NL2Type: Inferring JavaScript Function Types from Natural Language Information

Authors: Rabee Sohail Malik, Jibesh Patra, Michael Pradel

Presenter: Safa Keskin / 504191536

Outline

1. Objective
2. Dataset Characteristics
3. Data Analysis and Preprocessing
4. Recommendation System Detail
5. Results
6. Comparison With Previous Works
7. My Observations
8. Conclusion

The Objective

Objectives are:

- Providing missing types for JavaScript files
- Finding inconsistencies between predicted and provided types

Dataset Characteristics

- Real-world projects
- JS files documented with JSDoc

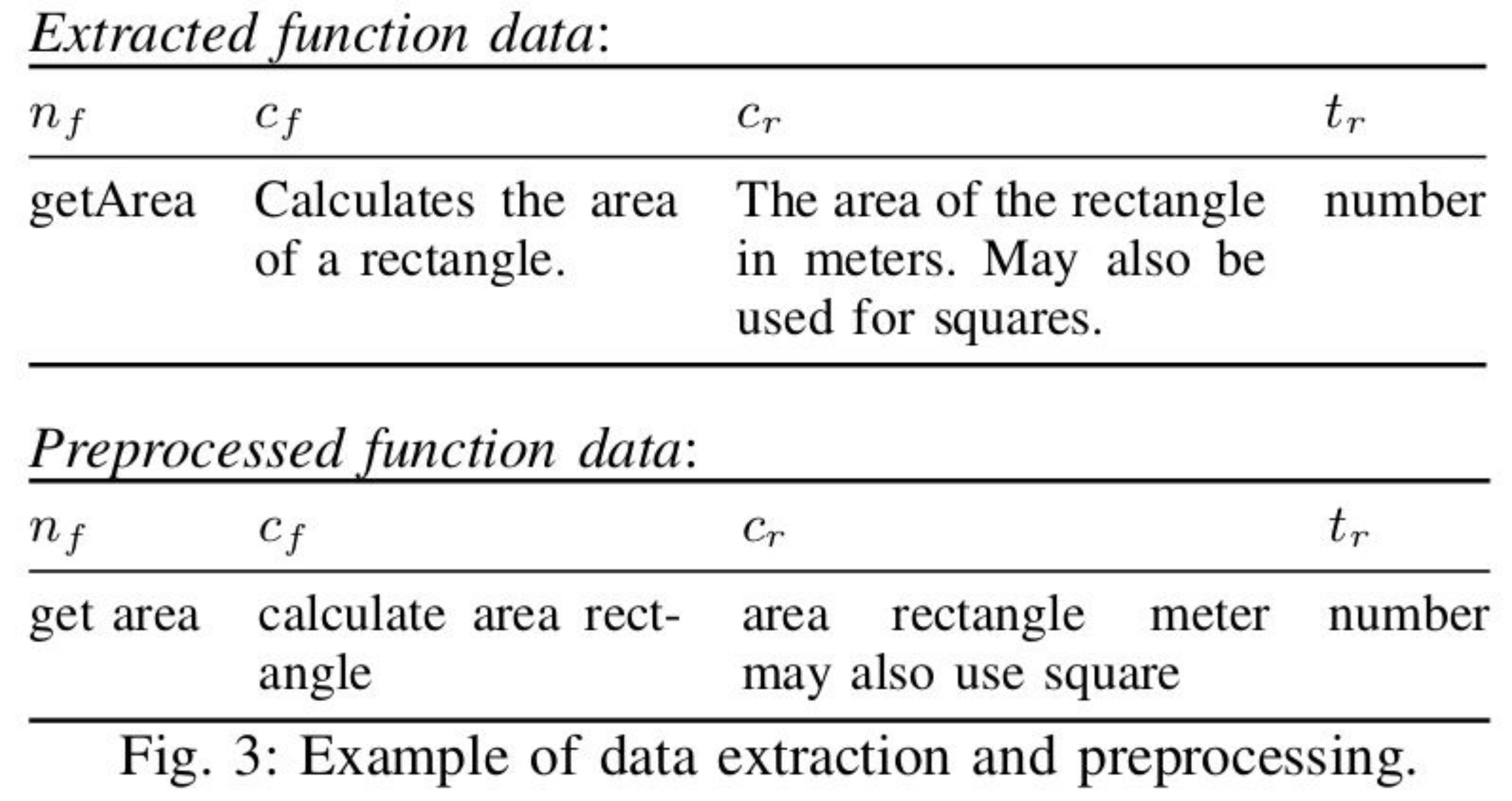
Data Analysis and Preprocessing

Data Extraction: Each function is visited and data extracted from JSDoc annotations

For a given function f:

- n_f = Name of the function
- c_f = Comment associated with f
- c_r = Comment associated with return type of f
- t_r = Return type of f
- P = Sequence of parameter data (tuple (n_p, c_p, t_p))
  - n_p = name of formal parameter p
  - c_p = comment associated with p
  - t_p = type of p

Comments include both useful and useless information



Extracted function data:

n_f	c_f	c_r	t_r
getArea	Calculates the area of a rectangle.	The area of the rectangle in meters. May also be used for squares.	number

Preprocessed function data:

n_f	c_f	c_r	t_r
get area	calculate area rectangle	area rectangle meter may also use square	number

Fig. 3: Example of data extraction and preprocessing.

Example of extracted and preprocessed data

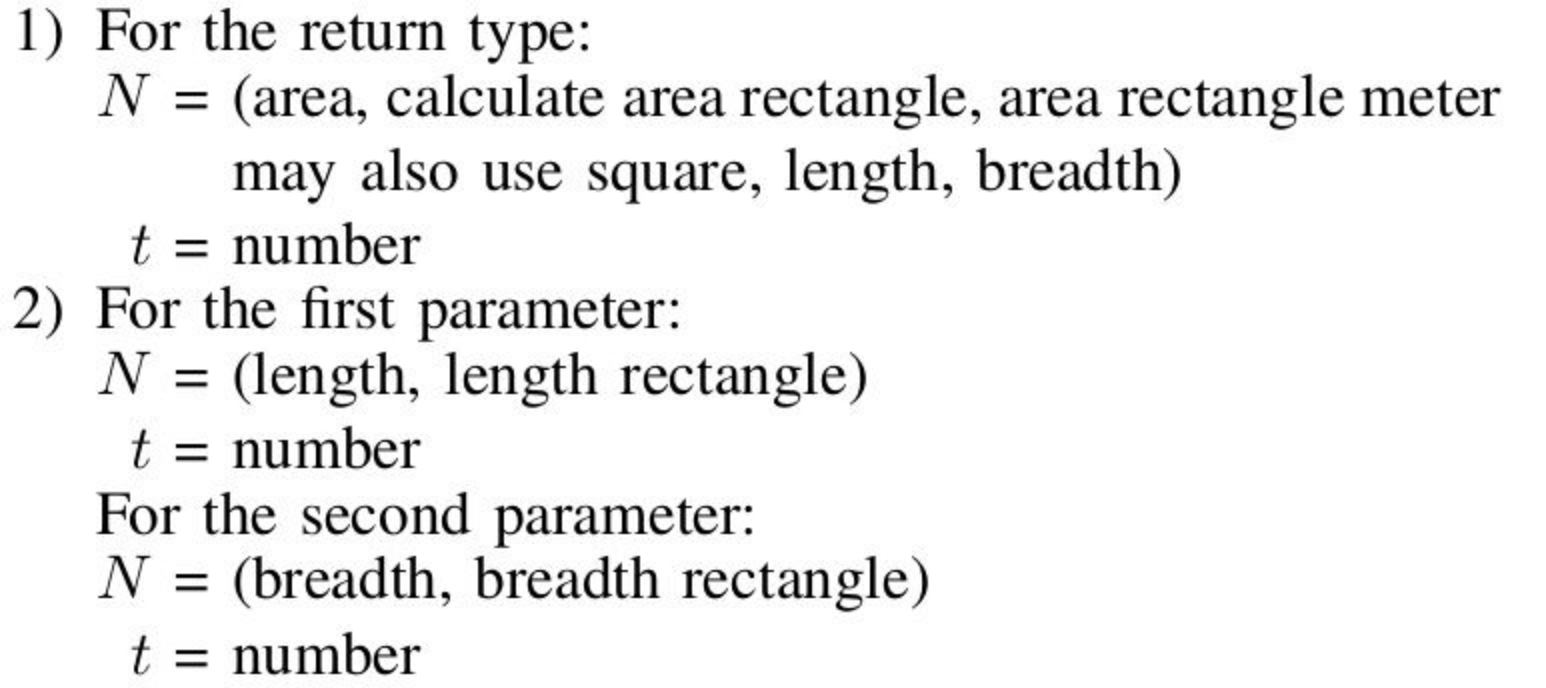
- In order to feed data to model, data should be converted into vectors
  - Word Embeddings (Word2Vec)
  - 2 word embeddings:
    - Comments
    - Identifier names

- Subset of all types occurred on comments are used.
    - $T \subseteq T_{all}$

Recommendation System Detail

- Recurrent neural network
    - Based on LSTM units

- Data Point
    - Return type
        - $N = (n_f, c_f, c_r, n_p^1, \dots , n_p)$ and $t = t_r$
    - Parameter type
        - $N = (n_p^i, c_p^i)$ and $t = t_p^i$



1) For the return type:

   N = (area, calculate area rectangle, area rectangle meter
   may also use square, length, breadth)

   t = number

2) For the first parameter:

   N = (length, length rectangle)

   t = number

   For the second parameter:

   N = (breadth, breadth rectangle)

   t = number

- Learning

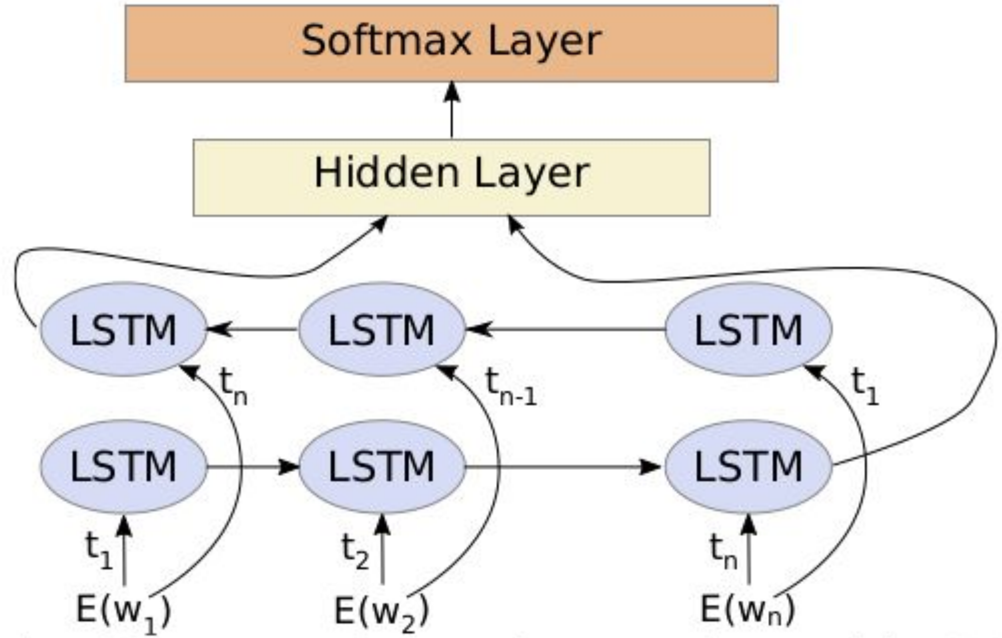


Fig. 4: Architecture of neural network used in NL2Type.

- Helper mapper: E
  - $E^* = w_1, \ldots, w_l$ -> $R^{l \times k}$

- While implementing
    - **JSDoc** is used for data extraction
    - Python **NLTK** is used for preprocessing
    - To convert words into embeddings, gensim’s **Word2Vec** is used
    - Recommender engine is implemented on top of **Keras**, using **TensorFlow** as backend

Results

- Evaluation metrics:
    - Precision
    - Recall
    - F1-score

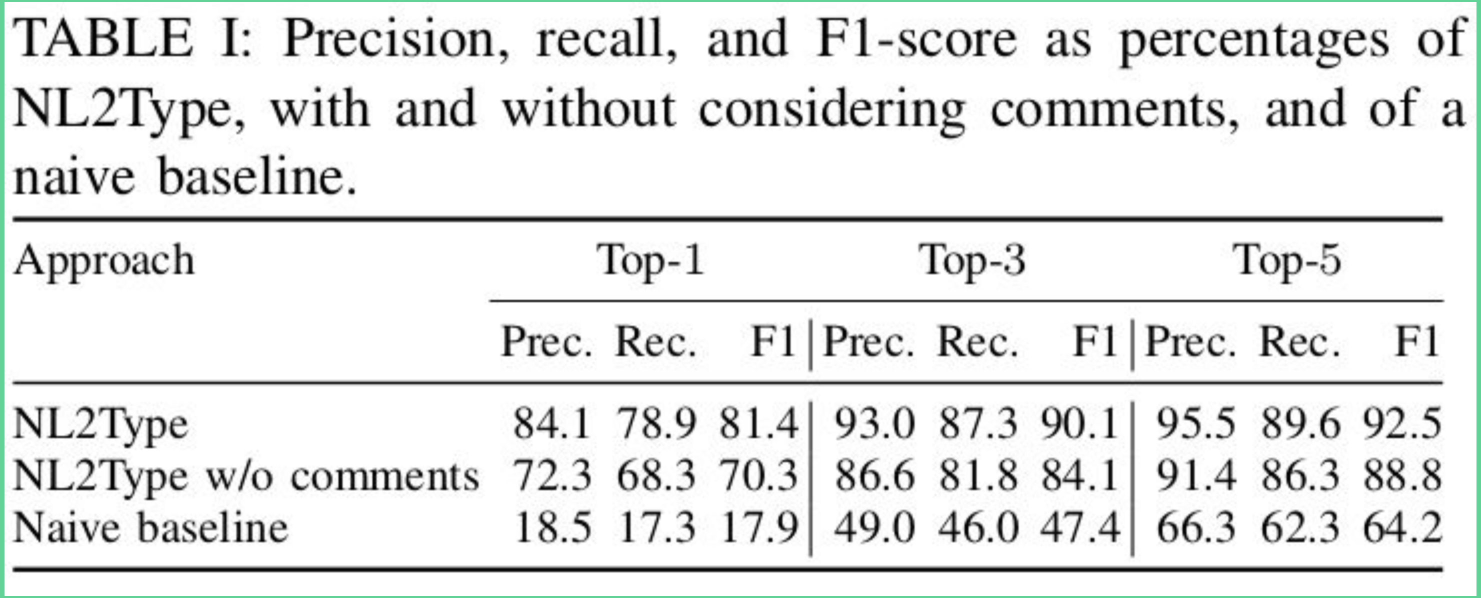
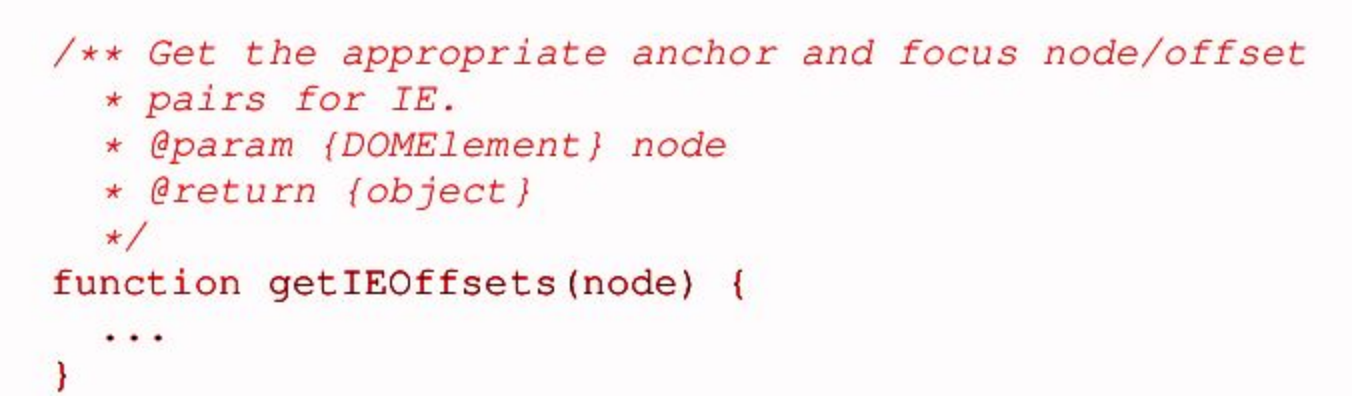


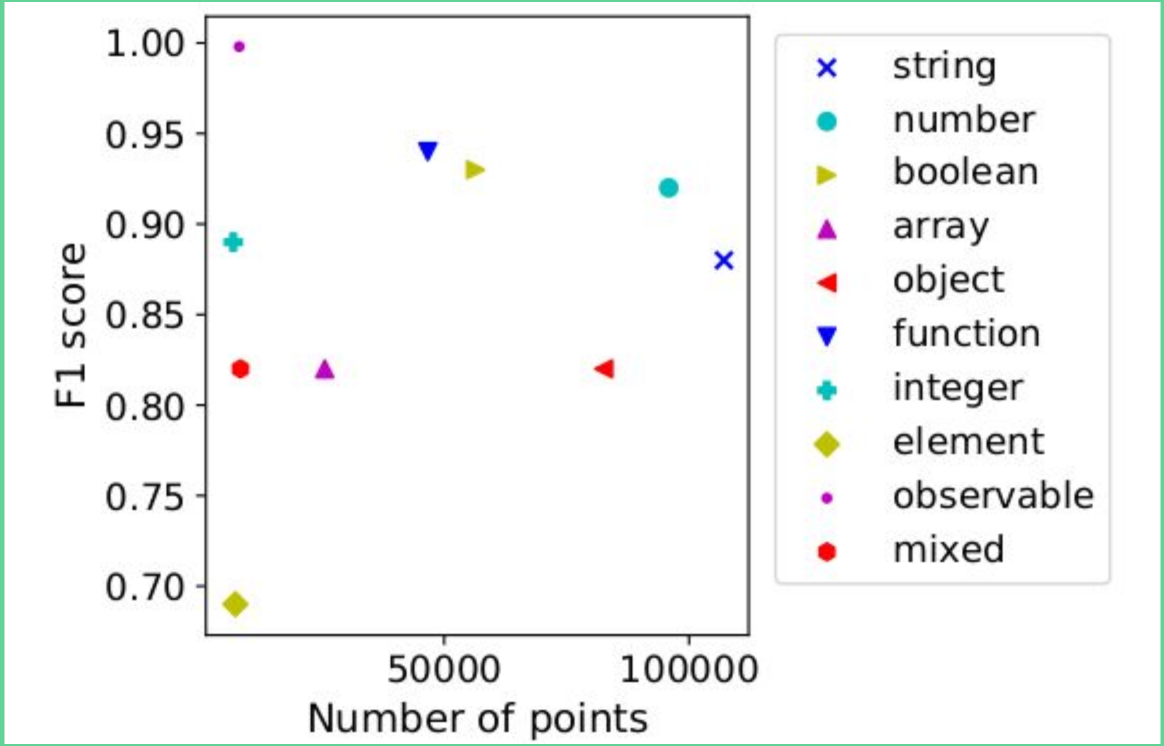
TABLE I: Precision, recall, and F1-score as percentages of NL2Type, with and without considering comments, and of a naive baseline.

Approach	Top-1			Top-3			Top-5		
	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
NL2Type	84.1	78.9	81.4	93.0	87.3	90.1	95.5	89.6	92.5
NL2Type w/o comments	72.3	68.3	70.3	86.6	81.8	84.1	91.4	86.3	88.8
Naive baseline	18.5	17.3	17.9	49.0	46.0	47.4	66.3	62.3	64.2

Example of correct prediction



```
/** Get the appropriate anchor and focus node/offset
  * pairs for IE.
  * @param {DOMElement} node
  * @return {object}
  */
function getIEOffsets(node) {
  ...
}
```


1.00
0.95
0.90
0.85
0.80
0.75
0.70
F1 score
50000
100000
Number of points
string
number
boolean
array
object
function
integer
element
observable
mixed

Comparison With Previous Works

Detecting Types

	Precision	Recall	F1-score
NL2Type	84.1%	78.9%	81.4%
JSNice	62.5%	45.0%	52.3%

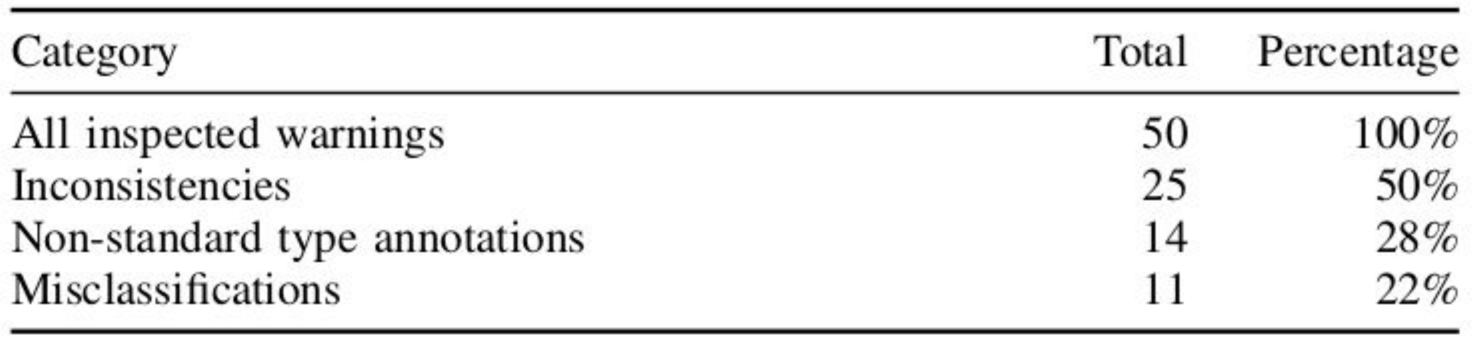
Detecting Types

	Precision	Recall
NL2Type	77.5%	44.6%
DeepTyper	68.6%	44.0%

- Investigating the code
- Outperforms tools that use Probabilistic Type Inference
- Uses comments
- Neural network
- Easy to transfer other languages, too

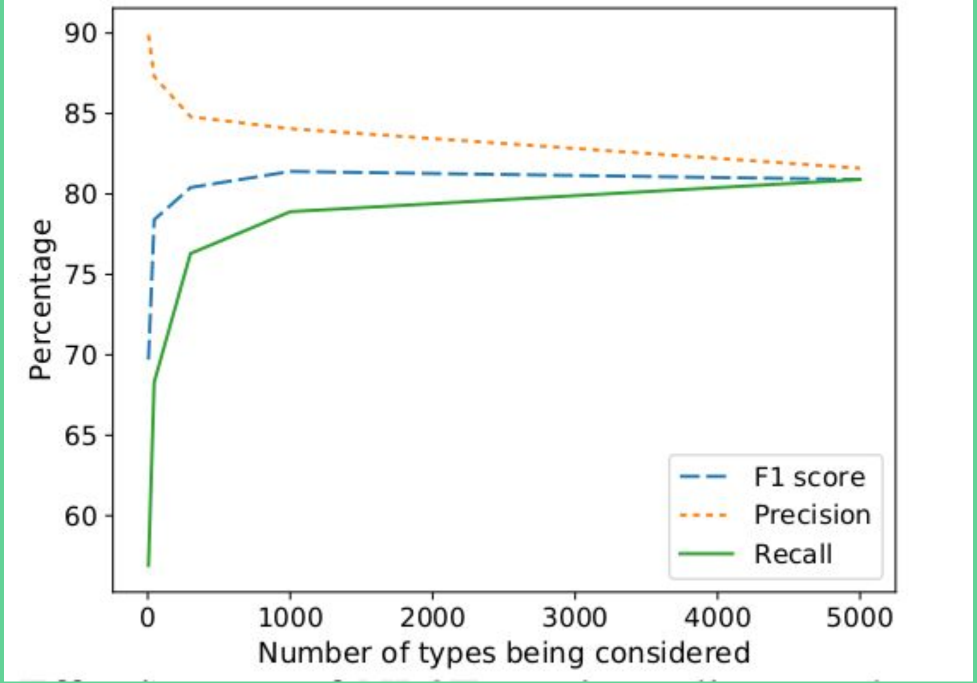
My Observations

Detecting Potential Inconsistencies



Category	Total	Percentage
All inspected warnings	50	100%
Inconsistencies	25	50%
Non-standard type annotations	14	28%
Misclassifications	11	22%

Effect of considered type numbers to the result



- This tool is not a requirement but it is nice to have such a tool

Conclusion

- Obviously better performance than alternatives
- Easy to understand
- Easy to use
- Predicts types well compared to others
- Open to development