Type validation of Type4Py using Mypy

Merlijn Mac Gillavry
Supervisor(s): Amir M. Mir, Sebastian Proksch
EEMCS, Delft University of Technology, The Netherlands

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
22-6-2022

Abstract

Researchers at the Delft University of Technology have developed Type4Py: a tool that uses Machine Learning to predict types for Python code. These predictions can be applied by developers to their python code to increase readability and can later be tested by a type-checker for possible type-errors. If a prediction does not return a type-error that prediction is called type-correct. Type4Py has been evaluated by matching its predictions with earlier annotations, also called ground-truth, and has gotten an MRR of 71.7%. However, Type4Py's predictions have not been evaluated on their type-correctness. Therefore, I sought out to answer the following research question: How well does Type4Py perform when validated by the static type-checker Mypy? I answered this research question by answering two sub-questions: How many of Type4Py's predictions are type-correct? And how many of Type4Py's predictions are type-correct and match ground-truth? I tested a cleaned subset of the ManyTypes4Py dataset with Mypy by running a greedy strategy where I would always pick Type4Py's prediction with the highest confidence on three different confidence thresholds: 0.25, 0.5 and 0.75 and reached accuracies in terms of type-correctness of 88%, 91% and 95% for those, respectively. For the case where Type4Py's predictions matched ground-truth, the predictions on those same thresholds reached accuracies in terms of type-correctness of 95%, 97% and 98%. Comparing this with a similar Type predictor namely, Typilus . Type4Py's predictions are more type-correct with a confidence level of at most 50%.

1 Introduction

Programming Languages can be divided into two main categories. Namely, Dynamically-typed Programming Languages (DPL's) and Statically-typed Programming Languages (SPL's). SPL's check the code's typing for mistakes at compile time and return errors to the developer(s) before running the program, which prevents these errors from occurring while running the program and makes them less error prone [1]. They do, however, require more overhead in the form of added syntax that defines types of arguments/ variables and functions. DPL's can be faster for developing because of less overhead and are useful in prototyping by not being rigid in terms of typing. DPL's are, however, more prone to having typing-related bugs at runtime, which in the long-run can have negative effects on (developing) a program.

Researchers at the Delft University of Technology have developed Type4Py: a tool that uses Machine Learning to predict types for Python code [2]. Type4Py's type prediction could help in writing better readable, less error-prone and better maintainable Python code. For evaluation Type4Py used mean reciprocal rank (MRR) to reflect the quality perceived by users for its type predictions and achieves an impressive MRR of 71.1% [2].

While a MRR of 71.1% is a good evaluation method for the usefulness for development, it does not constitute a proof of Type4Py giving type-correct predictions. Therefore, in this paper I have tried to answer the following research question:

How well does Type4Py perform when validated by the static type-checker Mypy?

This question can then be divided into two sub-questions:

1. How many of Type4Py's predictions are type-correct (giving no errors when checked by Mypy)? (RQ1)
2. How many of Type4Py's predictions match ground-truth (matching earlier type annotations of developers) and are type-correct? (RQ2)

By answering this research question and its sub-questions, new insights can be drawn about not only the validity of Type4Py but also the future of tools like Type4Py in programming and software engineering.

To answer these questions, the following methodology was used. I used Type4Py's pre-trained model to infer type predictions on the ManyTypes4Py dataset. ManyTypes4Py dataset is a massive python dataset containing source files used for machine learning [3]. Then I compared these predictions with earlier annotations and checked these predictions with Mypy. This methodological approach has given me a reproducible answer to the research questions. By having done this in a reproducible manner, the experiments and methodology in this paper can be used to also validate future/improved versions of Type4Py.

The next section places this work in the context of existing research. After that, the methodology section gives a more detailed description of the problem and the method that was applied. Then, a Responsible Research section reflects on the ethical aspects of the research and discusses reproducibility. Evaluation Setup explains how I implemented the methodology in practice by making a pipeline. Followed by that, is an Evaluation and Results section that explains the results that were acquired after running the experiments. After that, a Discussion section reflects on potential faults with the methodology. Finally, in the section Conclusion and Future work, the research questions are answered and recommendations for further research are made.

2 Related works

In this section MyType4Py is placed in the context and background of existing research and contributions by the scientific community. This section centers specifically around Python's relation with types but it's important to recognize that, in addition to these related works, there has been a done a lot of research on other DPL's such as Javascipt.

Python and types: In late 2014 PEP 484 was introduced. PEP 484 (Type hints) was the proposal of introducing optional type-annotations to python [4]. This proposal was accepted and a year later python version 3.5 was released which included optional type annotations and Mypy as an optional static type-checker. Multiple algorithms for type inference have been proposed such as Typette, A static Inference for Python and Starkiller [5] [6] [7]. Static type inference methods are, because of Python's dynamic nature, often unfeasible in practice. [8]

Machine learning for type inference: Another approach for type inference in Python is with machine learning instead of static type inference. For example, Typewriter which infers type annotations for python by using a deep neural network model that provides a rank list of k prediction for missing type annotations in python source code [9]. A problem with this approach is that it uses only a fixed number of types which puts a huge constraint on the amount of types the model can accurately predict. Moreover, user-defined and rare types become very hard (sometimes impossible) to predict with this approach. To mitigate this hinderance Typilus was introduced which uses a graph neural network (GNN)-based model for prediction and can discriminate between similar and not similar symbols in terms of typing [10]. Continuing in this approach researchers at Delft University of Technology introduced Type4Py: A deep similarity learning-based hierarchical neural network model that uses two recurrent neural networks to discriminate between similar and different types in a high-dimensional space [2]. Unlike previous work Type4Py is trained on a type-checked dataset namely, ManyTypes4Py [3] and was evaluated with an MRR to show the useability perceived by developers.

Static type checking for evaluating Type4Py: User-perceived usability is, however, not a guarantee for type-correct code. A developer can think their annotation is type-correct where it is not. Therefore, in this research I have evaluated Type4Py with a static type-checker, namely Mypy.

3 Methodology

This section details the methodology used to answer the research questions. It starts with a listing of definitions and is followed by a description of the overall methodology. After that, specific details of the methodology used to answer each sub-question is discussed.

3.1 Definitions

To answer the main research question and it's sub-questions it is first needed to set clear definitions:

- *Type-correctness*: A prediction is type-correct if Mypy does not return an error on that prediction.
- *Ground-truth*: A prediction matches ground-truth if it is an exact match of an already existing type annotation for the predicted variable.
- *Accuracy*: Given a set of predictions the accuracy of that set is calculated by dividing the amount of type-correct predictions by the amount of total predictions. That is, the Set P with P_c type-correct predictions and P_t total predictions has an accuracy P_a described by equation 1:

$$P_a = P_c/P_t \tag{1}$$

- *Prediction Element:* 3 different elements in python source code can have type-annotations: Variables, Function return types and Function parameters, I will call these from now on Prediction Elements.
- *Confidence*: A prediction made by Type4Py has a confidence attached to it between 0 and 1, where a confidence of 1 represents a maximum confidence of Type4Py in its prediction.[2]
- *prediction-slots*: Type4Py can give more than 1 prediction with varying confidences attached to those predictions.[2] The full list of possible predictions per prediction element are referred to as prediction-slots. [2]
- *Prediction Strategy*: Because Type4Py returns a set of prediction-slots, I had to choose one specific prediction of the prediction-slots to be tested. The strategy used to choose from these prediction-slots will be subsequently referred to as the prediction strategy.
- *Threshold*: A threshold of a prediction strategy would be the minimum confidence needed for a prediction to be chosen.
- *Match Case*: To make this research comparable with previous research on Typilus [10] I grouped the predictions for the prediction elements in three cases with regards to existing type-annotations.
  1. **Empty**: There was no previous type-annotation in the source file for the prediction.
  2. **Match**: The previous type-annotation in the source file is exactly the same as the prediction.
  3. **Mismatch**: The previous type-annotation in the source file is not exactly the same as the prediction.

3.2 General Methodology

The methodology used to answer both research sub-questions is largely the same.

Dataset: First, I selected a subset of type-checked Python source code, namely a type-checked version of the ManyTypes4Py Dataset [3]. The reason behind choosing type-correct source-files was because it made it easier to find errors produced by the predictions tested. If I had used code that would already have a lot of errors when checked by Mypy it could impact the results negatively, because it could skew the results to lower accuracy due to existing type-errors

Type4Py: Then, I used Type4Py's model to generate predictions for these files. For each predictable element I got a list of prediction-slots. [2].

P1 strategy: When I had to choose a prediction strategy for picking predictions from the available prediction-slots. I

implemented a greedy prediction strategy I named *P1 Strategy*. Because the prediction-slots from Type4Py are ordered in descending confidence. I used the first prediction in the prediction-slots on the hypothesis that higher confidence would be generally more type-correct. The P1 Strategy also has the added benefit that when comparing the evaluations of Type4Py's predictions with different thresholds, the set of predictions with a lower threshold is a super set of the predictions with higher thresholds. If we take for example the following 3 predictions:

x: [(str, 0.8), (int, 0.6), (list, 0.4)]
y: [(int, 0.6), (str, 0.4)]
z: [(list, 0.4)]

and we run the P1 strategy with a threshold of 0.7, 0.5 and 0.3. The following predictions are chosen:

threshold:	0.7	0.5	0.3
x:	(str, 0.8)	(str, 0.8)	(str, 0.8)
y:	-	(int, 0.6)	(int, 0.6)
z:	-	-	(list, 0.4)

If we would take two runs on different thresholds and denote the set with a higher threshold as set A and the set with a lower threshold as set B. It is clear that every prediction in set A, with a confidence higher than set B, is in set B. In this research, I applied the P1 Strategy with the following thresholds: 0.25, 0.5, 0.75

Type application and Checking: Finally, the predictions were applied to the source files and typechecked by Mypy. The output was combined with other data, subsequently referred to as meta-data about the predictions which includes their confidence, Match case, what kind of Prediction element they were and line-number in the source code.

3.3 Methodology used to answer RQ1

For research question 1 I wanted to know how accurate Type4Py's predictions were on the thresholds 0.25, 0.5, 0.75. Because of me keeping meta-data of the predictions, I could evaluate the results of different Prediction elements and see if there were differences between those in terms of accuracy and proportion. I was mainly interested in the Empty and Mismatch cases because those are instances where Type4Py might make better predictions than a developer or can help developers make good type annotations. As a whole, the data could help theorize about any correlation between a minimum confidence (measured by using a threshold) and accuracy. Because I have used the P1 Strategy, I could also get more exact results on accuracy in ranges of the following confidence percentages: 25-50%, 50-75%. Which in turn could give more information about accuracy regarding specific confidence levels of Type4Py.

3.4 Methodology used to answer RQ2

For answering research question 2 I specifically focused on the accuracy and proportion of the Match case. This gives information about how type-correct Type4Py is when it is also matching ground-truth. After several preliminary runs of the pipeline, it became clear that some predictions would be labeled as a mismatch that were fundamentally the same. For example, the type [builtins.str] is the same the type [str] in terms of type-correctness. These would however, be flagged as a mismatch and could have caused a skewing of the data. Therefore, I sanitized these obvious false mismatches to get a more accurate answer on RQ2, with built-in functionaliaty of LibSA4Py [11].

4 Responsible Research

Responsible research describes the ethical aspects of the research done and the reproducibility of my methods. In terms of ethical aspects, there are no major concerns I have to reflect on. One thing that is however, important to note is that I have an individual gain by doing research that can improve development for Python. Either academically or reputation wise, it would be a benefit for me if my research had nice results.

Therefore, the most important thing I wanted to focus on in doing this research is making it reproducible. That is partly why I used the Open-source LibSA4Py [11] library to have transparency in the tools I used. Additionally, I made sure to make the pipeline modular to be able to be used for evaluating other type-prediction tools or implementing other type-evaluation functionalities. I made sure to document how the tool I made can be used by others by adding instructions int he LibSA4Py .README file.

5 Evaluation Setup:

For implementing the methodology I made a pipeline in LibSA4Py [11]. Instead of making a repository myself I wanted to add to an already existing Open-source project that had overlap with my research. It was also practically a good choice because a lot of the functionality I was going to need was already in there. A diagram of the Pipeline is shown in figure 1 which is implemented at: github.com/saltudelft/libsa4py/tree/MyType4Py.

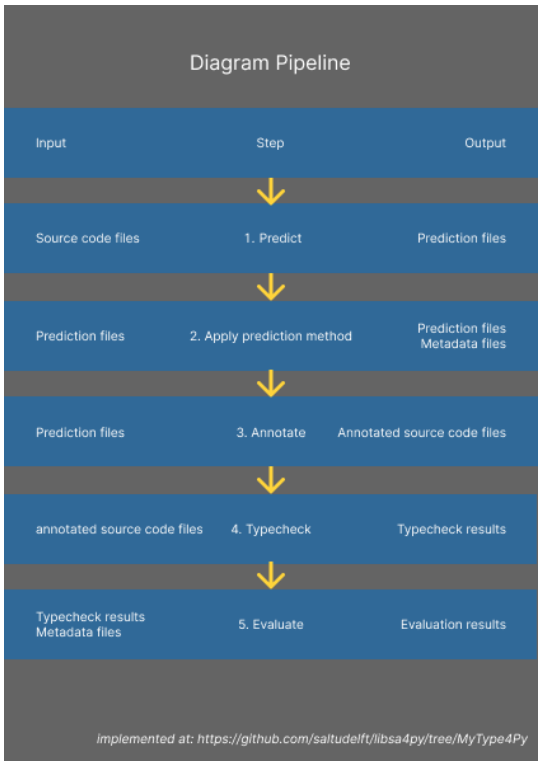

Figure 1: Pipeline Diagram

For reproducibility and transparency's sake I implemented the pipeline in steps with distinct input and output which could also make it useful for future research if researchers only need certain steps from the pipeline.

For checking type-correctness with Mypy certain error codes can be enabled/ disabled [12]. By disabling error codes, Mypy will not return errors having those error codes. Because I was more interested in the lower bound of accuracy than the higher bound, I tried to minimize the amount of error-codes I disabled. The assumption made was that more error codes disabled would mean generally higher accuracy. The following error codes were disabled when running the pipeline: [abstract, has-type, import, no-redef, syntax, var-annotated, return, attr-defined, call-overload]

6 Evaluation and Results

This section describes the evaluation and results of the methodology and its implementation described in the previous sections.

6.1 Overall results and Evaluation

Results: In table 1, 2 and 3 the results for running the P1 Strategy with the thresholds 0.25, 0.5 and 0.75 are displayed. The amount of predictions differs per threshold because of the fact that every set of predictions with a lower threshold has all predictions of a set with a higher threshold in addition to the predictions that are only accepted at the lower threshold. For all runs 138,066 files were analyzed. There is a difference between the total amount of predictions per run because of the fact that there are more prediction-slots that only have lower confidence.

Threshold = 0.25	Proportion (%)	Accuracy (%)	Total predictions	type-correct predictions
Empty	62	86	1,649,185	1,415,684
Match	15	95	393,856	373,985
Mismatch	23	91	602,465	546,296
Total	100	88	2,645,506	2,335,965

Table 1: Type checking accuracy of Type4Py's predictions with the P1 Strategy on the 0.25 threshold displaying different match cases

Threshold = 0.5	Proportion (%)	Accuracy (%)	Total predictions	type-correct predictions
Empty	56	89	982,696	870,443
Match	22	97	383,297	370,506
Mismatch	22	94	392,896	368,350
Total	100	91	1,758,889	1,609,299

Table 2: Type checking accuracy of Type4Py's predictions with the P1 Strategy on the 0.5 threshold displaying different match cases

Threshold = 0.75	Proportion (%)	Accuracy (%)	Total predictions	type-correct predictions
Empty	46	92	531,125	490,082
Match	31	98	355,706	350,235
Mismatch	23	97	266,337	257,430
Total	100	95	1,153,168	1,097,747

Table 3: Type checking accuracy of Type4Py's predictions with the P1 Strategy on the 0.75 threshold displaying different match cases

Evaluation: Looking at tables 1, 2 and 3 and figure 2 it becomes clear that higher threshold runs (containing predictions with higher confidence) have higher accuracy. This is clear for all match cases (Empty, Match and Mismatch) and if we look at the proportions it seems that higher threshold also indicates more predictions matching ground-truth which agrees with the evaluations of Type4Py's paper [2].

6.2 Confidence range results and evaluation

Results: A possible problem with interpreting the results is that a run with threshold 0.25 also includes all the predictions of runs with higher thresholds such as 0.5 and 0.75. To take advantage of the added benefit of applying the P1 Strategy, I subtracted the type-correct and total predictions of the higher threshold runs from the lower threshold runs. With this method I got results for accuracy in the confidence ranges of 25-50% and 50-75%. I combined these ranges with the threshold run of 75% (which is the same as the confidence range 75-100% to get again three ranges of accuracy which is shown in table 4 and figure 3. I also kept track of the proportions of the different match cases which are displayed in figures 4, 5 and 6.

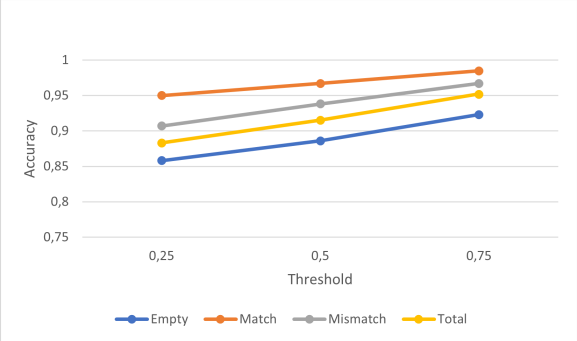

Figure 2: Type checking accuracy of Type4Py's predictions with the P1 Strategy on the 0.25, 0.5 and 0.75 threshold

	25-50%	50-75%	75-100%
Empty	81	84	92
Match	33	73	98
Mismatch	85	88	97
Total	82	84	95

Table 4: Type4Py's accuracy in percentages with confidence ranges of 25-50%, 50-75% and 75-100%

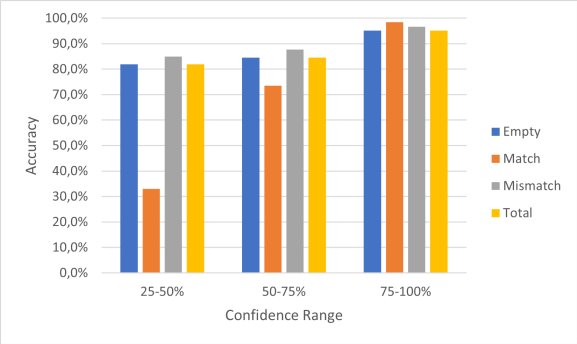

Figure 3: Type4Py's accuracy in percentages in confidence ranges of 25-50%, 50-75% and 75-100%

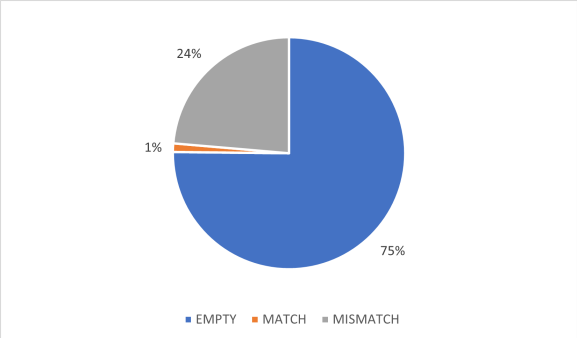

Figure 4: Proportions of match cases in confidence range 25-50%

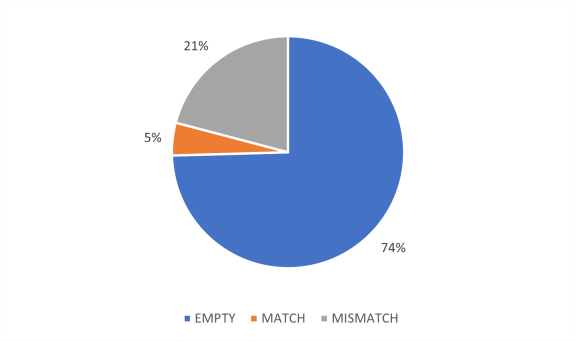

Figure 5: Proportions of match cases in confidence range 50-75%

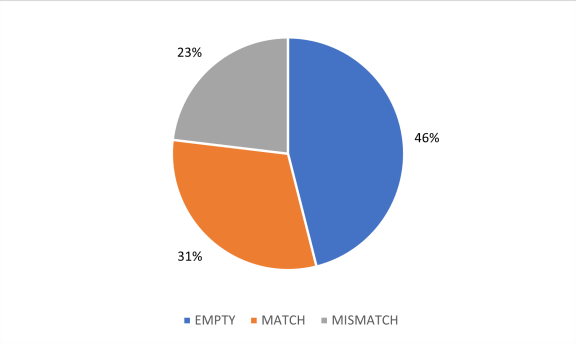

Figure 6: Proportions of match cases in confidence range 75-100%

Evaluation: The results in table 4 and figure 3 solidify the result of 6.1 by again showing an increasing accuracy for the increase of confidence in predictions. Now working with ranges shows that even at the lowest confidence range of 25-50% Type4Py's predictions are still type-correct 82% of the time and at the confidence range of 75-100% that accuracy increases to 95%. Figures 4, 5 and 6 show also a steady increase of the amount of matches when confidence in the predictions increases, which confirms the evaluation of Type4Py's research. [2]

6.3 Match case specific results and evaluation

Evaluation: When I applied the P1 Strategy to a file I also kept track of metadata about the predictions. The results were also split up per prediction element category (either variable, function parameter or function return value). Figures 7, 8 and 9 show the accuracy of the different prediction element categories per match case on different thresholds. Table 5 shows the proportions of the prediction element categories.

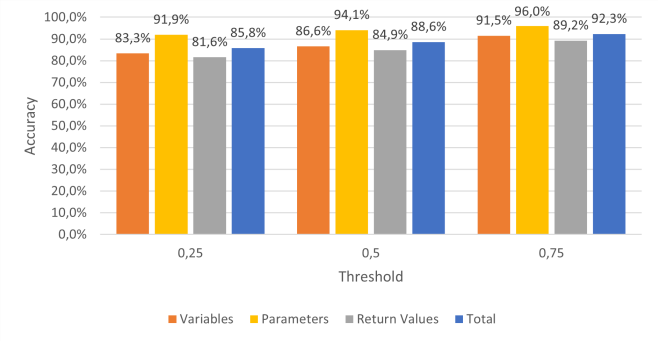

Figure 7: Accuracy for prediction element categories on the thresholds 0.25, 0.5, 0.75 and match case Empty

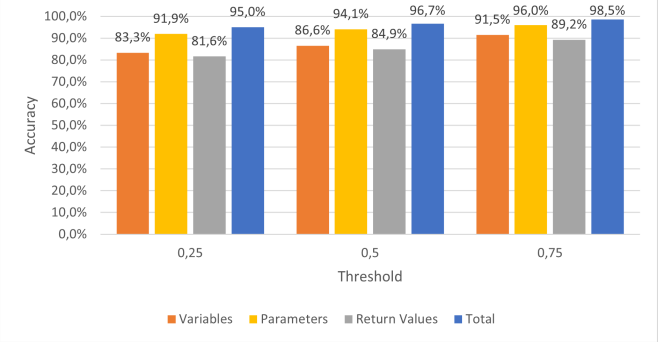

Figure 8: Accuracy for prediction element categories on the thresholds 0.25, 0.5, 0.75 and match case Match

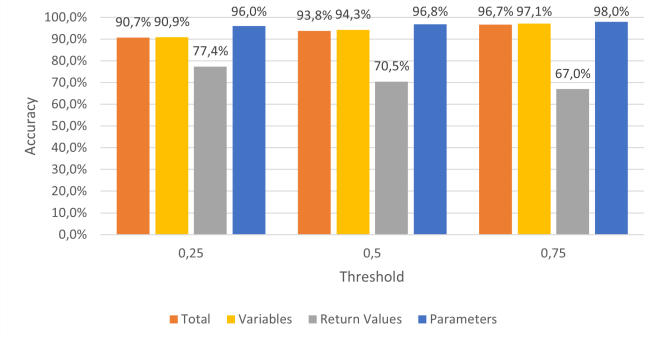

Figure 9: Accuracy for prediction element categories on the thresholds 0.25, 0.5, 0.75 and match case Mismatch

Threshold	0.25	0.5	0.75
Variables	64.9%	68.3%	72.9%
Parameters	10.8%	10.0%	9.5%
Return types	24.3%	21.7%	17.6%
Total	100%	100%	100%

Table 5: prediction element category proportions in percentages with the thresholds of 0.25, 0.5 and 0.75

Results: Looking at figures 7, 8 and 9 shows that the accuracy for parameter prediction is the highest and return types the lowest with variable predictions in the middle for each match case. Figure 9 shows an interesting trend where a higher threshold significantly decreases the accuracy. From table 5 it shows that the proportion of return type predictions is also decreasing with higher thresholds. Both of these findings are in agreement with the performance evaluation of Type4py's paper [2], where return type predictions were also the least accurate.

7 Discussion

By running the pipeline over a total of 138,066 files on three different thresholds, I have found that Type4Py has a good accuracy between 88% on a 0.25 threshold and 95% on a 0.75 threshold. Furthermore, it has become clear that higher confidence in predictions implies a higher chance of type-correctness of those predictions. If we compare these results against the ones in Typilus which had an overall accuracy of 89% [10], Type4Py seems to be performing quite well. For the predictions that match ground-truth an even higher accuracy was measured on all 3 thresholds.

One problem I faced when evaluating the errors that were generated by Mypy was linking them to the predictions made by Type4Py. I linked each prediction to a line-number and error to a line number and if there existed an error and a prediction on that linenumber I would flag the prediction as false. I would have rather also used column numbers for extra precision but because of time-constraints for the implementation and the fact that column numbers can change when applying type annotations on the same line I chose the line number only approach. This could give some false negatives when it comes to parameters and return types on the same line, but looking at figure 7, 8 and 9 it seems that there were still plenty of cases where predictions for parameters and return types had different accuracy scores.

This is, however, not a clear-cut case for the type-correctness of Type4Py's predictions. It is evident that with less disabled error codes in the Mypy configuration more errors can be found. The question is how many of these errors are actually Type4Py's "fault" and how many are because of the coding style/ practices of the developers who made the source files. These results point to the fact that Type4Py is not only a good potential tool for developers in the sense that it will make predictions that adhere to the choices of the developers. It will also produce results that are (when checked with Mypy) mostly type-correct.

8 Conclusions and Future Work

8.1 Conclusion

To answer sub-research question 1: "How many of Type4Py's predictions are type-correct". Using the P1 Strategy:

- on a threshold of 0.25, Type4Py's accuracy was 88%
- on a threshold of 0.5, Type4Py's accuracy was 91%

- on a threshold of 0.75, Type4Py's accuracy was 95%

To answer sub-research question 2: "How many of Type4Py's predictions are type-correct and match ground-truth". Using the P1 Strategy:

- on a threshold of 0.25, Type4Py's accuracy was 95%
- on a threshold of 0.5, Type4Py's accuracy was 97%
- on a threshold of 0.75, Type4Py's accuracy was 98%

To answer the main research question: "How well does Type4Py perform when validated by the static type-checker Mypy?". I analyzed 138,066 files and 2,645,506 predictions for those files. I compared Type4Py with a similar tool for predicting types that also uses Machine Learning, namely Typilus [10]. Typilus has one measured accuracy of 89% [10], whereas, I had three measured accuracy's in total. At the 0.25 threshold Type4Py seems to be 1% less accurate in type-correctness compared to Typilus. But for the thesholds 0.5 and 0.75 Type4Py's measured accuracy's where higher. Overall Type4Py's predictions are for the most part type-correct when checked by Mypy. An interesting finding was that when the confidence of the predictions increased (higher threshold run), the accuracy of specifically return type predictions decreased.

8.2 Future work

Type4Py has now been evaluated on two fronts: with an MRR of 71.7% [2] it shows that its predictions are often the ones developers would annotate themselves and in terms of type-correctness with an accuracy between 88% and 95% it show that its predictions are also mostly type-correct.

A future approach to evaluating the type-correctness of Type4Py could also be to a apply a Combinatoral Strategy to complete files with Type4Py's predictions where researchers can check if given all predictions for all type-slots, there exists a possible combination of predictions where the whole file is type-correct.

It would also be interesting to add a testing feature using Mypy to the VSCode extension of Type4Py to give developers only type-correct predictions when developing in Python.

Acknowledgements

I hereby want to thank Amir M. Mir and Sebastian Proksch for their valuable feedback and guidance with this research

References

[1] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 155 – 165. [Online]. Available: https://doi.org/10.1145/2635868.2635922

[2] A. M. Mir, E. Latoskinas, S. Proksch, and G. Gousios, "Type4py: Deep similarity learning-based type inference for python," *CoRR*, vol. abs/2101.04470, 2021. [Online]. Available: https://arxiv.org/abs/2101.04470

[3] A. M. Mir, E. Latoskinas, and G. Gousios, "Manytypes4py: A benchmark python dataset for machine learning-based type inference," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 585–589. [Online]. Available: https://ieeexplore.ieee.org/document/9463150

[4] G. van Rossum, J. Lehtosalo, and Langa, "Pep 484 - type hints," https://peps.python.org/pep-0484/, 2014, (Accessed: June 1, 2022).

[5] M. Hassan, C. Urban, M. Eilers, and P. Müller, "Maxsmt-based type inference for python 3," in *Computer Aided Verification*, H. Chockler and G. Weissenbacher, Eds. Cham: Springer International Publishing, 2018, pp. 12–19.

[6] E. Maia, N. Moreira, and R. Reis, "A static type inference for python," 2012.

[7] M. Salib, "Faster than c: Static type inference with starkiller," in *in PyCon Proceedings, Washington DC*. SpringerVerlag, 2004, pp. 2–26.

[8] Z. Pavlinovic, "Leveraging program analysis for type inference," Ph.D. dissertation, 2019.

[9] M. Pradel, G. Gousios, J. Liu, and S. Chandra, *TypeWriter: Neural Type Prediction with Search-Based Validation*. New York, NY, USA: Association for Computing Machinery, 2020, p. 209–220. [Online]. Available: https://doi.org/10.1145/3368089.3409715

[10] 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*. ACM, jun 2020. [Online]. Available: https://doi.org/10.1145%2F3385412.3385997

[11] A. M. Mir and G. Gousios, "Libsa4py: Light-weight static analysis for extracting type hints and features." https://github.com/saltudelft/libsa4py, 2020.

[12] J. Lehtosalo and mypy contributors, "Mypy error codes," https://mypy.readthedocs.io/en/stable/error_codes.html, 2016, (Accessed: June 1, 2022).