code2vec:

Learning Distributed Representations of Code



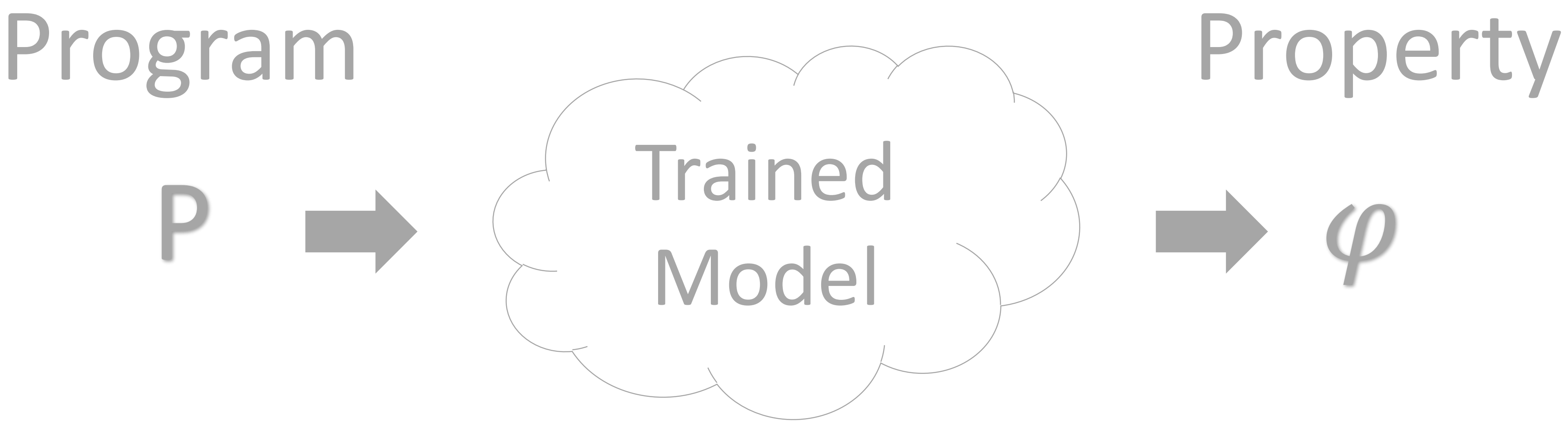

Uri Alon, Meital Zilberstein, Omer Levy, Eran Yahav

Facebook AI
Research

Technion

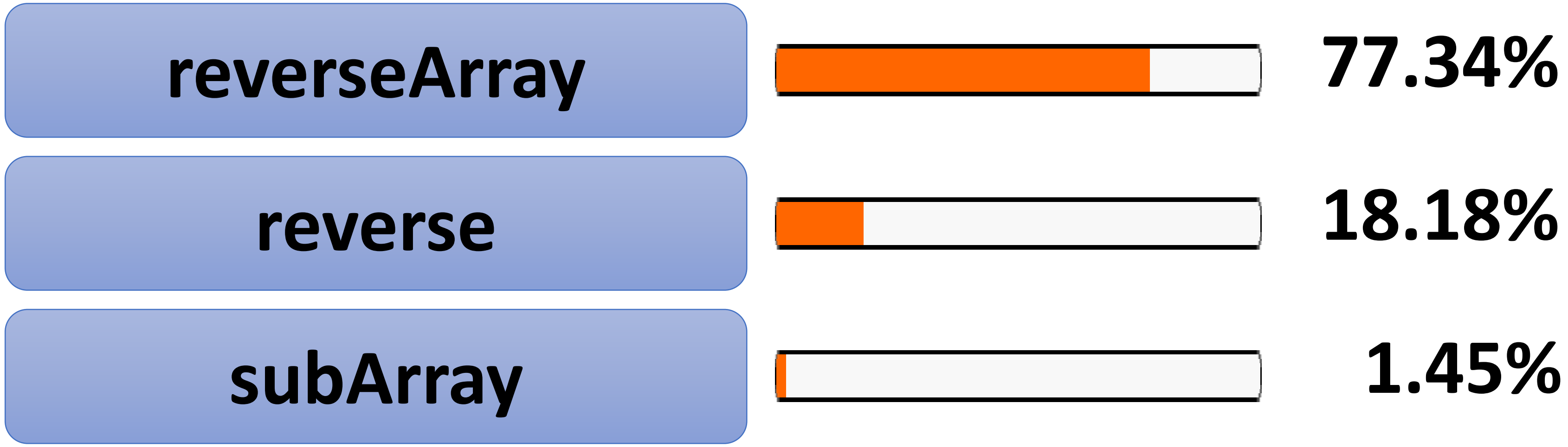
SEVENTH FRAMEWORK
PROGRAMME

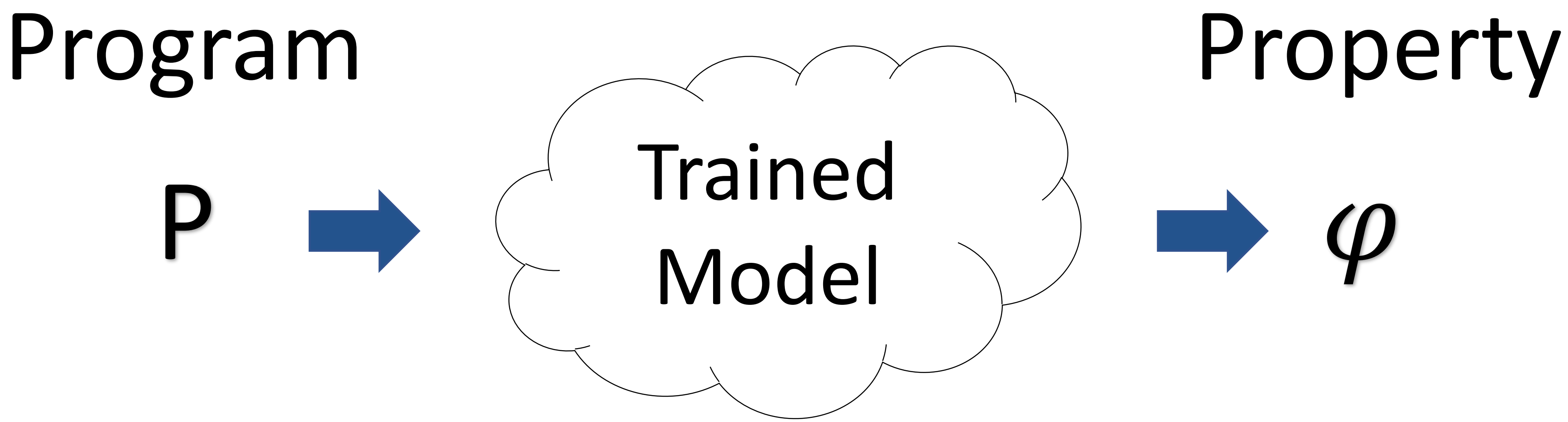
Predicting Properties of Programs



A Motivating Example: Semantic Labeling of Code

```
String[] ________(final String[] array) {
    final String[] newArray = new String[array.length];
    for (int index = 0; index < array.length; index++) {
        newArray[array.length - index - 1] = array[index];
    }
    return newArray;
}
```





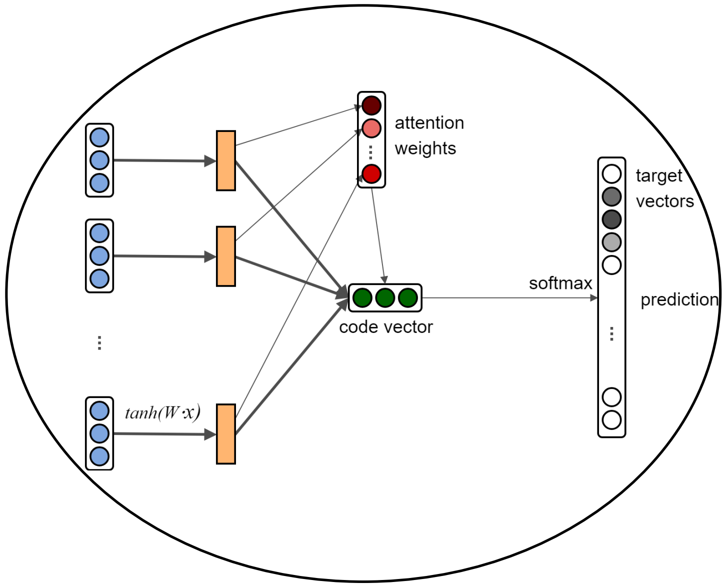
<u>Training data</u>
<u>(millions of examples)</u>:

$$\begin{cases} (P_1, \varphi_1) \\ (P_2, \varphi_2) \\ \ldots \\ (P_m, \varphi_m) \end{cases}$$

<u>Test data</u>: $(P', \underline{\varphi'})$

Program

P



Property

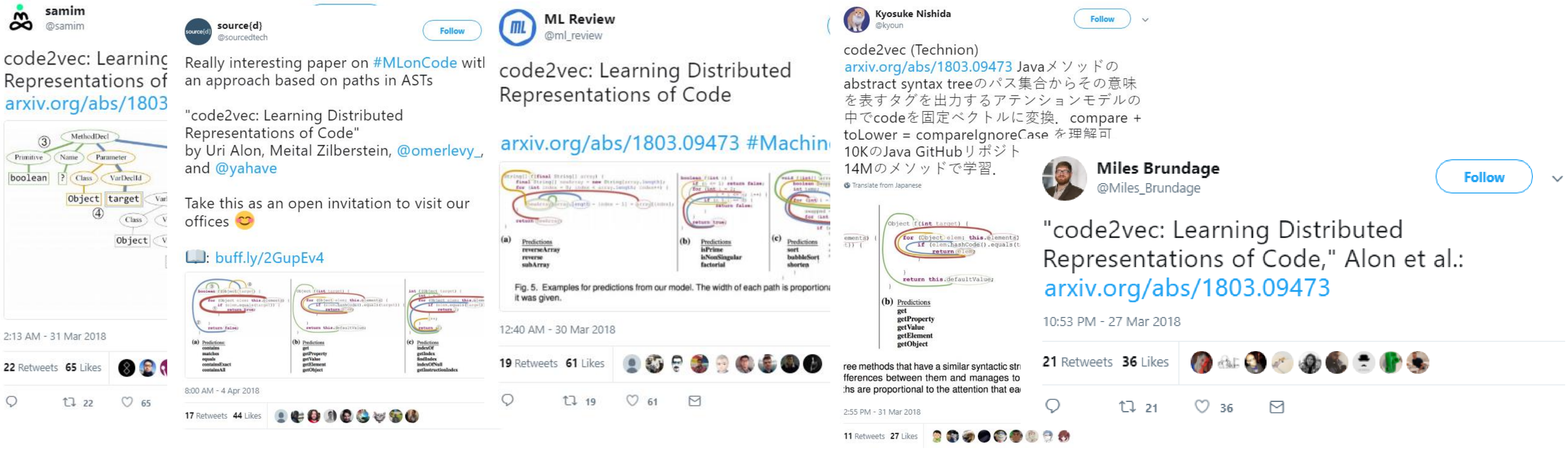
φ

Training data (millions of examples):

$$\begin{cases} (P_1, \varphi_1) \\ (P_2, \varphi_2) \\ \dots \\ (P_m, \varphi_m) \end{cases}$$

Test data: $(P', \underline{\varphi'})$

Code2vec: a neural network for predicting properties of code



Code2vec: a neural network for predicting properties of code

Example application: predicting method names

```
String[] ______(...) {
   final String[] = ...;
   ...
   return newArray;
}
```

reverseArray

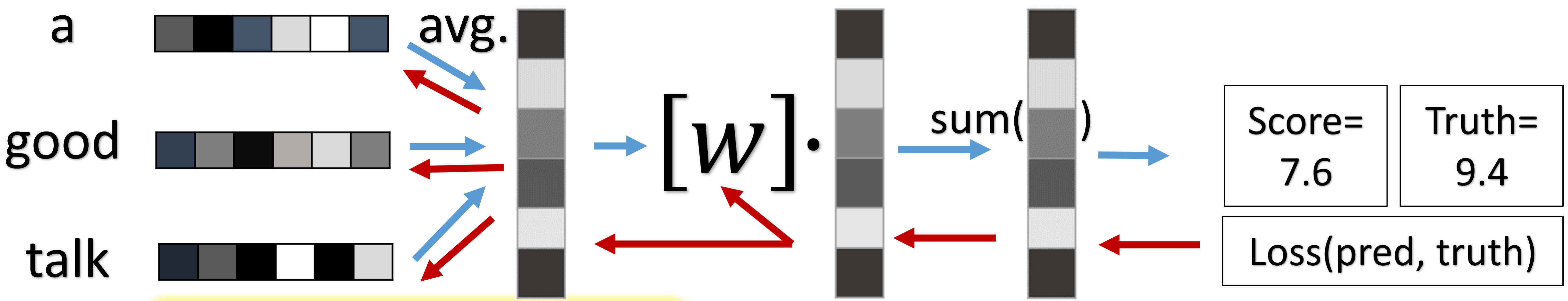
- A general approach – has many possible applications:
  - Yes/no malware, required dependencies, keywords / hashtags, clone detection…

How does code2vec work?

Neural Networks

- Sequences of simple algebraic functions over vectors and matrices

- A simple example: Predict how positive is a given sentence (regression)



$$vec \leftarrow vec - \alpha \frac{\partial loss}{\partial vec}$$

$$w \leftarrow w - \alpha \frac{\partial loss}{\partial w}$$

Back to our problem

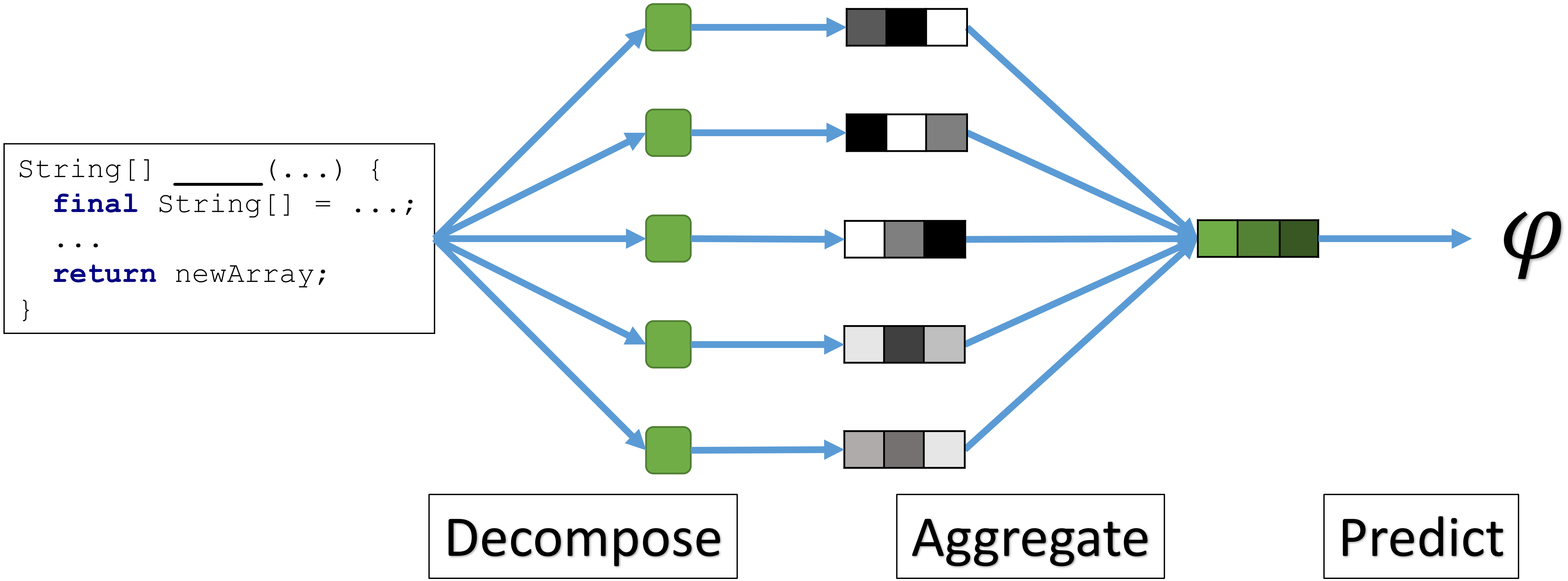
Two main challenges in encoding programs:

1. How to decompose programs to smaller building blocks?
   - Small enough to repeat across programs
   - Large enough to be meaningful

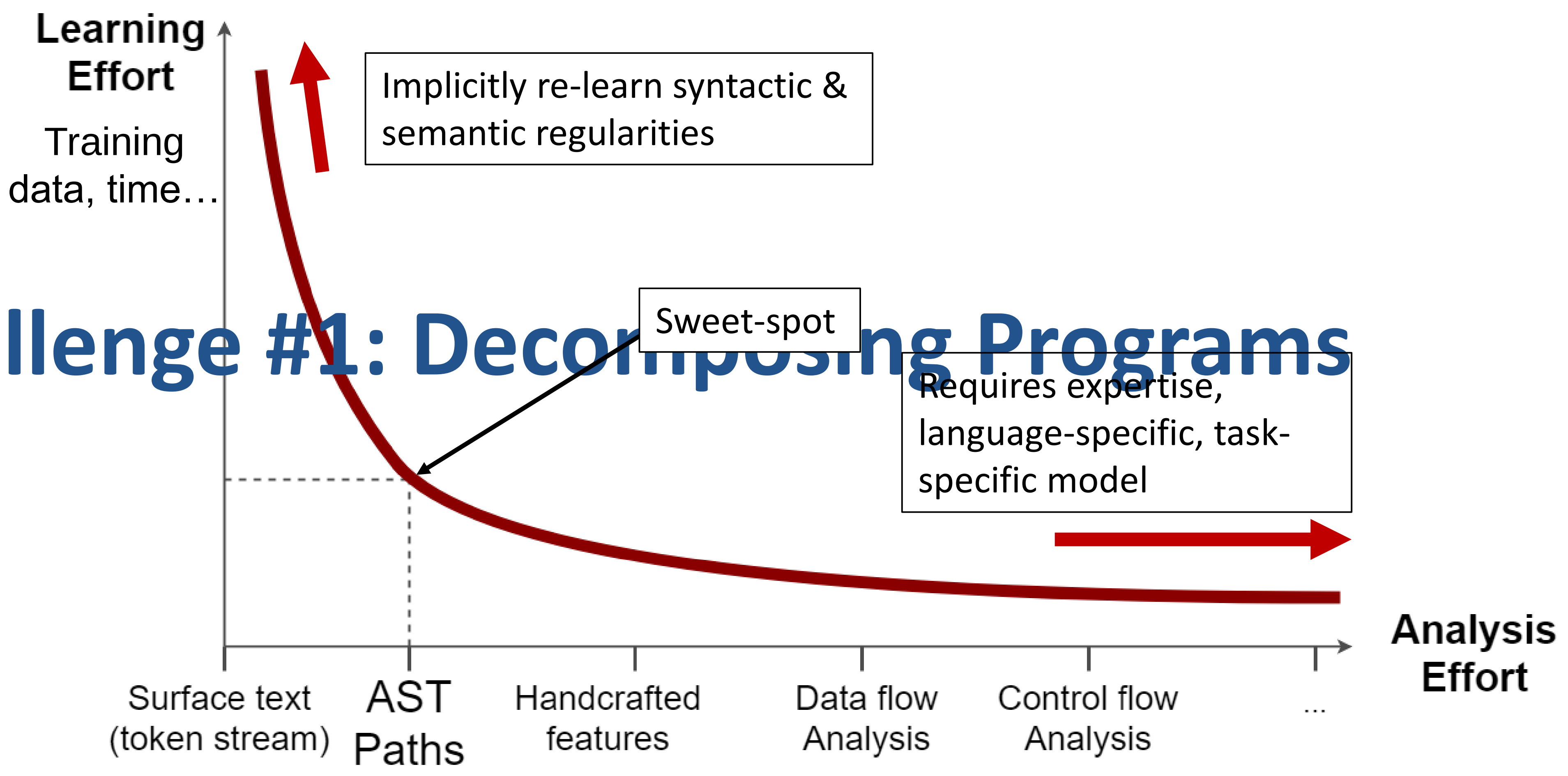
   the "bias-variance tradeoff"

2. How do we aggregate a set of these building blocks?

Code2vec: High-level Overview



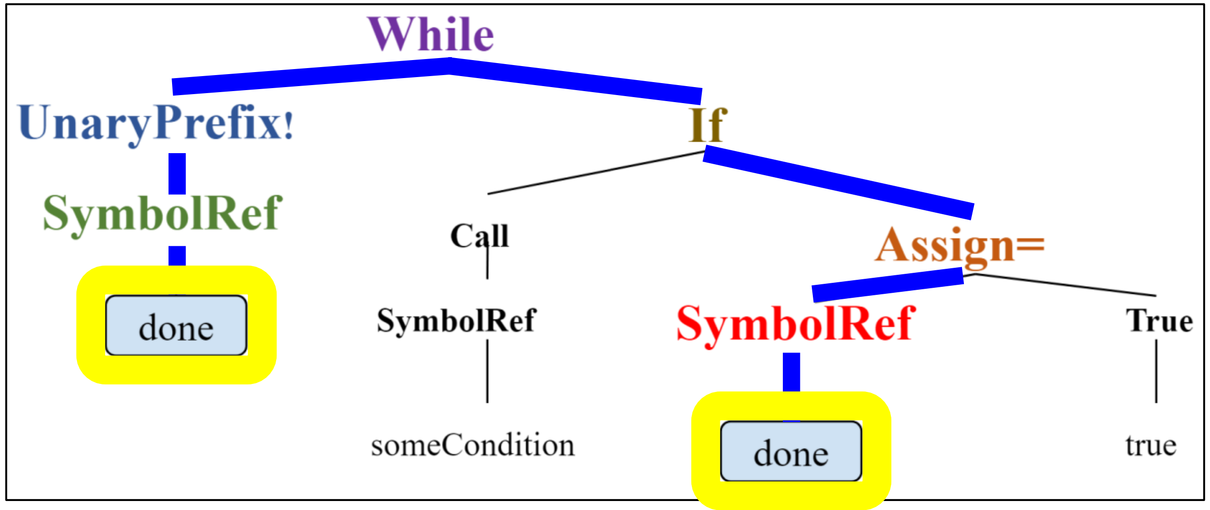
Challenge #1: Decomposing Programs



["A General Path-based Representation for Predicting Program Properties", PLDI'2018]

A Program as a Set of AST Paths

```
while (!done) {
    if (someCondition()) {
        done = true;
    }
}
```



(done, SymbolRef ↑

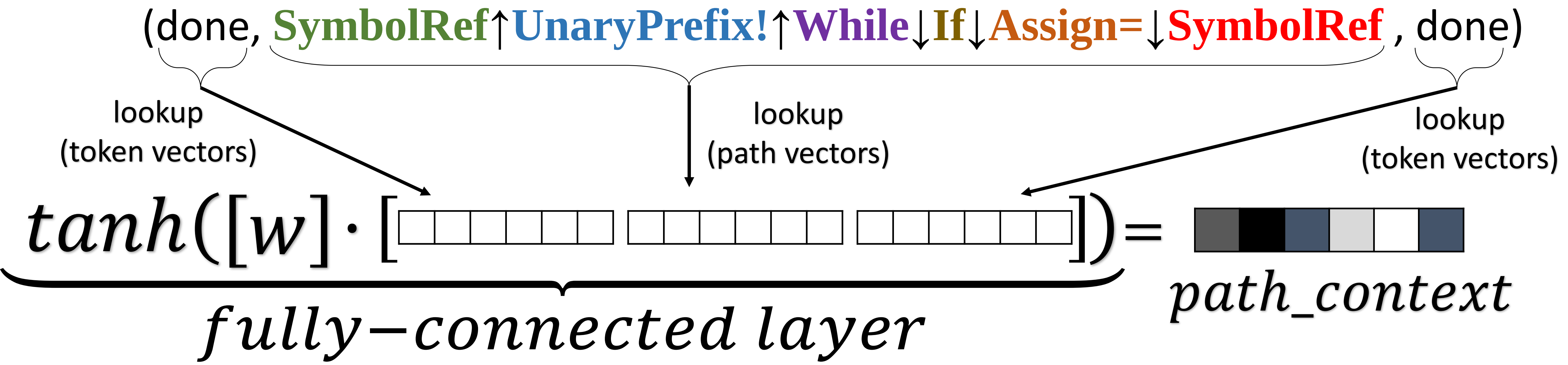
- AST paths capture some of the semantics, by using only the syntax.
- We represent a program as <u>the set of all its paths</u>.

[“A General Path-based Representation for Predicting Program Properties”, PLDI’2018]

Representing AST-Paths as Vectors

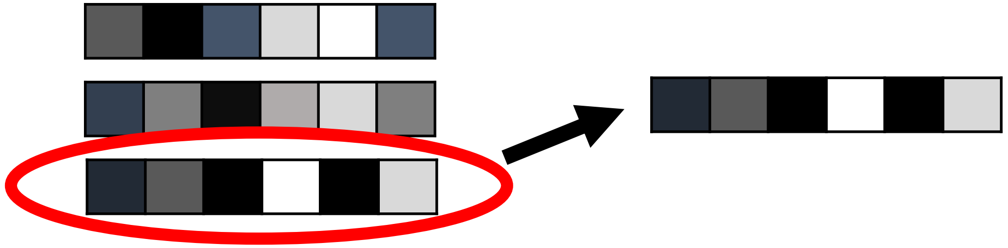
Two sets of learned vectors:

- Token vectors
- Path vectors

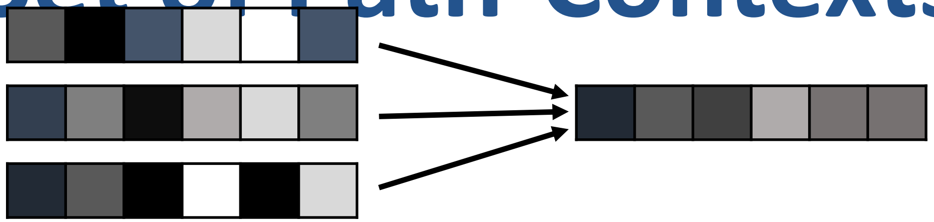


Challenge #2: Aggregating a Set of Path-Contexts

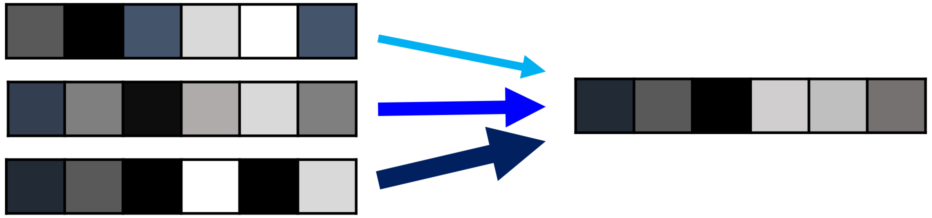
- Input: an arbitrary-sized set of vectors representing AST paths
  - Select the “most important vector”



  - Use all vectors, e.g., by averaging them



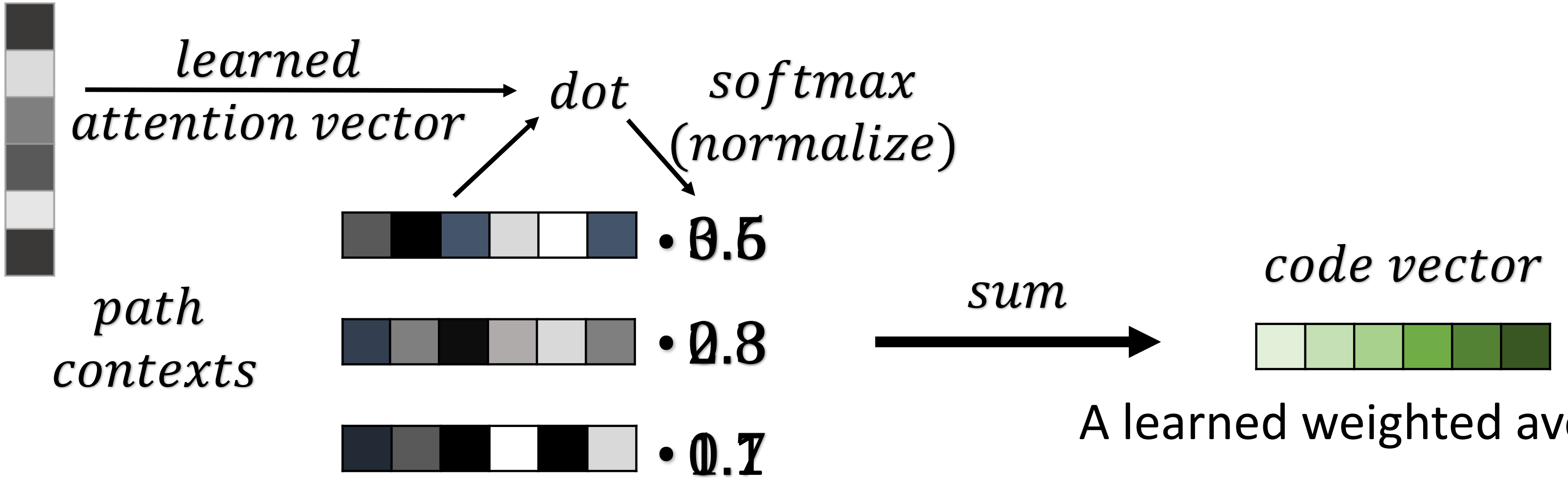
  - **Attention – a learned weighted average**



Attention

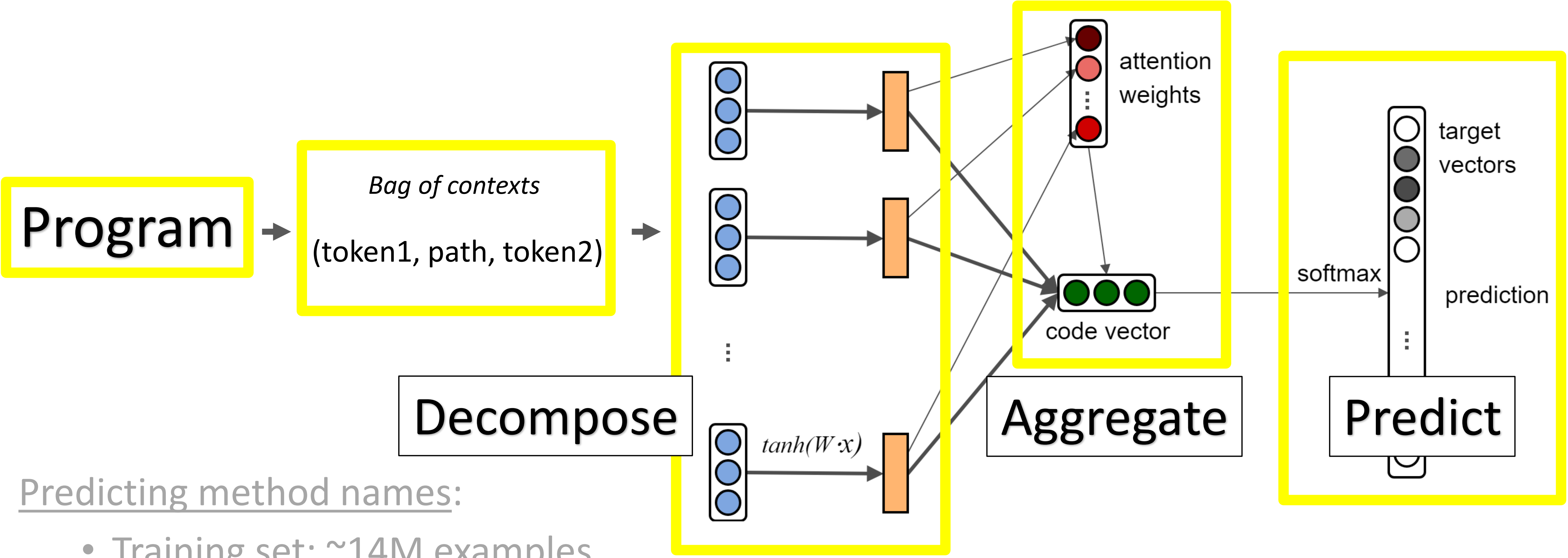
Core idea - the values of the vectors learn two distinct goals:

1. The **semantic meaning** of the path-context
2. The amount of **attention** this path-context should get



A learned weighted average!

Code2vec Architecture



Predicting method names:

- Training set: ~14M examples
- Training time: <1 day (very fast) thanks to its simplicity
- End-to-end: the entire network is trained simultaneously

```java
boolean f(Object target) {
    for (Object elem: this.elements) {
        if (elem.equals(target)) {
            return true;
        }
    }
    return false;
}
```

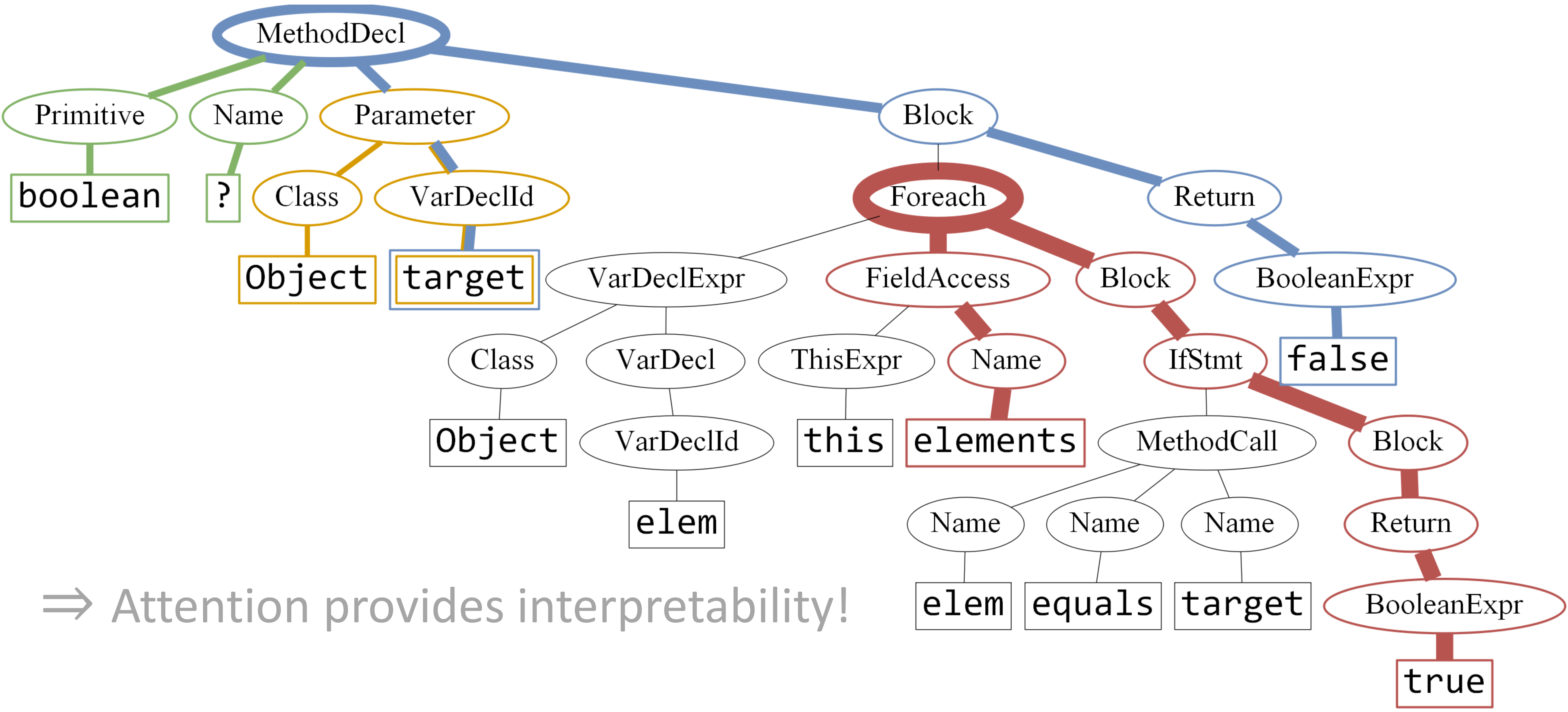
Predictions:

contains	90.93%
matches	3.54%
canHandle	1.15%
equals	0.87%
containsExact	0.77%

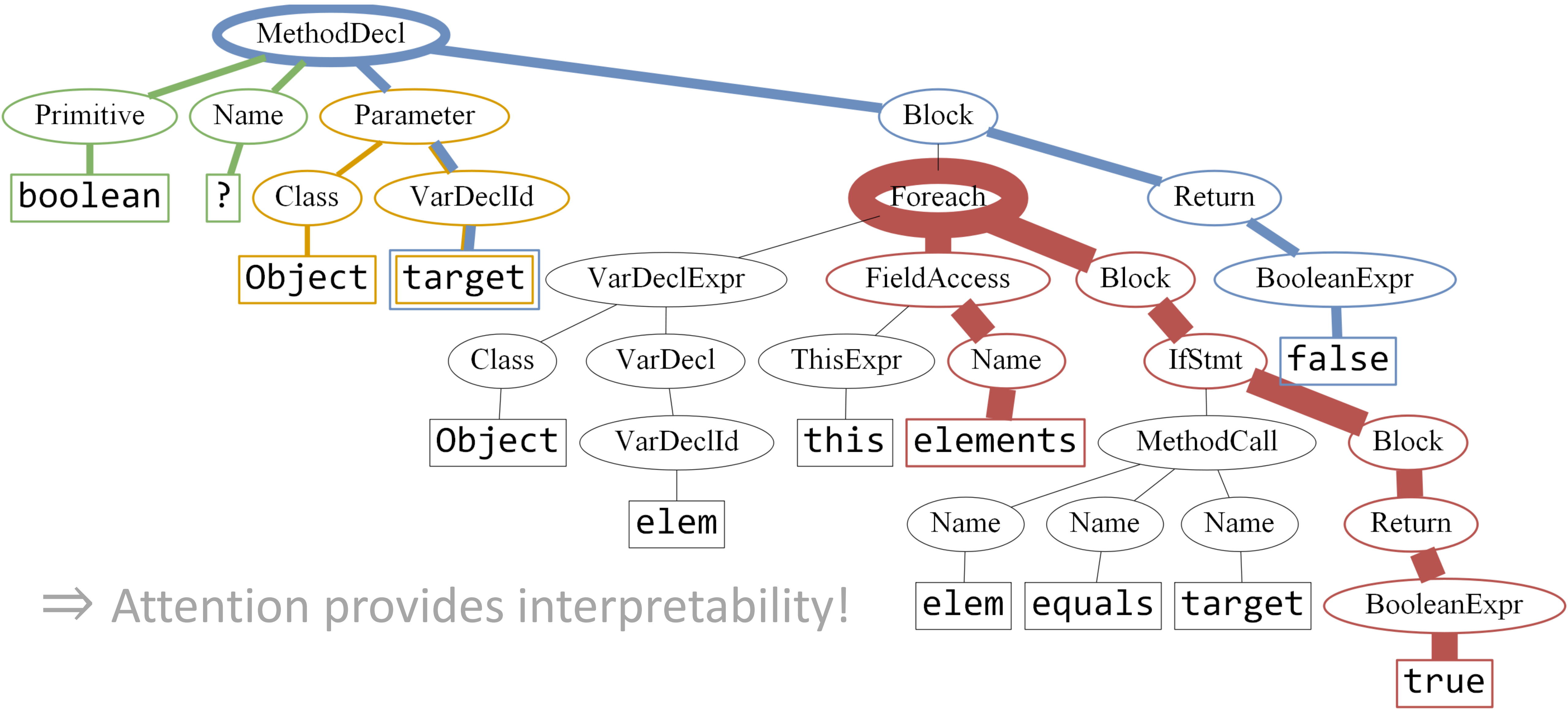
```java
Object f(int target) {
    for (Object elem: this.elements) {
        if (elem.hashCode().equals(target)) {
            return elem;
        }
    }
    return this.defaultValue;
}
```

Predictions

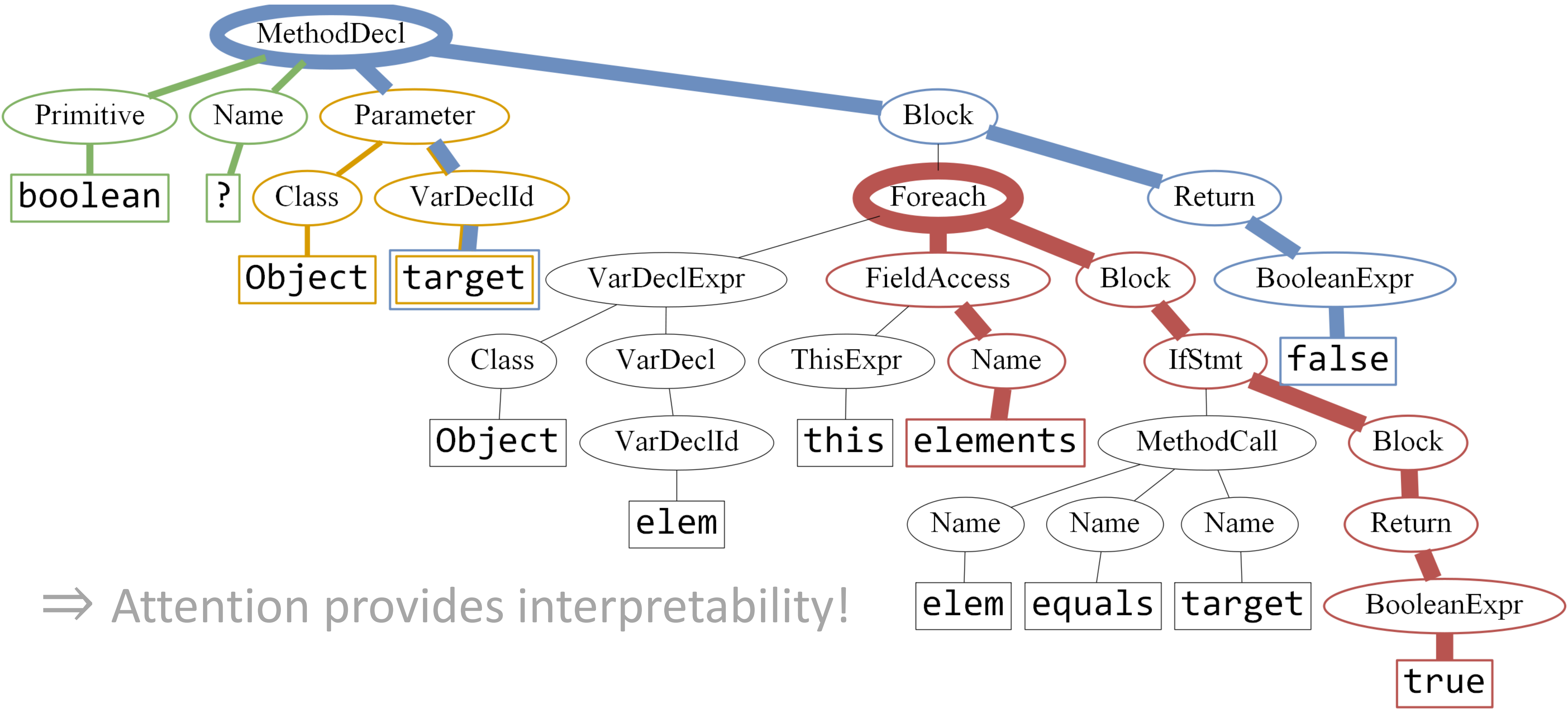
get	31.09%
getProperty	20.25%
getValue	14.34%
getElement	14.00%
getObject	6.05%



⇒ Attention provides interpretability!



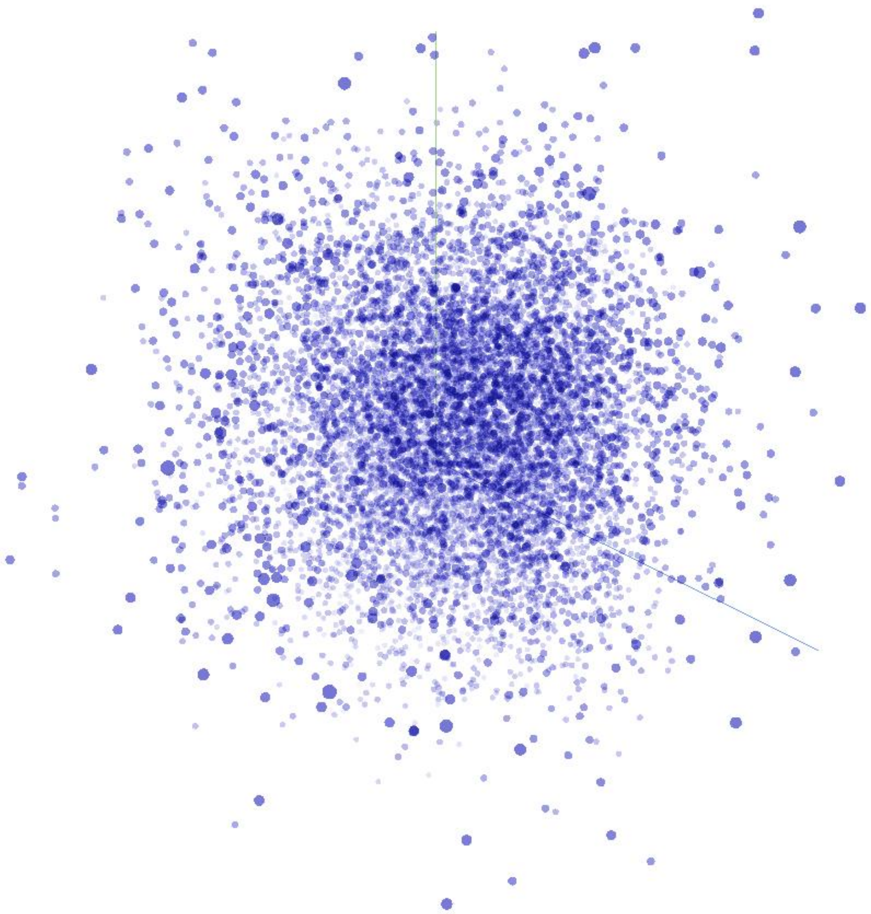
⇒ Attention provides interpretability!



⇒ Attention provides interpretability!

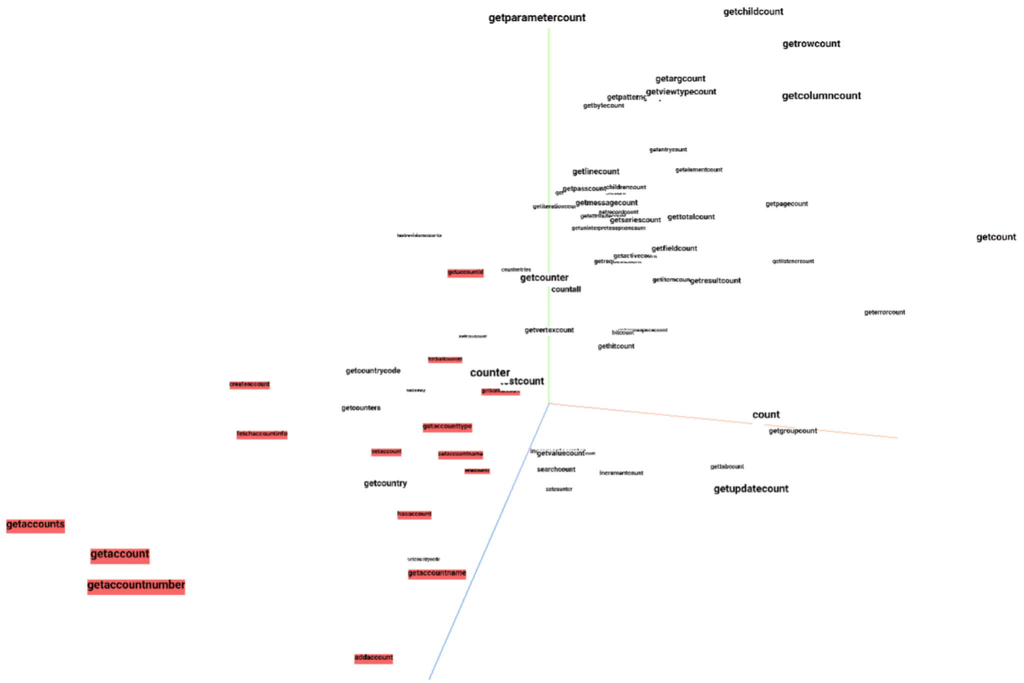
The Vector Space of Target Labels

Cosine-**similar vectors** are learned for **semantically similar** labels.



The Vector Space of Target Labels

Cosine-**similar vectors** are learned for **semantically similar** labels.



https://code2vec.org

CODE2VEC

http://code2vec.org

MOST SIMILAR

count

...is similar to:

PREDICT

getCount	70.02%
size	69.64%
index	64.99%

https://code2vec.org

CODE2VEC

http://code2vec.org

COMBINATIONS

equals and toLower

...combined, are similar to:

PREDICT

equalsIgnoreCase	78.75%
isUpperCase	75.82%
equiv	75.72%

https://code2vec.org

CODE2VEC

http://code2vec.org

ANALOGIES

receive is to download as... send

...is to:

PREDICT

upload	76.38%
delete	71.53%
connect	70.51%

CODE2VEC

Demonstration of principles shown in the paper

CODE2VEC: **LEARNING DISTRIBUTED REPRESENTATIONS OF CODE**

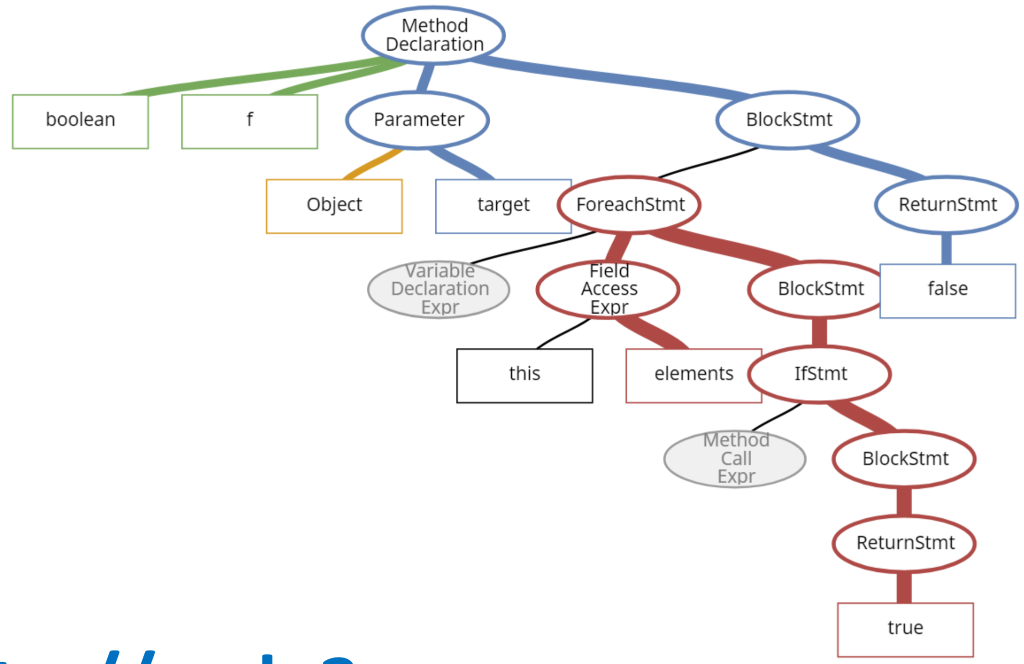
Source Paper Examples

AST

JAVA

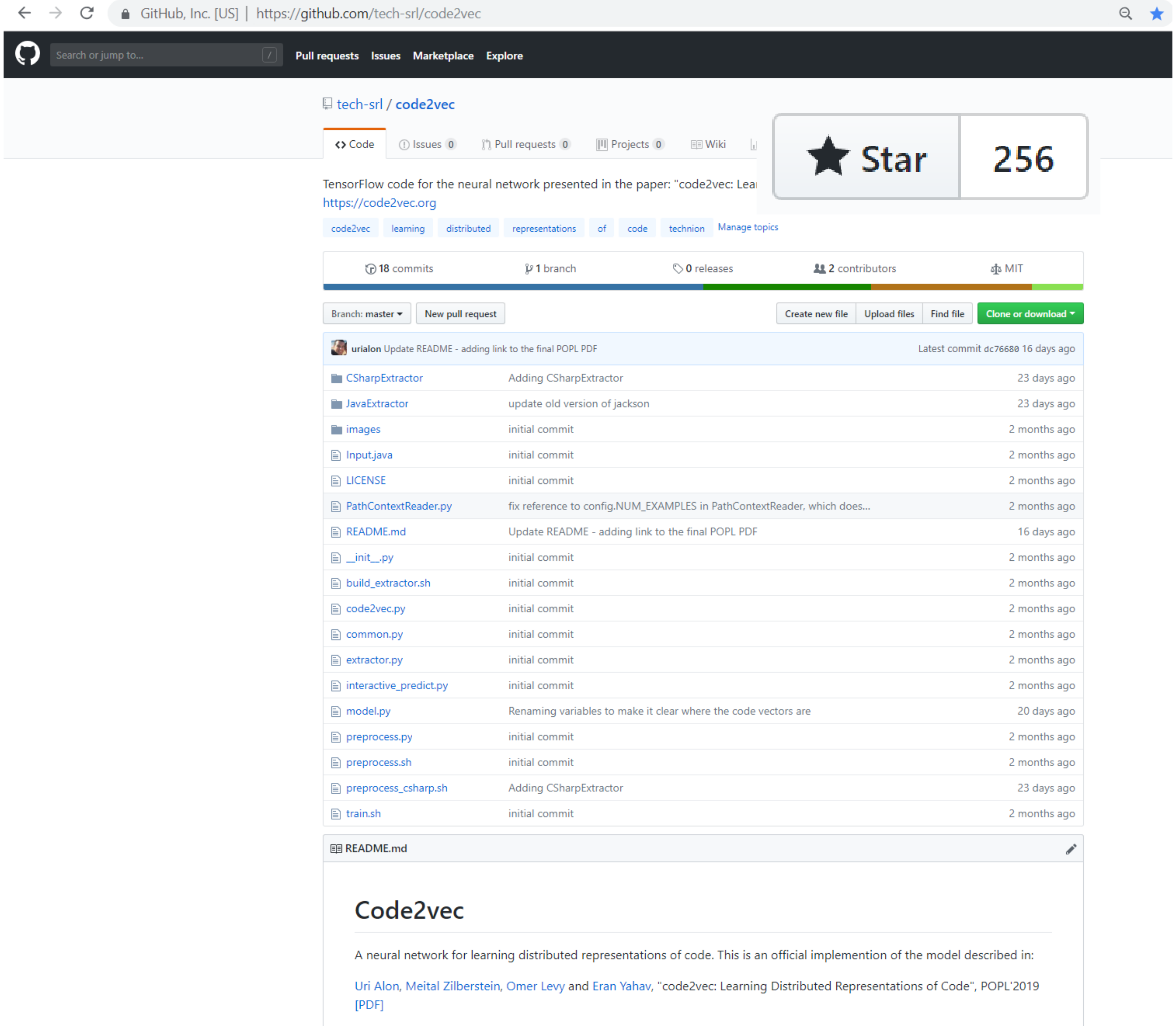
```
boolean f(Object target) {
    for (Object elem: this.elements) {
        if (elem.equals(target)) {
            return true;
        }
    }
    return false;
}
```

contains	90.93%
matches	3.54%
canHandle	1.15%
equals	0.87%
containsExact	0.77%



http://code2vec.org

EXAMPLES: 1 2 3 4 5 6 7 8 9 10 11 ?


GitHub, Inc. [US] | https://github.com/tech-srl/code2vec
Search or jump to...
Pull requests
Issues
Marketplace
Explore
tech-srl / code2vec
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Star
256
TensorFlow code for the neural network presented in the paper: "code2vec: Lea
https://code2vec.org
code2vec
learning
distributed
representations
of
code
technion
Manage topics
18 commits
1 branch
0 releases
2 contributors
MIT
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
urialon Update README - adding link to the final POPL PDF
Latest commit dc76680 16 days ago
CSharpExtractor Adding CSharpExtractor 23 days ago
JavaExtractor update old version of jackson 23 days ago
images initial commit 2 months ago
Input.java initial commit 2 months ago
LICENSE initial commit 2 months ago
PathContextReader.py fix reference to config.NUM_EXAMPLES in PathContextReader, which does... 2 months ago
README.md Update README - adding link to the final POPL PDF 16 days ago
__init__.py initial commit 2 months ago
build_extractor.sh initial commit 2 months ago
code2vec.py initial commit 2 months ago
common.py initial commit 2 months ago
extractor.py initial commit 2 months ago
interactive_predict.py initial commit 2 months ago
model.py Renaming variables to make it clear where the code vectors are 20 days ago
preprocess.py initial commit 2 months ago
preprocess.sh initial commit 2 months ago
preprocess_csharp.sh Adding CSharpExtractor 23 days ago
train.sh initial commit 2 months ago
README.md
Code2vec
A neural network for learning distributed representations of code. This is an official implemention of the model described in:
Uri Alon, Meital Zilberstein, Omer Levy and Eran Yahav, "code2vec: Learning Distributed Representations of Code", POPL'2019 [PDF]

Summary

- Core ideas in learning code snippets:
    1. Representing a code snippet as a set of syntactic paths
    2. Aggregate all paths using neural attention
- A simple and fast to train architecture
- Interpretable thanks to the attention mechanism
- The learned vectors capture interesting phenomena

http://code2vec.org

How many paths do you take from each code snippet? Taking all paths is quadratic!

- An unlimited number!
  - Since attention is simply a weighted average, it can handle an arbitrary number of path-contexts.
  - Empirically, we found that sampling 200 from each code example is sufficient. ~200 is also the average number of paths per example.
  - This number (200) can be easily increased if the dataset contained especially large pieces of code.
  - Paths that are missed due to sampling are “covered” by other paths.

Why not performing additional control flow or data flow analyses?

- These might help, but we are not sure they are necessary here. Most of the important signals are expressed in the syntax.

- Our pure-syntactic approach has the advantage of generality – the same approach can be easily applied to other languages.

- Semantic analysis is probably necessary in other tasks (for example, when the programs are binaries).

How robust are the results for variable renaming?

- As any machine learning model, confusing or adversarial examples can mislead our model.

- Since the network was trained on “well-named” examples from top-starred GitHub projects, it does perform worse without names.

- We are exploring similar approaches for obfuscated code as part of ongoing research.

Do you keep vectors for *all* paths and tokens?

- Almost all!
  - Limiting to the most occurring 1M tokens, 1M paths, and 300k target labels.

- Each token and path vectors has 128 elements of 4 bytes (float32)
- Each target vector has 384 elements of 4 bytes
- Attention vector has 384 elements
- Fully connected layer is a matrix of size 384×384

- Total size: $\underbrace{128 \cdot 4}_{vector} \cdot \left(\underbrace{1M + 1M}_{\substack{token+path \\ vocab \\ sizes}} \right) + \underbrace{384 \cdot 4}_{vector} \cdot \underbrace{300k}_{\substack{target \\ vocab\ size}} + \underbrace{384}_{attention} + \underbrace{384^2}_{\substack{fully- \\ connected}} \approx \mathbf{1.5}\ \boldsymbol{GB}$

- Standard GPU memory size: $\mathbf{12}\ \boldsymbol{GB}$

Did you try Gated Graph NNs (Allamanis et al., ICLR'2018)?

- GGNNs were applied to a simpler task of Var-Misuse.
  - Their code is not fully available.

- Two conceptual advantages of **code2vec** over **GGNNs**:
  1. **Much faster to train** - thus practically easier to leverage huge corpora (our dataset is orders of magnitude larger).
  2. **Our model is purely syntactic** - the same algorithm can work for every programming language. In GGNNs, the edges in the graph are analyses like "ComputedFrom" and "LastWrite", that need to be re-implemented for different languages.

Can a non-neural model solve the same task?

- Yes, and pretty well (PLDI'2018).
  - But not as good as a neural model.

- Main advantages of using a neural network:
  1. Much **better generalization** (Section 5 in the paper)
  2. Our neural network can **produce a vector**, which can be fed to a variety of other (neural and non-neural) ML models and tasks.