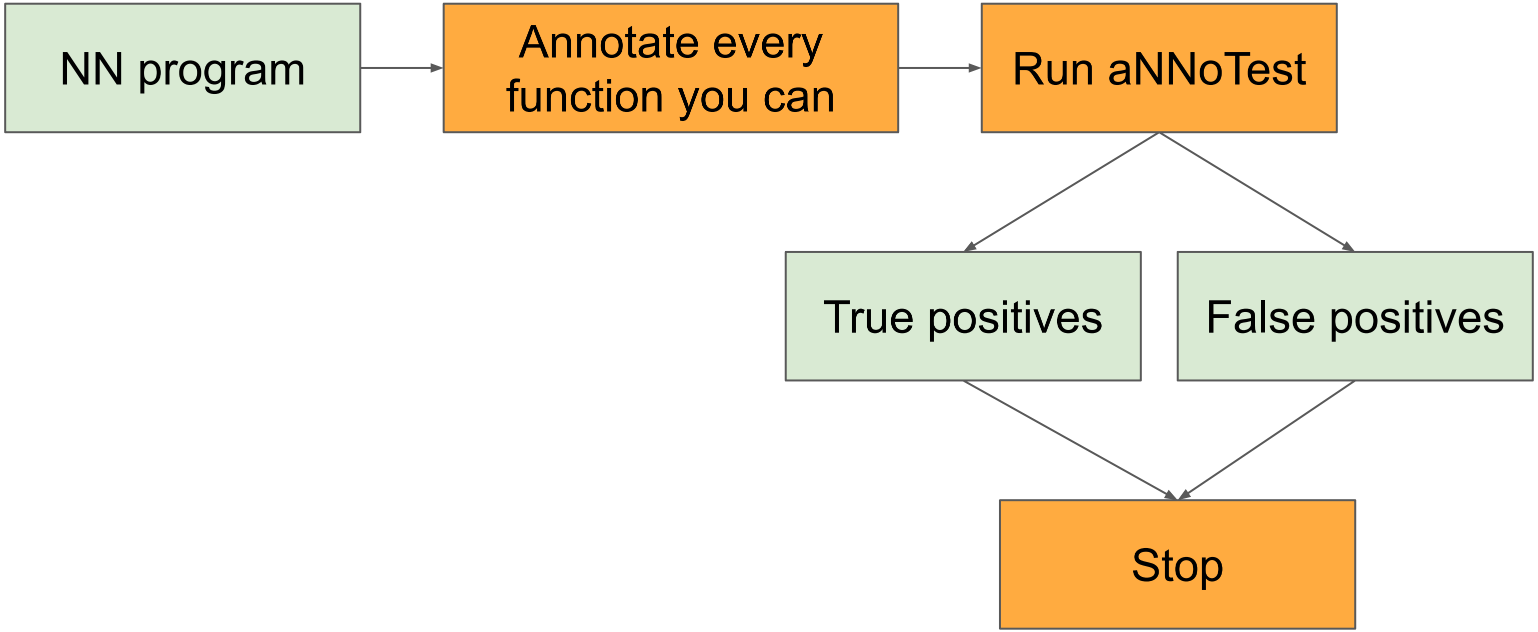
Recall: Can aNNoTest reproduce known, relevant bugs (that were discovered and confirmed by expert manual analysis)?

Experimental Subjects

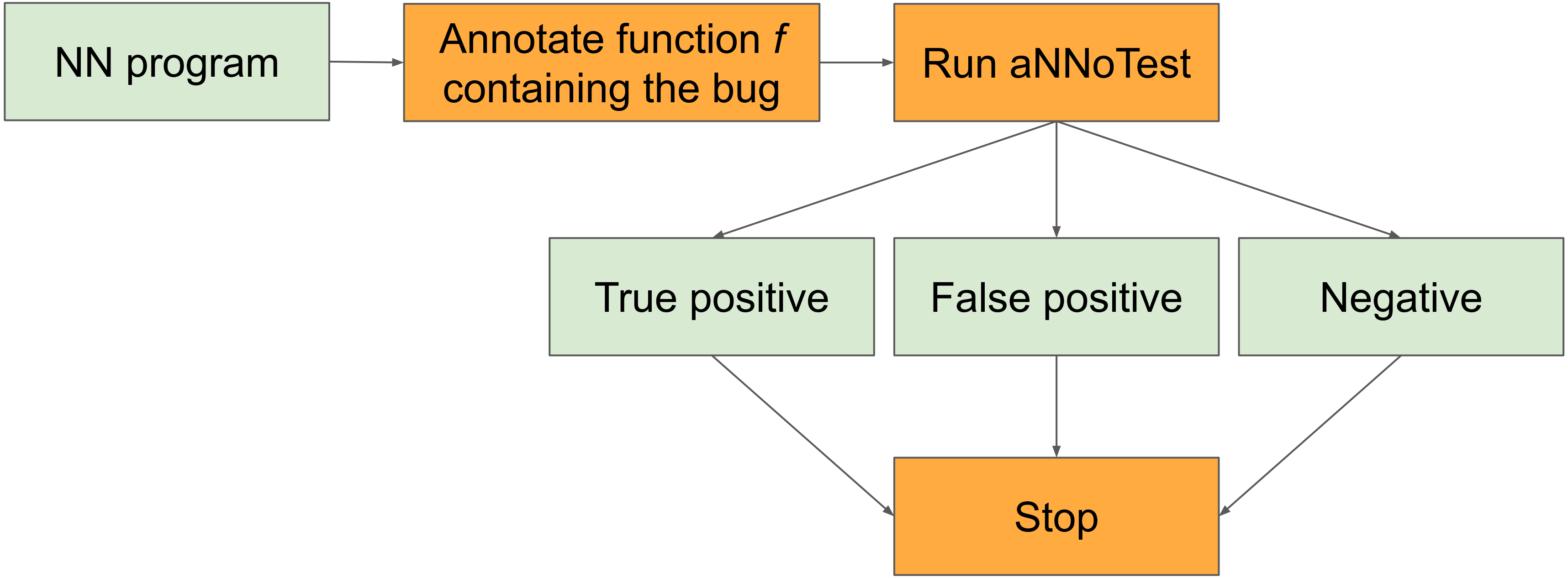
	Projects	LOC	Avg. Number of Annotations	Known Bugs
Subjects P	2	3917	1.3	?
Subjects R	19	14219	6.0	81

K Keras TensorFlow PyTorch

Precision


NN program
Annotate every function you can
Run aNNoTest
True positives
False positives
Stop

Recall


NN program
Annotate function *f* containing the bug
Run aNNoTest
True positive
False positive
Negative
Stop

Experimental Results

	Known Bugs	Found Bugs	Spurious	Precision	Recall
Subjects P	0	50	6	89%	?
Subjects R	81	63	0	100%	78%

mohrez86 Add citation file — 326ffd7 · 10 months ago — 23 Commits

annotest	Blacken app.py and add code to blacken generated tests	10 months ago
tests	Blacken tests package	10 months ago
.gitignore	Squash dev to main	last year
CHANGELOG.md	Add changelog file	10 months ago
CITATION.cff	Add citation file	10 months ago
LICENSE	Initial commit	last year
README.md	Add new pip command to install from GitHub	10 months ago
pyproject.toml	Add pyproject.toml	last year
requirements.txt	Add requirements	10 months ago
setup.py	Blacken setup.py and add requirements and modify desc...	10 months ago

README GPL-3.0 license

aNNoTest

pip install annotest

pypi package 0.1 | license GPL-3.0 | downloads 1k | code style black

https://github.com/atom-sw/annotest

Part 1

- Test generation approach for neural network programs
- Test generation tool
- Curated dataset of neural network of bugs

Part 2

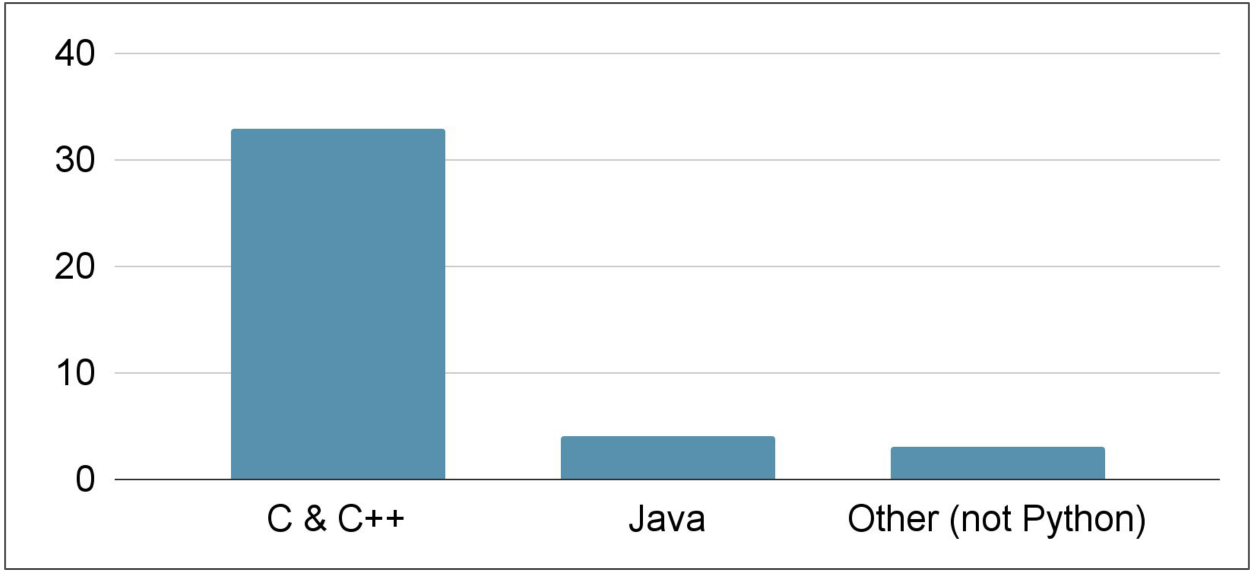
- Empirical study of fault localization in Python programs
- Fault localization tool

What is Fault Localization?

Line number	Score
23	0.7
13	0.3
21	0.3
124	0.2

May 2024	May 2023	Change	Programming Language
1	1		Python
2	2		C
3	4	^	C++
4	3	˅	Java
5	5		C#
6	7	^	JavaScript

https://www.tiobe.com/tiobe-index



Subject programs (1977 - 2014)

Wong et al. (2016, Table 6)

One SBFL empirical study (Widyasari et al. 2022)

One SBFL tool (Sarhan et al. 2021)

An Empirical Study of Fault Localization Families and Their Combinations

Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D. Ernst, and Lu Zhang

Abstract—The performance of fault localization techniques is critical to their adoption in practice. This paper reports on an empirical study of a wide range of fault localization techniques on real-world faults. Different from previous studies, this paper (1) considers a wide range of techniques from different families, (2) combines different techniques, and (3) considers the execution time of different techniques. Our results reveal that a combined technique significantly outperforms any individual technique (200 percent increase in faults localized in Top 1), suggesting that combination may be a desirable way to apply fault localization techniques and that future techniques should also be evaluated in the combined setting. Our implementation is publicly available for evaluating and combining fault localization techniques.

Index Terms—Fault localization, learning to rank, program debugging, software testing, empirical study

Differentiated conceptual replication

The first multi-family large-scale empirical study of fault localization in open-source Python programs

Spectrum-based (SBFL)

DStar
Ochiai
Tarantula

Mutation-based (MBFL)

Metallaxis
Muse

Predicate switching (PS)

Stack trace (ST)

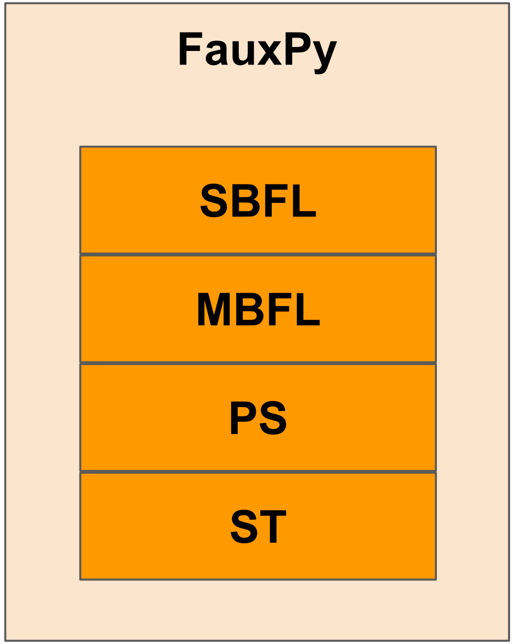
Combined

CombineFL
AvgFL

Statement

Function

Module


FauxPy
SBFL
MBFL
PS
ST

pip install fauxpy

https://github.com/atom-sw/fauxpy

Comparing the effectiveness and efficiency of fault localization techniques

Effectiveness

Top-1	0
Top-3	0
Top-5	1
Top-10	1
Exam score	4/10
Location List Length	4

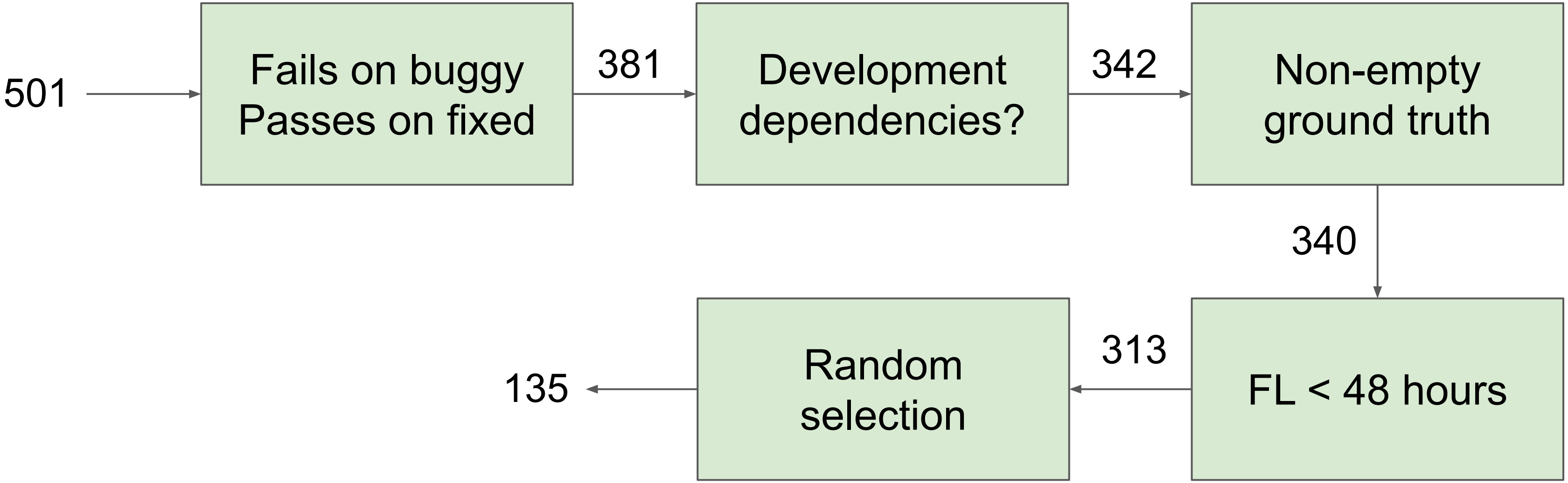
Program size = 10

Line number	Score	Rank
23	0.7	1
13	0.3	3
21	0.3	3
124	0.2	4

Efficiency

Wall-clock running time

Subjects	Projects	kLOC	Tests	Faults
BugsInPy	17	714.0	24 817	501
Selected	13	515.4	18 882	135



Subjects	Projects	kLOC	Tests	Faults
BugsInPy	17	714.0	24 817	501
Selected	13	515.4	18 882	135

httpie
Black
nvbn/thefuck
TQDM
Keras
SANIC FRAMEWORK
pandas
Tornado
COOKIECUTTER
FastAPI
Luigi
spaCy: Industrial-strength NLP
EXPLOSION
YouTube-DL

Project category

- Command line (CL)
- Development (DEV)
- Data science (DS)
- Web (WEB)

Bug kind

- Crashing
- Predicate
- Mutable

Summary of findings on the fault localization empirical study

Effectiveness

SBFL > MBFL ≫ PS ≃ ST

Family	% of bugs in Top-5
MBFL	27
PS	7
SBFL	43
ST	6

Effectiveness

Combined > SBFL > MBFL ≫ PS ≃ ST

Family	% of bugs in Top-5
MBFL	27
PS	7
SBFL	43
ST	6
Combined	49

Efficiency

ST ≫ SBFL ≫ PS > MBFL

Family	Time (s)
MBFL	15774
PS	9751
SBFL	589
ST	2

Project category

Bugs in data science (DS) projects challenge fault localization

Family	% of bugs in Top-5			
	CL	DEV	DS	WEB
MBFL	38	28	19	20
PS	5	10	7	5
SBFL	60	37	30	40
ST	9	10	0	5

Most findings replicate

⇨ Java = Python

Python vs Java

Python: ST ≃ PS

Java: ST > PS

Family	Top-1%		Top-3%		Top-5%		Top-10%	
	Python	Java	Python	Java	Python	Java	Python	Java
PS	3	1	5	4	7	6	7	6
ST	0	6	4	9	6	11	13	11

Conclusions and future work

Many bugs

Bugs with different characteristics

Domain specific data types

Contributions

- The aNNoTest approach
- The aNNoTest tool
- Curated dataset of neural network of bugs

- Empirical study of fault localization in Python programs
- The FauxPy tool

Future work: Test generation

```
@arg(model_g): objs(gen_model_g)
@arg(model_d): objs(gen_model_d)

def build_gan(model_g, model_d, name="gan"):
    # ...
```

Extending argument constraints

```
@arg(model_g): keras_models(par_a1, par_a2, ...)
@arg(model_d): keras_models(par_b1, par_b2, ...)

def build_gan(model_g, model_d, name="gan"):
    # ...
```

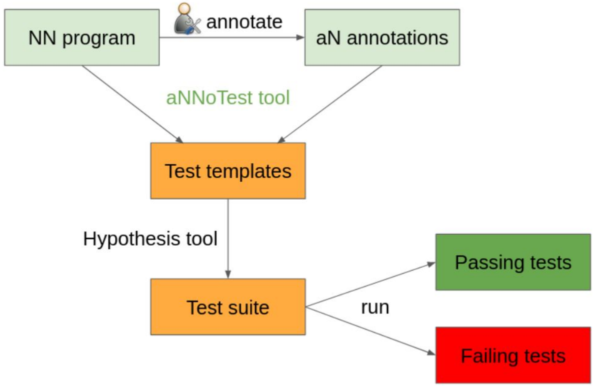
Future work: Fault localization

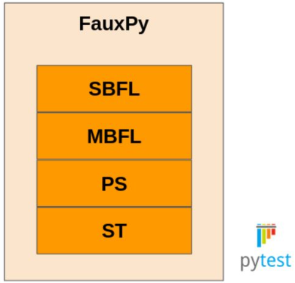
No mutations on buggy statement → SBFL > MBFL
Mutations on buggy statements → MBFL ≥ SBFL

Family	Mutable bugs			
	Top-1	Top-3	Top-5	Top-10
MBFL	14	41	50	63
PS	5	9	12	12
SBFL	12	35	50	57
ST	0	4	5	19

Thesis statement

Understanding the capabilities and limitations of standard test generation and fault localization techniques on data science programs implemented in dynamic languages such as Python informs the development of new techniques that can be more effective.





Effectiveness

SBFL > MBFL ≫ PS ≃ ST

Family	% of bugs in Top-5
MBFL	27
PS	7
SBFL	43
ST	6

Extra slides

Experimental Subjects

	Projects	LOC	Total Functions	Tested Functions	Avg. Number of Annotations	Known Bugs
Subjects P	2	3917	249	105	1.33	-
Subjects R	19	14219	735	24	6.00	81

Keras TensorFlow PyTorch

Effectiveness

A ≫ B: "A is much more effective than B", if A@k% > B@k% for all ks, and A@k%− B@k% ≥ 10 for at least three ks out of four.

A > B: "A is more effective than B", if A@k% > B@k% for all ks, and A@k%− B@k% ≥ 5 for at least one k out of four.

A ≥ B: "A tends to be more effective than B", if A@k% ≥ B@k% for all ks, and A@k% > B@k% for at least three ks out of four.

A ≃ B: "A is about as effective as B", if none of A ≫ B, A > B, A ≥ B, B ≫ A, B > A, and B ≥ A holds.

Efficiency

$A \gg B$: "A is much more efficient than B", if $T(B) > 10 \cdot T(A)$.

$A > B$: "A is more efficient than B", if $T(B) > 1.1 \cdot T(A)$.

$A \simeq B$: "A is about as efficient as B", if none of $A \gg B$, $A > B$, $B \gg A$, and $B > A$ holds.

Effectiveness vs Granularity

Statement ≃ Function ≃ Module

Family	% of bugs in Top-5		
	Statement	Function	Module
MBFL	27	61	86
PS	7	13	21
SBFL	43	72	92
ST	6	27	36

Python vs Java - best on crashing bugs

Python: SBFL

Java: ST

Family	Crashing bugs			
	Top-1%	Top-3%	Top-5%	Top-10%
MBFL	7	21	27	34
PS	0	0	0	0
SBFL	14	31	43	53
ST	0	10	16	37