ManyTypes4Py

A benchmark python dataset for machine learning-based type inference

Mir, Amir M.; Latoskinas, Evaldas; Gousios, Georgios

DOI
10.1109/MSR52588.2021.00079

Publication date
2021

Document Version
Accepted author manuscript

Published in
2021 IEEE/ACM 18th International Conference on Mining Software Repositories, MSR 2021

ManyTypes4Py: A Benchmark Python Dataset for Machine Learning-based Type Inference

Amir M. Mir
Department of Software Technology
Delft University of Technology
Delft, The Netherlands
s.a.m.mir@tudelft.nl

Evaldas Latoškinas
Department of Software Technology
Delft University of Technology
Delft, The Netherlands
e.latoskinas@student.tudelft.nl

Georgios Gousios
Department of Software Technology
Delft University of Technology
Delft, The Netherlands
g.gousios@tudelft.nl

***Abstract*—In this paper, we present ManyTypes4Py, a large Python dataset for machine learning (ML)-based type inference. The dataset contains a total of 5,382 Python projects with more than 869K type annotations. Duplicate source code files were removed to eliminate the negative effect of the duplication bias. To facilitate training and evaluation of ML models, the dataset was split into training, validation and test sets by files. To extract type information from abstract syntax trees (ASTs), a light-weight static analyzer pipeline is developed and accompanied with the dataset. Using this pipeline, the collected Python projects were analyzed and the results of the AST analysis were stored in JSON-formatted files. The ManyTypes4Py dataset is shared on zenodo and its tools are publicly available on GitHub.**

***Index Terms*—Type Inference, Machine Learning, Python, Type Annotations, Static Analysis**

I. Introduction

In recent years, dynamic programming languages (DPLs) have become immensely popular as they give developers fast prototyping [1]. However, DPLs lack static typing, which causes several issues such as unexpected run-time exceptions, sub-optimal support for integrated development environments (IDEs), and less precise program analysis. To address these issues, optional static typing is introduced for DPLs like Python [2], JavaScript [3], and PHP [4]. Yet, developers are required to manually add type annotations to their existing codebases, which is a laborious task [5]. To ease the type annotation burden, researchers have recently employed machine learning (ML) techniques to infer types for DPLs [6]–[8].

ML techniques need a sufficiently large dataset to achieve an acceptable level of generalization for the task at hand [9]. Concerning the ML-based type inference for DPLs, it is difficult to create a benchmark dataset that contains software projects with a sufficient number of type annotations. Because of the optional static typing, many software projects written in DPLs lack type annotations. Nevertheless, to train an ML-based type inference model for Python, researchers created their own dataset by either gathering a small set of projects with type annotations [7] or employ static type inference tools to add type annotations to existing projects [8].

We believe that there is a need for a large benchmark dataset that facilitates training ML-based type inference models, especially for Python. Unlike TypeScript's compiler, the Python interpreter cannot infer the type of variables or function signatures at compile time [10]. Motivated by this, we present the ManyTypes4Py dataset, a large dataset to train ML models for predicting type annotations in Python. Currently, we are working on the Type4Py model [11], which is trained on the earlier version of the ManyTypes4Py dataset. The experimental results show that the model trained on our dataset is overall more accurate when compared to the same model trained on a smaller dataset [11].

In summary, the paper has the following contributions:

- **ManyTypes4Py dataset**, which features 5,382 Python projects with more than 869K type annotations. The latest version of the dataset can be downloaded on zenodo[1].
- **LibSA4Py tool**, a light-weight static analyzer pipeline to process Python projects and extract type hints/features for training ML-based type inference models. The tool is publicly available on a GitHub repository[2].

II. Method

We created the ManyType4Py dataset using the following methodology:

- To find Python projects with type annotations, we intuitively search for projects that have mypy as a dependency on libraries.io. Since mypy is the most used type checker for Python, projects that use mypy have most likely type annotations. Our search resulted in 5,382 Python projects that are publicly available on GitHub. We cloned all the discovered projects in Sep. 2020 and created a file that contains projects' URL and their latest commit hash.
- As demonstrated by Allamanis [12], it is essential to de-duplicate a code corpora before training ML models, as code duplication negatively affects the performance of ML models when testing on duplicated code corpora. Following this, we de-duplicated the collected Python corpora using our code de-duplication tool, namely, CD4Py[3]. In short, the CD4Py tool tokenizes Python source code files, vectorizes files using Term Frequency-Inverse Document Frequency (TF-IDF), and performs k-

[1]https://zenodo.org/record/4479714
[2]https://github.com/saltudelft/libsa4py
[3]https://github.com/saltudelft/CD4Py

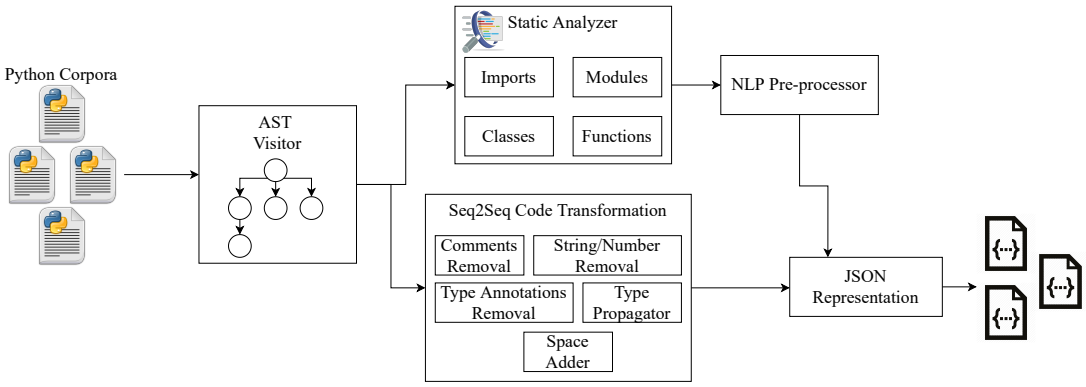

Fig. 1. Overview of light-weight static analysis pipeline (LibSA4Py tool)

TABLE I
DUPLICATION STATISTICS ACROSS THE MANYTYPES4PY DATASET

Duplication stats	Value
# Detected duplicate files	400,245 (78.43%)
# Detected clusters	45,836
Avg. # of files per clusters	8.73
Median # of files per clones	3.00
Duplication ratio	69.45%

nearest neighbor search to identify candidate duplicates files.

- After removing duplicate files, we split the dataset into three sets by files, i.e., 70% training data, 10% validation data, and 20% test data. This is a common practice that is considered in recent research work [7], [8], concerning machine learning-based models for type inference.
- Given the de-duplicated code corpora and a list of files for the three sets, we ran light-weight static analysis pipeline, which is depicted in Figure 1. First, the Abstract Syntax Tree (AST) of Python source files are visited. Second, type hints and features are statically extracted from imports, modules, classes, and functions, inspired by recent ML-based type inference approaches [6], [7]. Third, the seq2seq representation[4] of source code files [13] are generated by removing comments, string, number literals, and propagating types. Forth, common Natural Language Processing (NLP) practices such as tokenization and lemmatization are applied to identifier names in source code files. Finally, the processed Python projects are stored as a JSON-formatted file.

After completing all the aforementioned steps, a zip file is created which contains: (1) JSON file of processed Python projects (2) a file containing projects' URL and their latest commit hash (3) a file containing duplicate files in the dataset (4) a CSV file containing a list of files and their corresponding set. The helper scripts and instructions for preparing the dataset are publicly available on a GitHub repository[5].

[4]Each token is aligned with a type if present. Otherwise, zero is inserted.
[5]https://github.com/saltudelft/many-types-4-py-dataset

III. DESCRIPTION

Before describing the characteristics of the ManyTypes4Py dataset, we first describe the duplication statistics across the dataset, which is shown in Table I. The duplication ratio of the dataset is 69.45% as detected by the CD4Py tool. This is in line with the findings of Lopes et al. [14], which showed that the Python ecosystem on GitHub has 71% file-level duplicates. It should be noted that the duplication ratio is obtained using the following formula:

$$\frac{(\text{no. of duplicate files} - \text{ no. of detected clusters})}{\text{no. of source code files} * 100.0} \quad (1)$$

After keeping a file from each duplicate cluster, we removed 354,409 duplicate files from the dataset.

The characteristics of the ManyTypes4Py are shown in Table II after code de-duplication. Overall, the dataset has 5,382 Python projects and 183,916 source code files (i.e. `.py` files). 27.6% of source code files have type annotations, i.e., there is at least one type-annotated function in those files. Of 2,096,797 functions in the dataset, 53.8% has comments[6,7] and 15.5% has return type annotations. However, Of 3,923,667 functions' arguments, 5.6% have comments and 12.2% have type annotations.

As shown in Table II, there are a total of 869,825 type annotations and 67,060 unique types in the ManyTypes4Py dataset. To demonstrate the distribution of types, top 10 most frequent types in the dataset are shown in Figure 2. Of 869,825 types, 50.56% of them are present in the top 10 most frequent types. As can be observed from Figure 2, types follow a long-tail distribution. In other words, the majority of type annotations are either `str`, `None`, `int`, or `bool`.

As stated in Section II, the dataset provides processed Python projects in JSON-formatted files, which contains various type hints and features. As of this writing, there are 23 fields in JSON-formatted files that are described in Table III. Of 23 extracted fields, 16 of them are natural/contextual type hints or features that can be used for training ML-based

[6]Note that here comments are functions' docstring in Python, which can be a one-line description or a complete description of a function.
[7]Our LibSA4Py tool can detect Google, reST, and NumPy docstrings.

TABLE II
CHARACTERISTICS OF THE MANYTYPES4PY DATASET

Metrics	Dataset			
	All	Training	Validation	Test
Repositories[a]	5,382	4,913	2,789	3,796
Lines of code[b]	22M	-	-	-
Files	183,916	132,409	14,675	36,832
...with type annotations	50,838 (27.6%)	36,542	4,105	10,191
Functions	2,096,797	1,509,048	169,519	418,230
...with comment	1,129,573 (53.8%)	812,632	91,325	225,616
...with return type annotations	325,532 (15.5%)	234,319	26,104	65,109
Arguments	3,923,667	2,822,699	310,685	790,283
...with comment	220,976 (5.6%)	159,453	16,924	44,599
...with type annotations	480,793 (12.2%)	347,898	37,148	95,747
Types	869,825	347,898	89,334	192,102
...unique	67,060	53,614	13,995	23,572

[a] Note that there is an intersection among repositories in the three sets as the dataset is split by files.
[b] Comments and blank lines are ignored when counting lines of code.

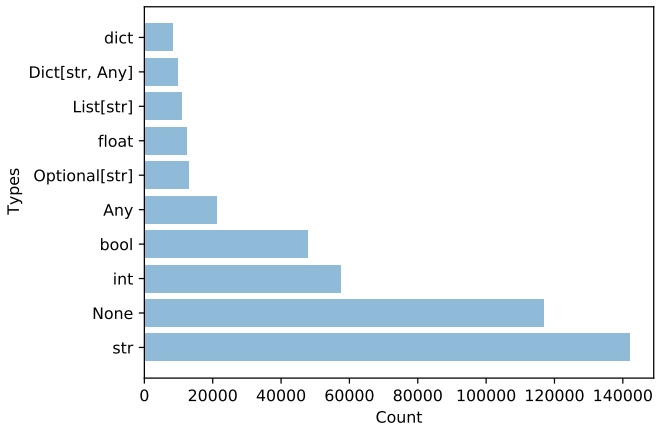

Fig. 2. Top 10 most frequent types in the ManyTypes4Py dataset

type inference models. For instance, the `name` field stores the name of a class or a function, which is a natural source of information for predicting types [6]. The `ret_exprs` and `params_occur` fields provide return expression(s) of a function and usages of functions' parameter(s) in its body, respectively. These are considered contextual type hints, i.e., the context in which a variable or an argument is used provides a hint for predicting types [7]. Also, the `untyped_seq` and `typed_seq` fields provide the normalized seq2seq representation of a Python source code file and the type of identifiers in the file, respectively. They both can directly be used for training an ML-based type inference model.

IV. APPLICATIONS

a) ***ML-based type inference****:* In this task, ML models are trained to predict the type of functions' arguments, return types, and variables for DPLs (e.g. Python). To do so, the AST of source code files are analyzed to extract features that give a hint for predicting types. By processing ASTs, the ManyTypes4Py dataset provides common features, i.e., natural and contextual type hints that can be employed to create code embeddings and train an ML model. Moreover, the provided seq2seq representation of source code files gives the full context around identifiers.

b) ***Learning-based code completion****:* In this application, an ML model is expected to predict part of a word or token for a function or a variable. For DPLs, code completion is a challenging task as there is no type information available. To overcome this, ASTs are statically analyzed while providing type information [15]. The ManyTypes4Py dataset can be used as a baseline for training a code completion model as it provides partial type annotations for functions and variables. Also, our AST analysis pipeline (LibSA4Py tool) can further be extended to infer types of nodes and variables for simple cases.

V. LIMITATIONS

Currently, our static analysis pipeline cannot parse source code files in Python 2. Therefore, Python2-style type annotations[8] cannot be extracted. Due to this limitation, about 1% of source code files in the dataset cannot be parsed.

VI. RELATED WORK

There are several Python code corpora that can be used for machine learning-based type inference. Recently, Allamanis et al. [8] proposed the Typilus model, which is a graph-based neural model that predicts type annotations for Python. The Typilus model [8] is accompanied by a dataset that contains 600 Python projects. Moreover, the source code files of Typilus' dataset are converted to graph representations that are only suitable for training the Typilus model. The ManyTypes4Py dataset provides JSON-formatted analyzed source

[8]https://www.python.org/dev/peps/pep-0484/#suggested-syntax-for-python-2-7-and-straddling-code

TABLE III
DESCRIPTION OF FIELDS IN THE JSON FILE OF PROJECTS PRODUCED BY THE LIBSA4PY PIPELINE

Field Name in the JSON	Description
	Project
`author/repo`	The name of a project and its author on the GitHub URL
`src_files`	Contains the path of a project's source code files
`file_path`	The path of a source code file to differentiate it with other files
	Module
`untyped_seq`	The normalized seq2seq representation of an analyzed source code file
`typed_seq`	Contains the type of identifiers in `untyped_seq` if present. Otherwise 0 is inserted.
`imports`	Contains the name of imports in an analyzed source code file
`variables`	Contains variables' name and their type defined in a module (i.e. global variables)
`classes`	Contains the JSON object of analyzed classes in a module which is described below
`funcs`	Contains the JSON object of analyzed functions in a module, which are described below
`set`	The set to which a source code file belongs to, i.e., `train`, `valid`, `test`
	Class
`name`	The name of an analyzed class in a module
`variables`	Contains class variables' name and their type if present
`funcs`	Contains the JSON object of analyzed functions in a class, which are described below
	Function
`name`	The name of an analyzed function in either a class or a module
`params`	Contains an analyzed function's parameter names and their type if present
`ret_exprs`	Contains the return expression(s) of an analyzed function
`ret_type`	The return type of an analyzed function if present
`variables`	Contains local variables' name and their type in an analyzed function
`params_occur`	Contains parameters and their usages in the body of an analyzed function
`docstring`	Contains docsting of an analyzed function if present, which has the below subfields
`docstring.func`	One-line description of an analyzed function if present
`docstring.ret`	Description of what an analyzed function returns if present
`docstring.long_descr`	Long description of an analyzed function if present

code files that contains useful type hints for training various machine learning models. Raychev et al. [16] published the Python-150K dataset in 2016, which contains 8,422 Python projects. Unlike our dataset, the Python-150K dataset [16] is not collected solely for the ML-based type inference task, meaning that a large number of projects in the dataset may not have type annotations at all, especially given the time that the dataset was created. Allamanis [12] showed that the Python-150K dataset suffers from code duplication despite the removal of project forks.

VII. CONCLUSION

In this paper, we present the ManyTypes4Py dataset, a benchmark Python dataset for ML-based type inference. It contains 5,382 Python projects from GitHub with more than 869K type annotations. The collected Python projects were de-duplicated by removing duplicate source code files to ensure that trained ML models do not have duplication bias. Using the accompanying LibSA4Py tool, the AST of Python source code files were analyzed to provide 16 type hints plus a seq2seq representation for training ML-based type inference models. For each analyzed project, the result of the AST analysis is saved in a JSON-formatted file. Although the dataset's main application is ML-based type inference, it can be a useful baseline for learning-based code completion.

In the near future, we will extend our static analysis pipeline (LibSA4Py tool) to add more type annotations to the dataset by implementing cheap and simple type inference heuristics. Also, we will perform data and control flow analysis to create graph representation of source code files for training graph-based neural models. To include more projects with type annotations, we will consider projects that use other type checkers other than mypy.

ACKNOWLEDGMENT

This research work was funded by H2020 grant 825328 (FASTEN).

REFERENCES

[1] [n. d.], "Ieee spectrum's the top programming languages 2019," https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019.

[2] G. Van Rossum, J. Lehtosalo, and L. Langa, "Pep 484–type hints," *Index of Python Enhancement Proposals*, 2014.

[3] G. Bierman, M. Abadi, and M. Torgersen, "Understanding typescript," in *European Conference on Object-Oriented Programming*. Springer, 2014, pp. 257–281.

[4] S. Klingström and P. Olsson, "Type inference in php using deep learning," *LU-CS-EX*, 2020.

[5] J.-P. Ore, S. Elbaum, C. Detweiler, and L. Karkazis, "Assessing the type annotation burden," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 190–201.

[6] R. S. Malik, J. Patra, and M. Pradel, "Nl2type: inferring javascript function types from natural language information," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 304–315.

[7] M. Pradel, G. Gousios, J. Liu, and S. Chandra, "Typewriter: Neural type prediction with search-based validation," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 209–220.

[8] M. Allamanis, E. T. Barr, S. Ducousso, and Z. Gao, "Typilus: neural type hints," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 91–105.

[9] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 843–852.

[10] M. L. Scott, *Programming Language Pragmatics*, 4th ed. Morgan Kaufmann, 2016.

[11] A. M. Mir, E. Latoskinas, S. Proksch, and G. Gousios, "Type4py: Deep similarity learning-based type inference for python," *arXiv preprint arXiv:2101.04470*.

[12] M. Allamanis, "The adverse effects of code duplication in machine learning models of code," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2019, pp. 143–153.

[13] V. J. Hellendoorn, C. Bird, E. T. Barr, and M. Allamanis, "Deep learning type inference," in *Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2018, pp. 152–162.

[14] C. V. Lopes, P. Maj, P. Martins, V. Saini, D. Yang, J. Zitny, H. Sajnani, and J. Vitek, "Déjàvu: a map of code duplicates on github," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–28, 2017.

[15] A. Svyatkovskiy, Y. Zhao, S. Fu, and N. Sundaresan, "Pythia: Ai-assisted code completion system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2727–2735.

[16] V. Raychev, P. Bielik, and M. Vechev, "Probabilistic model for code with decision trees," *ACM SIGPLAN Notices*, vol. 51, no. 10, pp. 731–747, 2016.