With the increasing popularity of the dynamic programming languages, type prediction in dynamic programming languages has become a very well-known research problem. Type prediction is needed in dynamic typed languages to avoid simple compatibility error, increase IDE support and to handle APIs easily. Among the some other existing strategies to issue this problem, TypeWriter is the first combination of probabilistic type predictions with feedback-directed search based validation.

To annotate the dynamic typed program, it first extracts the useful information from the program such as identifiers (function and argument names), token sequences (code segment where any argument/return type has been used), function level comments and available types in the program that are locally defined or imported. Then it runs a neural network model (RNN) that gives the predicted types from the statically extracted data. It actually returns a probability distribution over a fixed set of types for each type slot in the program. After this it executes the feedback-guided search method to assign the correct type for each type slot from the predicted type list for that slot. It calculates a feedback score for each assignment with the help of gradual type checker (pyre) to get the number of missing types and error types in the program. Then it runs both greedy and non-greedy approach to search for the best assignment for every type slots and finds that the greedy approach is more powerful than non-greedy approach. The work is really outstanding in the context of precision (84-92% for top 5 predicted types) and recall (69-72% for top 5 predicted types) in comparison with the other competitors of the TypeWriter such as NL2Type, DeepTyper.

The thing that looks very good to me is that the approach is very well explained with suitable examples. The writers addressed exact research questions and supported with real time results like the effectivity of the TypeWriter's neural model and search method, how much different kind of context information contributes to the prediction ability etc.

The negative thing is that it largely depends on the performance of the gradual type checker. It assumes that gradual type checker will always produce correct results. But in any case of the gradual type checker does not work efficiently, then the performance of the TypeWriter may degrade largely. It leads the TypeWriter to type-correctness rather than type-soundness as there may be some assignments that give no error for the program but among them the developer wants only one correct assignment. So, it can reduce the dependency on the default gradual type checker rather can construct a module that will efficiently do this job with no errors. Besides, I think the feedback score function is much rigid regarding the weights of the missing types and error types. The writers could show a result/graph regarding the performance of the TypeWriter in context of these two weights to see what combination of these works well for the TypeWriter. It has also assumed that the function level comments will always be in good format. But there may be a very bad commenting style in the program that may greatly impact the performance of the TypeWriter.