Software Lab
Institute of Software Engineering
University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart

Master Thesis

DyPyBench: A Benchmark of Executable Python Software

Piyush Krishan Bajaj

Course of study:	INFOTECH
Examiner:	Prof. Dr. Michael Pradel
Supervisor:	Islem Bouzenia
Started:	November 7, 2022
Completed:	May 7, 2023

University of Stuttgart
Germany

SOLA
SoftwareLab

Abstract

Short summary of thesis.

Zusammenfassung

Kurzfassung der Arbeit.

Contents

1 Introduction

Python is a high-level, interpreted, general-purpose programming language that has become one of the most widely used languages in the world. It has a large and active community of users and contributors that support its development and growth. One of the reasons for its popularity is its readability and concise syntax, making it a great choice for both beginners and experienced programmers. In addition to its readability, Python supports a variety of programming paradigms including procedural, object-oriented, and functional programming, giving it the versatility to be used for a range of tasks. The language uses dynamic typing, meaning the type of a variable is determined during runtime, and employs a combination of reference counting and cycle-detecting garbage collection for effective memory management. Python also has a dynamic name resolution, or late binding, which enables the binding of method and variable names during program execution. Rather than having all its functionality built into its core, Python was created to be highly extensible through the use of modules. This compact modularity has made it a popular choice for adding programmable interfaces to existing applications [4].

DynaPyt is a tool for dynamic analysis of Python code. It aims to provide insights into the behavior of Python programs, such as performance and memory usage, by analyzing the runtime behavior of code. DynaPyt provides several features, such as the ability to collect data on function calls, memory allocation, and object creation, and to visualize the results in a user-friendly interface. The tool is designed to be easy to use and to work with existing Python code, making it accessible to a wide range of users, from researchers to developers. DynaPyt is based on PyPy, a fast and compliant Python interpreter, and is implemented using PyPy's tracing and profiling capabilities. The tool provides several different types of analysis, including time profiling, memory profiling, and function call tracing, making it possible to get a detailed understanding of the behavior of Python programs. This information can be used to identify performance bottlenecks and optimize the program for better performance. With its ease of use, flexibility, and range of features, DynaPyt is a valuable tool for anyone working with Python code. [9]

The past decade has seen tremendous progress in the field of artificial intelligence thanks to the resurgence of neural networks through deep learning. This has helped improve the ability for computers to see, hear, and understand the world around them, leading to dramatic advances in the application of AI to many fields of science and other areas of human endeavor [1]. Machine learning, which is a branch of artificial intelligence (AI) and computer science focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. Over the last couple of decades, the technological advances in storage and processing power have enabled some innovative products based on machine learning, such as Netflix's recommendation engine and self-driving cars [3].

Tasks such as bug detection [13], code completion [11], quality analysis [5, 7], code refactoring [6], and testing [10, 12, 16] can be performed using machine learning algorithms by analysing software source code and improving the development process. Such program analysis has the potential to improve the productivity, efficiency and quality of code. Machine learning works by training algorithms on data to make predictions or take actions based on that data. There are three main steps in the machine learning process namely, Data collection and preparation, model training, and finally Model evaluation and deployment. Data collection is a critical step as the quality and quantity of the data used to train a model can greatly impact its accuracy and effectiveness. When collecting data for machine learning in software development, it's important to focus on the right kind of data to train the algorithm on. This data should be relevant, diverse, and representative of the problem being solved. For example, if a machine learning model is being trained to detect bugs in code, the data should consist of code snippets with bugs as well as code snippets without bugs.

The program analysis algorithms of machine learning, either use the code snippets which are available via the source code[14] or logs which are generated by the execution of the software [8]. In both of these cases we do not get the detailed information related to the run time behaviour of the executed software. Run time behaviour can provide us a different perspective and has the potential to provide deeper insights which can help us in improving the code and the development process. For example, ***

As described above, DynaPyt is a framework which provides us the insights into the run time behaviour of python programs. Since we are already seeing the usefulness of machine learning in the software engineering tasks, we can combine the best of both of these to achieve a program analysis tool which uses machine learning to improve development process of code using the run time behaviour of python programs. At the time of writing this thesis, there are no framework or benchmark tools available which combine them. With this thesis we provide a framework, which is a benchmark of executable python software which can be used to generate data set for machine learning tasks in software engineering tasks such as code generation, test case generation etc. for researchers and developers.

The benchmark consists of 50 executable python software from diverse application domains which Python covers being a highly popular general purpose programming language. By using the DynaPyt framework, the data set generated is able to encapsulate the run time behaviour of python programs. In DyPyBench framework, we provide three different program analysis tasks of software engineering using machine learning approach. The first task *** . The second task ***. Finally, the third task ***. In this thesis, we have further used the machine learning algorithm ** to test and evaluate our framework. The results are ***.

2 Background and Motivation

2.1 Python and its dynamic properties

2.2 Dynamic Analysis Frameworks

2.2.1 DynaPyt

2.3 Dynamic Benchmarking and its usage for Machine Learning

2.4 Python Benchmarks

3 Methodology

In this section, we describe the overall approach we adopt in this thesis to create a benchmark of executable python software and using this benchmark to generate a data set for training a machine learning model for program analysis and further use this trained model to perform some of the machine learning tasks in program analysis.

The overall workflow of our approach is shown in the Figure 3.1. In the section 3.1, we describe the details on how we select specific projects from the large corpus of python projects. We describe the installation and setup of the selected projects in the section 3.2. In section 3.3 we describe the command line interface and what can be done using it. In section 3.4 we describe the various dynamic analysis that we can perform to generate data set for program analysis. We describe the machine learning model and its training data in the section 3.5. Finally, in section 3.6 we describe how we can use this framework and the trained machine learning model for program analysis.

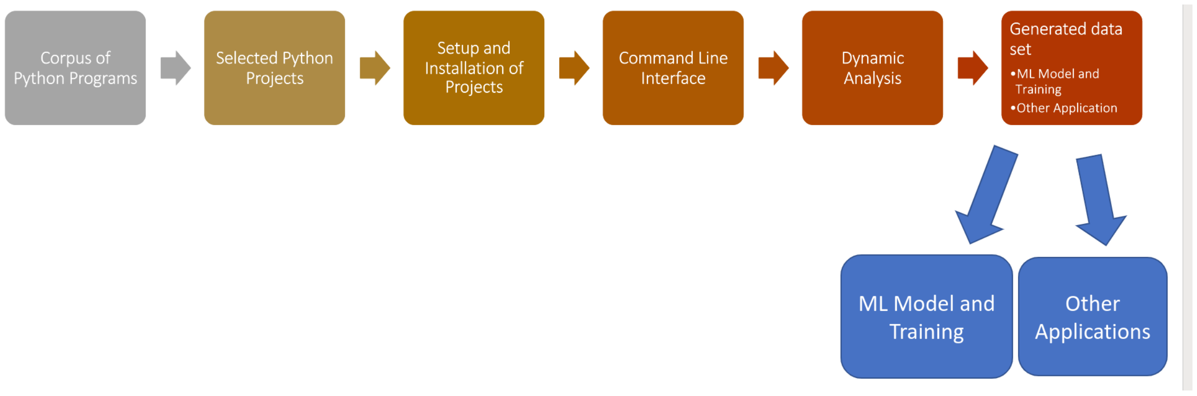

Figure 3.1: Overall Approach of DyPyBench.

3.1 Selected Python Projects

Since Python is a very popular and general purpose programming language, it has been used in many domains and we have a large set of open source python projects available on the internet. In this thesis we target projects from many of these varied domains which are well accepted in the community and at the same time open sourced. Another important factor is the availability of test suites in the project source code.

3.2 Setup and Install

With a set of selected projects based on the required criteria, we then proceed to setup and install these projects with each one having its own virtual environment to install all the dependencies with their specific version from the requirements file. And then also using the source to install the project and its dependencies with pip package manager. Each of these projects are numbered and are placed inside their own folders. The projects source is cloned to a particular date. We also install some of the other python packages using pip in order to run the test suites successfully. We end the setup with installing the packages for testing and analysis tools present in the benchmark.

3.3 Command Line Interface

With the selected projects from section 3.1 installed and setup, we provide a command line interface for the user to run tests and dynamic analysis of a single project or a collection of projects. This command line interface is a single command with various available options. The various options available are to list all the installed projects with their urls, run the test suite of the project, perform instrumentation of the source code for dynamic analysis using dynapyt or lexecutor. execution of lexecurtor or dynapyt analysis. It also provides the option to update dynapyt and lexecutor. WE can also store the output to a particular file or spefic a timeout for the specific tasks where applicable.

3.4 Dynamic Analysis

While the command line interface provides us with varied options, one such option is the dynamic analysis. There are two available analysis, dynapyt and lexeecutor. We can use the command line to perform the specific analysis and generate the logs which we can use to train machine learning models for program analysis or use these logs on our own to understand the behaviour of the selected projects. These analysis can also help us in discovering some issues in the selected python projects. An example is dynapyt which explored a bug in pytorch.

3.5 ML Model

3.6 Application

4 Implementation

In this section, we describe the various tools and libraries used to implement our approach. WE provide the entire benchmark inside a docker container, which contains all the required projects and dynamic analysis frameworks. We have used bash scripting to automate the repetitive tasks of setting up the projects and installing the required dependencies. Furthermore, we have also used python scripting to provide a single command line interface to the user with varied options to perform the various steps as needed. We have used the DynaPyt [9] and the LExecutor[15] projects in order to perform the analysis on the projects and generate logs as required for the machine learning model. The implementation details of the steps in the approach are explained in the following subsections.

4.1 Selected Python Projects

The programming language, Python, has a large and active community. This community has created a vast collection of libraries, tools, and frameworks that make it easy to perform a wide range of tasks. For example, there are libraries for machine learning, data analysis, web development, and more. These libraries and tools make it easy to accomplish complex tasks with just a few lines of code, making Python a powerful and productive language to use. One such community project is Awesome Python [2], which provides a curated list of awesome Python framework, libraries, software and resources. These frameworks, libraries and software are open source projects, which are available with their source code. The Awesome Python project also provides us with a category and sub-category of these projects based on the common domains of usage of these projects. Since, we aim to cover projects from a vast range of domains in our benchmark, we use Awesome Python and its categories as a guiding factor.

Listing provides the details of the projects and categories as per the awesome python website

We select the projects from this curated list as mentioned above while adding some more selection criteria to make the projects compatible with our benchmark. These additional selection criteria and why we use them is listed below: 1, Stars on github: Ensure that the community has accepted this project 2, Available test suites: To be able to run dynamic analysis out of box 3. Ability to run tests using pytest: We and also the dynamic analysis framework use pytest for execution of test suites

We select 50 python projects, which satify the above criteria from the awesome python project and store the github url, requirement file location (if any) and test files location in a text file name github-urls.txt. This file also specifies the flags for requirements and test. Combining the different categories provided by awesome python and the additional selelction criteria, we select 50 python projects which

4.2 Setup and Installation

4.3 Command Line Interface

4.4 Dynamic Analysis

5 Evaluation

6 Related Work

7 Conclusion

Bibliography

[1] A golden decade of deep learning: Computing systems applications.

[2] vinta/awesome-python: A curated list of awesome python frameworks, libraries, software and resources.

[3] Machine learning, Dec 2022.

[4] Feb 2023. Page Version ID: 1136880732.

[5] H. Alsolai and M. Roper. A systematic literature review of machine learning techniques for software maintainability prediction. *Information and Software Technology*, 119:106214, Mar 2020.

[6] M. Aniche, E. Maziero, R. Durelli, and V. H. S. Durelli. The effectiveness of supervised machine learning algorithms in predicting software refactoring. *IEEE Transactions on Software Engineering*, 48(4):1432–1450, Apr 2022.

[7] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology*, 108:115–138, Apr 2019.

[8] B. Debnath, M. Solaimani, M. A. G. Gulzar, N. Arora, C. Lumezanu, J. Xu, B. Zong, H. Zhang, G. Jiang, and L. Khan. Loglens: A real-time log analysis system. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, page 1052–1062, Jul 2018.

[9] A. Eghbali and M. Pradel. Dynapyt: A dynamic analysis framework for python. page 12, 2022.

[10] R. Lima, A. M. R. da Cruz, and J. Ribeiro. Artificial intelligence applied to software testing: A literature review. In *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, page 1–6, Jun 2020.

[11] F. Liu, G. Li, Y. Zhao, and Z. Jin. Multi-task learning based pre-trained language model for code completion. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, page 473–485, New York, NY, USA, Jan 2021. Association for Computing Machinery.

[12] S. Omri and C. Sinz. Deep learning for software defect prediction: A survey. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 209–214, New York, NY, USA, Sep 2020. Association for Computing Machinery.

[13] M. Pradel and K. Sen. Deepbugs: A learning approach to name-based bug detection. (arXiv:1805.11683), Apr 2018. arXiv:1805.11683 [cs].

[14] T. Sharma, M. Kechagia, S. Georgiou, R. Tiwari, I. Vats, H. Moazen, and F. Sarro. A survey on machine learning techniques for source code analysis. (arXiv:2110.09610), Sep 2022. arXiv:2110.09610 [cs].

[15] B. Souza and M. Pradel. Lexecutor: Learning-guided execution. (arXiv:2302.02343), Feb 2023. arXiv:2302.02343 [cs].

[16] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36, Jan 2022.

Selbsständigkeitserklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Datum	**Unterschrift**