PyTraceBugs: A Large Python Code Dataset for Supervised Machine Learning in Software Defect Prediction

E. Akimova, A. Bersenev, A. Deikov, A. Konygin, K. Kobylkin, I. Mezentsev, V. Misilov

Krasovskii Institute of Mathematics and Mechanics,

Ural Federal University, Ekaterinburg, Russia

The 28th Asia-Pacific Software Engineering Conference, online, 6-9 December, 2021

IMPACT OF BUGS ON THE SOFTWARE DEVELOPMENT

Bugs in complex software programs have a variety of undesirable consequences, including:

- increasing cost of software development;
- data loss;
- program crashes;
- hardware failures,

which can cause significant money losses.

BUGS FORMS

Finding bugs is a longstanding problem in software engineering. One can distinguish between the following (possibly overlapping) forms:

BUGS FORMS

- defects (e.g. when a function implementation is too complex to understand and maintain, some of possible exceptional situations/program states is not handled, etc);
- bugs, which lead/may lead to unexpected programs behaviours (e.g. causing them to crash);
- anomalies, which are not essentially bugs, but represent some unusual coding practices, not following prescribed standards.

BUGS GRANULARITIES

Bugs can be at the following granularities:

BUGS GRANULARITY

- module;
- file;
- class;
- function/method (simply, snippet);
- range of lines.

SPECIFICS OF PYTHON LANGUAGE

Python is a language of choice for developers, working in a variety of domains, including web development, data science, and machine learning.

PYTHON SPECIFICS

- dynamic typing;
- pervasive object paradigm;
- its interpreter and existing static analysis tools do not provide any thorough checks of source code.

This postpones revealing of bugs to runtime stage and leads to the need of debugging programs, which might be costly.

SPECIFICS OF PYTHON BUGS

OUR AIM

Our focus is on identifying bugs in Python functions/methods source code implementations, which cause programs stop working.

```
a = [1, 2, 3, 4]

def getItem(index, array):
    return array[index]

getItem(5, a)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-1-43e12b2f0d17> in <module>
      4     return array[index]
      5
----> 6 getItem(5, a)

<ipython-input-1-43e12b2f0d17> in getItem(index, array)
      2
      3 def getItem(index, array):
----> 4     return array[index]
      5
      6 getItem(5, a)

IndexError: list index out of range
```

SPECIFICS OF PYTHON BUGS

- if a Python program fails to work, the interpreter usually throws an exception, accompanied with printing out an error exception report, containing the information about a stack of functions/methods calls, which cause the program stop;
- this report can be used for bug localization.

DEEP LEARNING FOR BUG PREDICTION IN PYTHON SOURCE CODE

- applying deep learning methodologies (say, Transformers) for natural language processing tasks leads to dramatic improvements;
- using those methodologies (e.g. graph networks) for finding bugs in Python source code also gives promising results (Allamanis et. al, 2021).

SPECIFICS OF DEEP LEARNING MODELS

1. have lots of parameters;
2. require large datasets for their training.

CODE ANALYSIS TASKS, UNDERLYING KNOWN DATASETS

Not every known dataset is suitable for training and evaluating deep learning models. Some of the datasets are intended for purposes other than fitting neural networks.

POSSIBLE PURPOSES

1. automatic test generation: tests must fail on buggy code pieces and work well on fixed pieces;
2. program repair: learn typical patterns of fixing source code and use them to fix unseen code;
3. bug prediction: report if a code piece contains bugs.

CONTENT OF THE KNOWN DATASETS

Tasks, underlying datasets, restrict them to contain a specific type of information. For example, datasets aimed to either automatic test generation or program repair usually contain the so called bugfixes.

```
def getItem(array,
            index):
    return array[index]
```

```
def getItem(array,
            index):
    if hasattr(array, '__sizeof__') and hasattr(array, '__getitem__'):
        if isinstance(index, int) and (index < len(array)):
            return array[index]
    return None
```

Each bugfix contains two consequtive versions of the same source code, where the first piece contains bugs whereas the second one is an immediate fix of those bugs.

SUMMARY OF THE KNOWN DATASETS

TABLE: Summary of the known datasets

Name	Language	Granularity	Size	Purpose	Content type
ManyBugs	C	module	185	test generation	bugfix pairs
Defects4J	Java	module	357	test generation	bugfix pairs
Bugs.jar	Java	module	1158	test generation	bugfix pairs
BugsInPy	Python	file	493	test generation	bugfix pairs
GHPR	Java	file	3026	bug prediction	bugfix pairs
BugHunter	Java	file/class/snippet	159k	bug prediction	bugfix pairs
CodRep	Java	snippet	800k 1 line bugs	program repair	bugfix pairs
ManySStuBs4J	Java	snippet	153652 1 line bugs	program repair	bugfix pairs
artif. coll. data[1]	Python	snippet	1M simple bugs	program repair	bugfix pairs

[1] M.Allamanis et al, Self-supervised bug detection and repair, 2021

SUMMARY OF THE PYTRACEBUGS DATASET

TABLE: Summary of the known datasets and our contribution

Name	Language	Granularity	Size	Purpose	Content type
ManyBugs	C	module	185	test gen.	bugfix pairs
Defects4J	Java	module	357	test gen.	bugfix pairs
Bugs.jar	Java	module	1158	test gen.	bugfix pairs
BugsInPy	Python	file	493	test gen.	bugfix pairs
GHPR	Java	file	3026	bug pred.	bugfix pairs
BugHunter	Java	file/class/snippet	159k	bug pred.	bugfix pairs
CodRep	Java	snippet	800k 1 line bugs	prog. repair	bugfix pairs
ManySStuBs4J	Java	snippet	153652 1 line examples	prog. repair	bugfix pairs
artif. coll. data[2]	Python	snippet	1M simple bugs	prog. repair	bugfix pairs
PyTraceBugs	**Python**	**snippet**	**24k buggy, 5.6M stable**	**bug pred.**	**buggy, stable code**

- in distinction to the known datasets, PyTraceBugs dataset is collected for the bug prediction task, considered in the form of the binary classification problem with two classes of snippets;
- 1st class contains buggy code, 2nd class - error-free (or stable) code.

[2] M.Allamanis et al, Self-supervised bug detection and repair, 2021

PURPOSE OF THE PYTRACEBUGS DATASET

AIM OF THE DATASET

- PyTraceBugs is a large dataset of Python source code, containing snippets labeled either buggy or correct;
- the dataset is aimed to pre-training and fine-tuning complex deep learning models for the bug prediction task.

BUGS IN THE PYTRACEBUGS DATASET

SPECIFICS OF BUGS FROM THE DATASET

Bugs from the dataset manifest themselves in the form of program breaks, accompanied with error exception reports, containing tracebacks of functions/methods calls, causing their corresponding Python programs stop.

TABLE: Most common error types in the dataset

Error type	Percentage of snippets (%)
AttributeError	16.6
TypeError	15.9
ValueError	10.
KeyError	8.2
RuntimeError	5.5
IndexError	5.3
Others	38.3

DESCRIPTION OF THE PYTRACEBUGS DATASET

STRUCTURE AND CONTENT OF THE DATASET

It is split into training, validation and test samples.

- training and validation samples contain automatically labeled source code from Github repositories;
- buggy code from those samples is taken from bugfix pairs of snippets, extracted from bugfix commits;
- error-free code is obtained from stable code of repositories;
- test sample of 330 snippets contains manually curated buggy code and selected stable code;
- training and test samples are taken from distinct repositores.

In distinction to the dataset of (Allamanis et al, 2021), both training and validation samples contain real bugs, i.e. not artificially generated

CONFIDENCE OF LABELING IN TRAINING AND VALIDATION SAMPLES

The following ideas are employed to provide high confidence of labeling of snippets into either buggy or correct.

APPROACHES TO ENFORCE CORRECTNESS OF LABELING

- source code is extracted from well-respected Github repositories, maintained by experienced developers;
- buggy snippets are extracted from repositories (bugfix) commits and pull requests, related to Github issues, having bug labels and containing error exception reports on their web pages;
- snippets of correct code are extracted from highly stable code of repositories;

CONFIDENCE OF LABELING IN TRAINING AND VALIDATION SAMPLES

To measure amount of noise in the dataset, a percentage is estimated of buggy snippets, for which changes, introduced in their corresponding fixed versions from bugfix commits and pull requests, are confined to refactoring.

WAYS TO EVALUATE AMOUNT OF NOISE

- to obtain a lower bound on the percentage of the refactoring changes, the rate is estimated of the changess bound to docstrings and comments (2.6%);
- the rate of refactoring changes is estimated manually on a random sample of several hundreds of snippets (10–15%).

CONFIDENCE OF LABELING IN THE TEST SAMPLE

Confidence of labeling of snippets in the test sample is made almost 100%, applying manual validation by two Python experts.

PRINCIPLES, GUIDING THE MANUAL VALIDATION

- a bug reported on the web page of the corresponding issue is simple to understand;
- the reported bug is not dependency, compatibility, or the regression bug;
- a fix of the bug introduced into a buggy snippet is also simple;
- correct snippets are chosen from stable snippets the restriction that the snippet should be called many times from other snippets.

TRAINING PREDICTIVE MODELS ON THE DATASET

An alternative way to demonstrate quality of the dataset consists in building predictive models using its data.

DETAILS OF TRAINING PREDICTIVE MODEL

- multi-language pretrained CodeBERT model is applied to compute embeddings of source code from the dataset;
- LightGBM classifier is trained on the computed embeddings.

TABLE: Results of the prediction experiments on the test sample

	Precision	Recall	F_1-measure
correct	0.61	0.99	0.76
buggy	0.96	0.34	0.5

CONCLUSION

BASIC TAKEAWAYS

- a large labeled dataset is proposed for both training and evaluating of deep learning models for software bug prediction;
- it contains a number of examples of real bugs in the Python source code at the granularity of snippets, *i.e.*, implementations of functions or methods;
- confidence in labeling of the snippets in both training and validation samples is about 85% according to our estimates, whereas it is almost 100% for the test sample;
- a model is built on the dataset: it predicts bugs with precision of 0.96 and recall of 0.34 on the manually validated test sample;
- the dataset is available at `https://github.com/acheshkov/pytracebugs`.

THANK YOU FOR YOUR ATTENTION!