Code2vec: Learning Distributed Representations of Code

URI ALON, Technion, Israel
MEITAL ZILBERSTEIN, Technion, Israel
OMER LEVY, Facebook AI Research, USA
ERAN YAHAV, Technion, Israel

Source code at https://github.com/tech-srl/code2vec

Proc. ACM Program. Lang., Vol. 3, No. POPL, Article 40. Publication date: January 2019.

https://code2vec.org

Overview

- Motivating use-case: Present an extreme summarization of source code snippets into short, descriptive function name.
- Architecture: Attention model over path-contexts
- Design Choices: Focus on syntax level
- Evaluation: Dependent on variable naming
- Discuss Limitations: Tendency to pick up on coding convention over learned semantic meaning
- Online Demo: as time permits

Motivating Example: Sematic Method Naming

Try to predict attribute a meaningful name to methods based on the body

Motivating Example: Sematic Method Naming

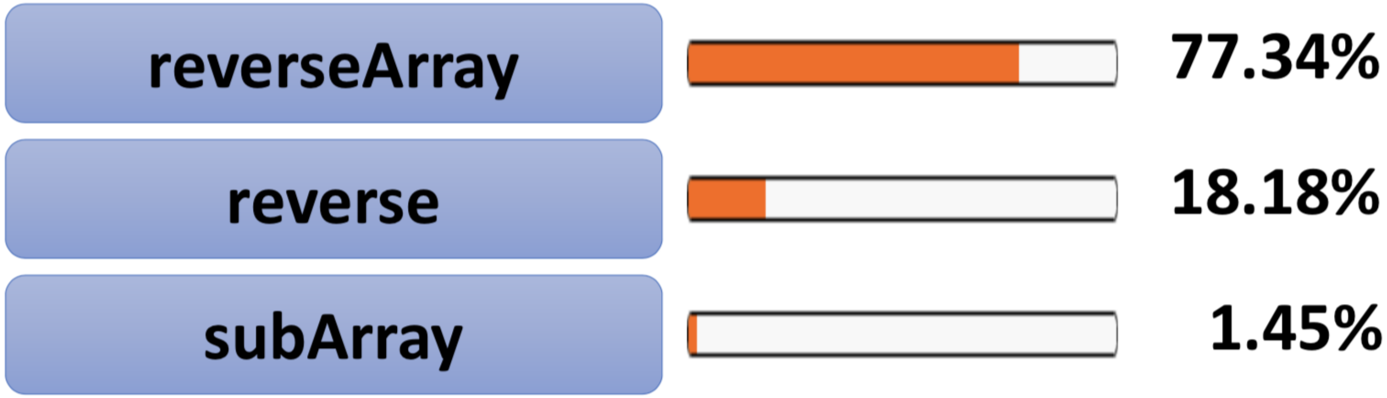
Try to predict attribute a meaningful name to methods based on the body

```
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;
}
```

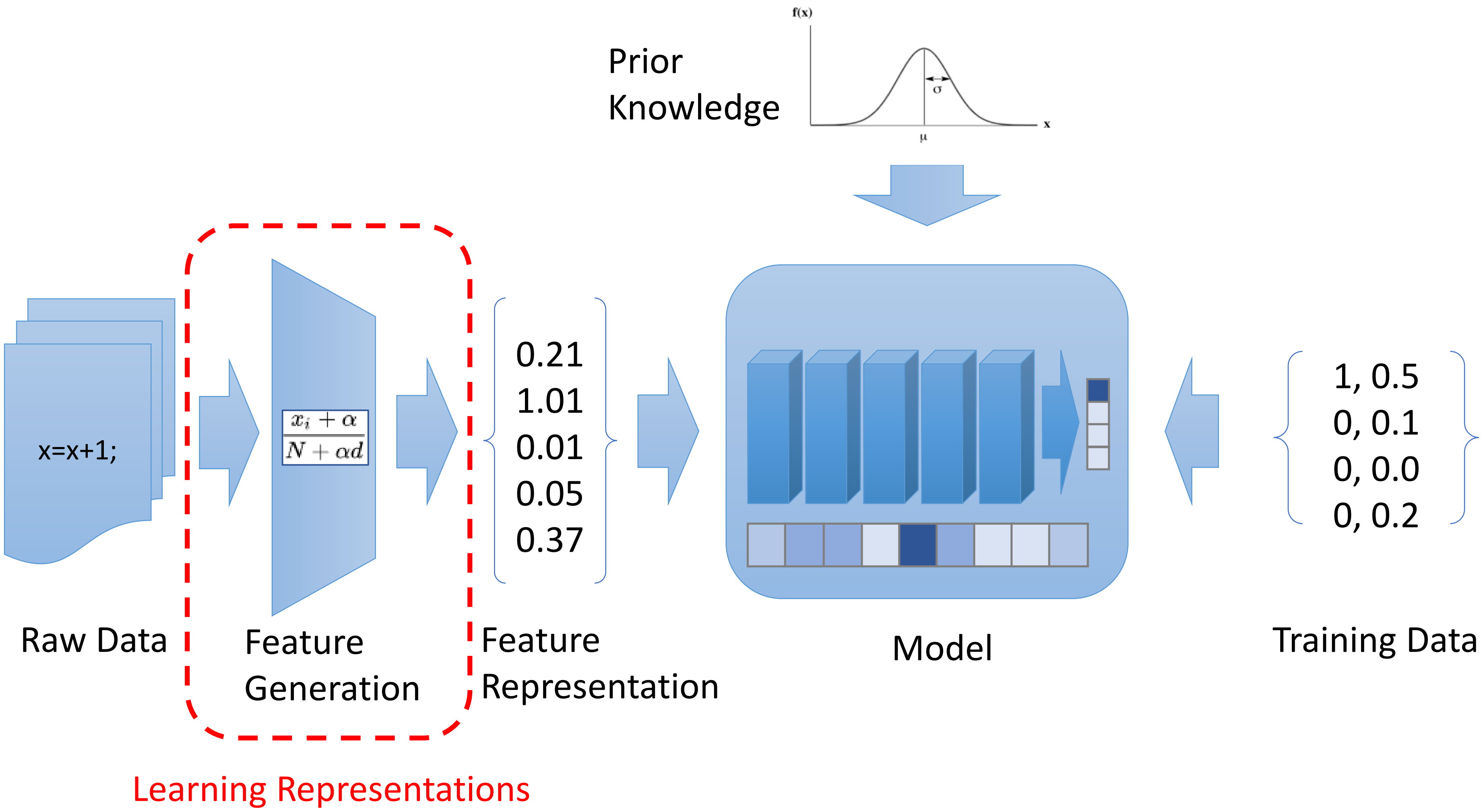
Motivating Example: Sematic Method Naming

Try to predict attribute a meaningful name to methods based on the body

```
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;
}
```



General Task: Representation Learning



History of Static Code Representation

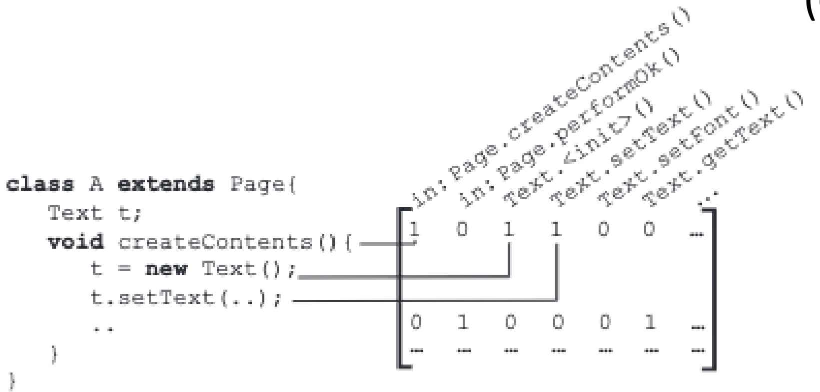
Exact Representation → **Constructed Features** → **Deep Learning Features**

Static Rule Inference + Checking

```
3: void foo() {
4:     lock(1);       // Enter critical section
5:     a = a + b;     // MAY: a,b protected by 1
6:     unlock(1);     // Exit critical section
7:     b = b + 1;     // MUST: b not protected by 1
8: }
```

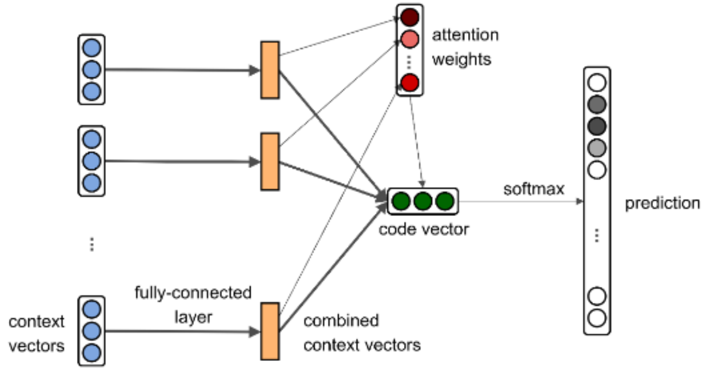
Engler, Dawson, et al. "Bugs as deviant behavior: A general approach to inferring errors in systems code." *ACM SIGOPS Operating Systems Review*. Vol. 35. No. 5. ACM, 2001.

Binary Feature Vectors, N-Grams



Bruch, Marcel, Martin Monperrus, and Mira Mezini. "Learning from examples to improve code completion systems." *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009.

"Semantic Space" Vector Embeddings (code2vec)

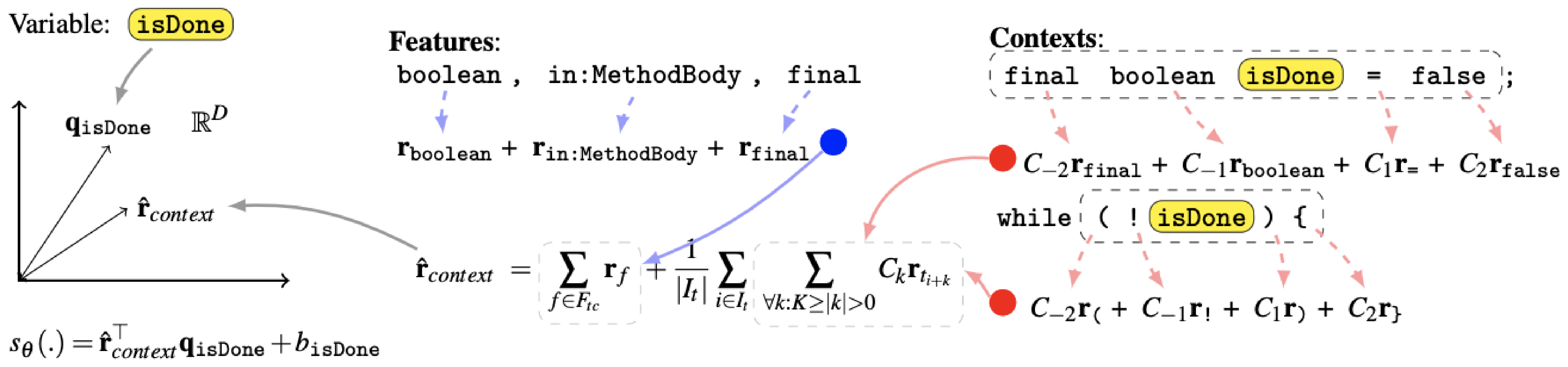


Alon, Uri, et al. "code2vec: Learning distributed representations of code." *Proceedings of the ACM on Programming Languages* 3.POPL (2019): 40.

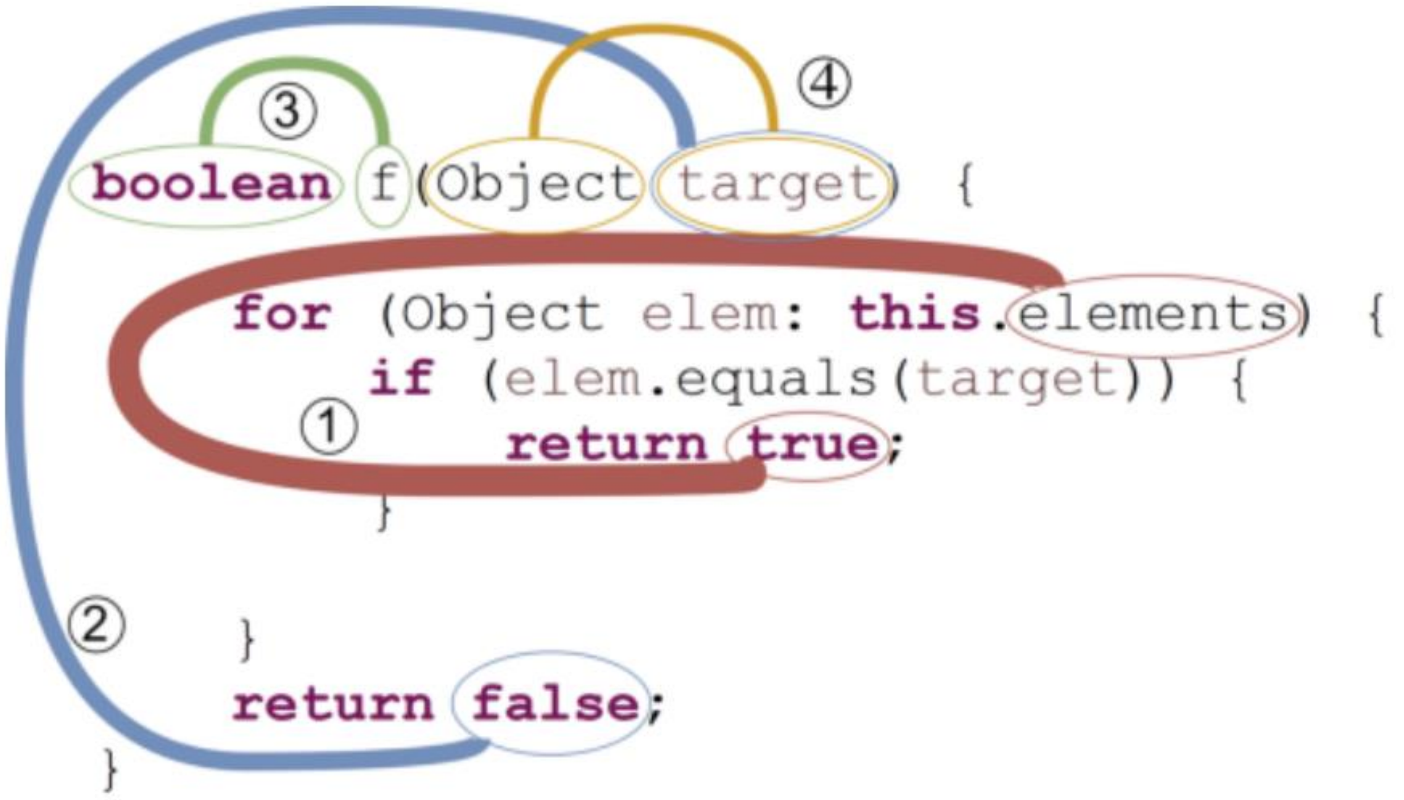
Comparison with inst2vec

- ‘How far a syntactic-only approach can go’
- Purely syntactic
  - no control flow/data flow information
- Scales more effectively
  - 1K method per second training on over 12M methods
- Can interpret how predictions are reached
  - Using attention

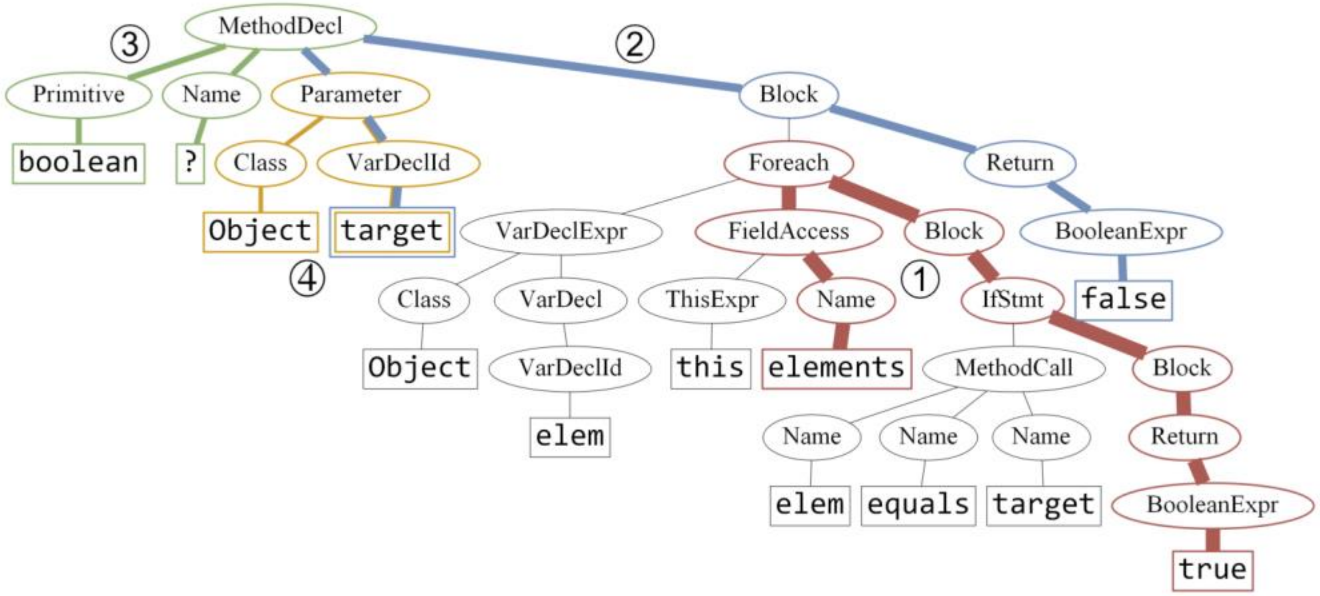
A naïve approach: N-gram model



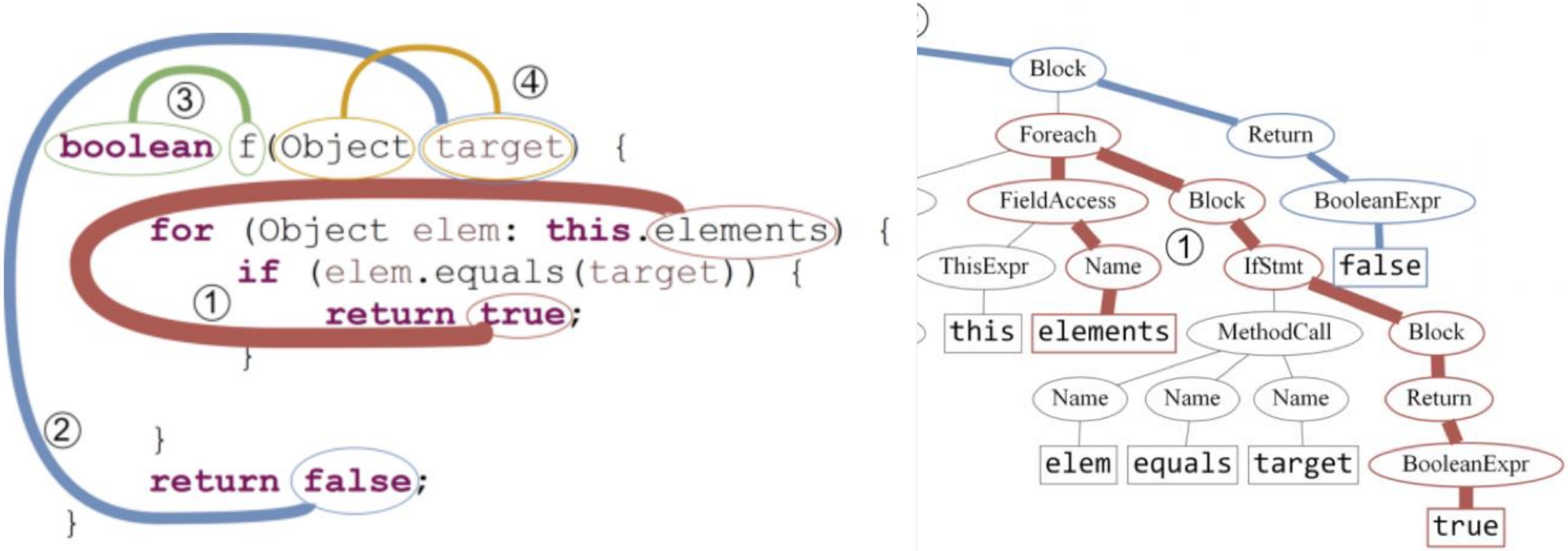
Path Context: Example – Contains function



Path Context: Parse into Abstract Syntax Tree

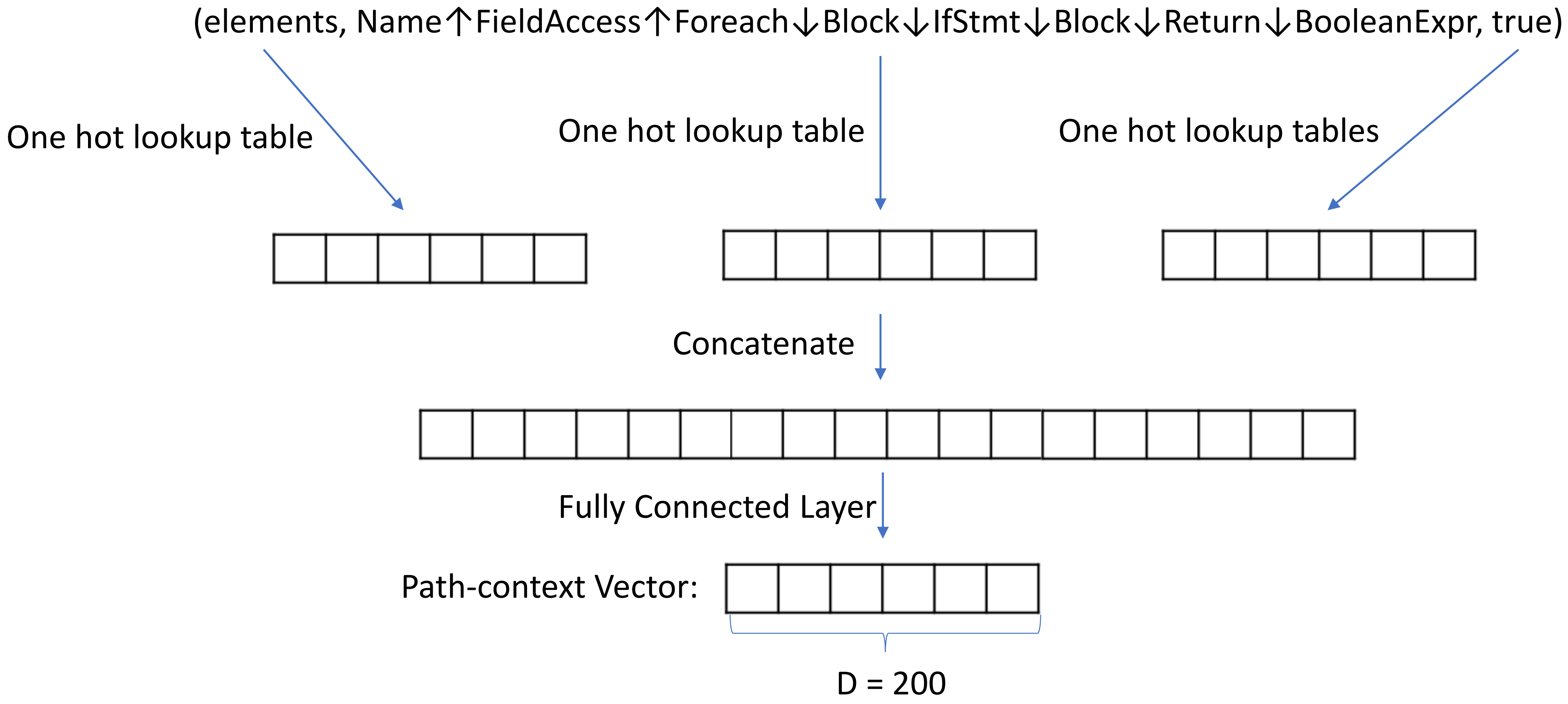


Path Context: representing the path

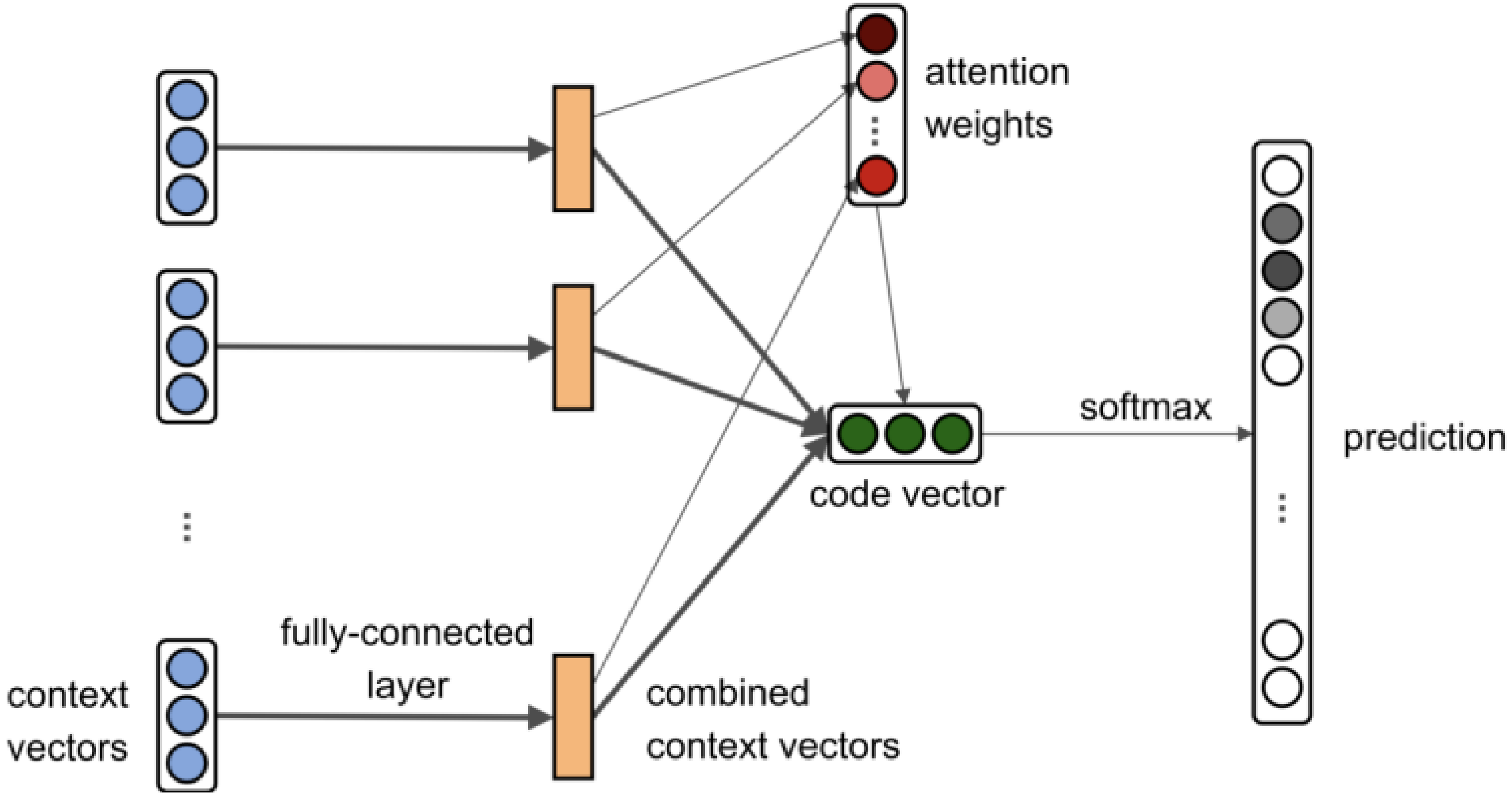


(elements, Name↑FieldAccess↑Foreach↓Block↓IfStmt↓Block↓Return↓BooleanExpr, true)

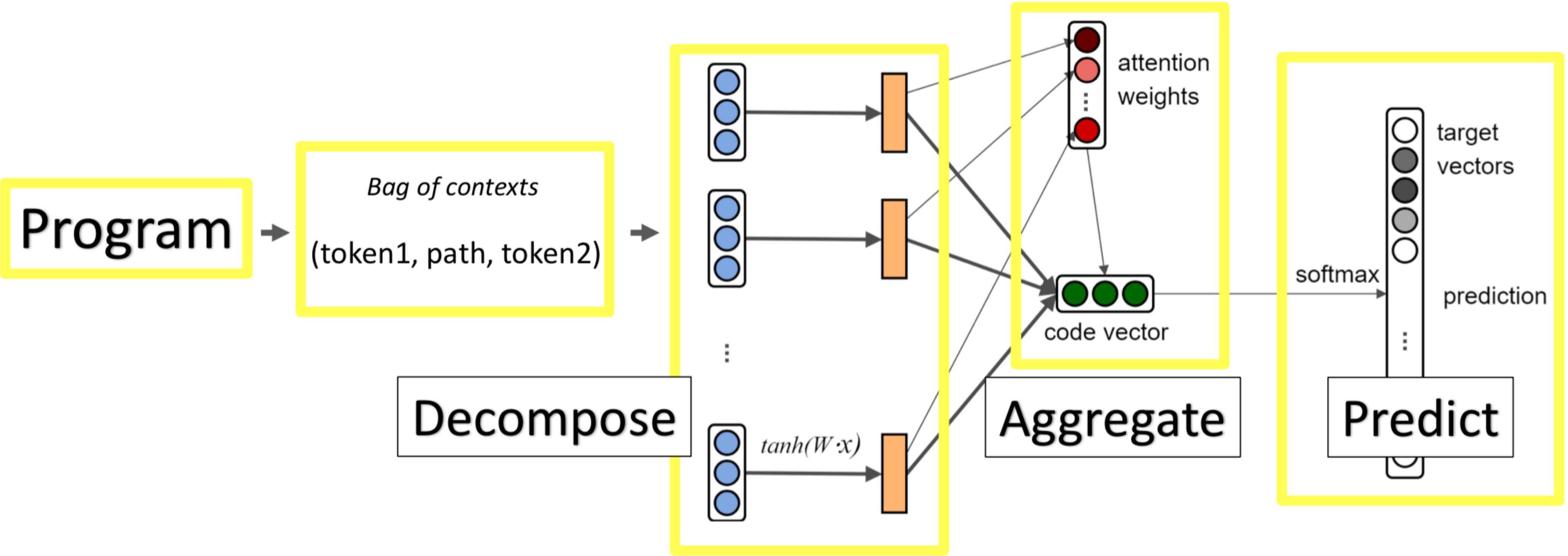
Path-context vector



Neural Network Architecture



Code2vec Architecture



Design Choices

- Bag of contexts
  - Existence not order
- Syntax-only
- Large corpus, simple model
  - “95% of the paths in the test set were already seen in the training set”
  - 1000 methods per second
  - 1.5 days to completely train a model

Table 2. Size of data used in the experimental evaluation.

	Number of methods	Number of files	Size (GB)
Training	12,636,998	1,712,819	30
Validation	371,364	50,000	0.9
Test	368,445	50,000	0.9
Sampled Test	7,454	1,000	0.04

Evaluation

- Prediction of names compared to recent approaches
- Compare with no attention or hard attention
- Evaluate the relative contribution from the components of the context vector
- Interpreting code vectors

Evaluation Metric

- Case-insensitive Sub-token Matching
- countLines == linesCount
- countLines vs count
    - Full precision -> no false positives
    - Low recall -> has false negatives
- countLines vs countBlankLines
    - Low precision -> has false positives
    - Full recall -> has no false negatives

Prediction evaluation

- Note exactly fair cause the CNN and LSTM only get to see a token stream.

Table 3. Evaluation comparison between our model and previous works.

Model	Sampled Test Set Precision	Recall	F1	Full Test Set Precision	Recall	F1	prediction rate (examples / sec)
CNN+Attention [Allamanis et al. 2016]	47.3	29.4	33.9	-	-	-	0.1
LSTM+Attention [Iyer et al. 2016]	27.5	21.5	24.1	33.7	22.0	26.6	5
Paths+CRFs [Alon et al. 2018]	-	-	-	53.6	46.6	49.9	10
PathAttention (this work)	**63.3**	**56.2**	**59.5**	**63.1**	**54.4**	**58.4**	**1000**

Evaluation of Attention

- No-attention – unweighted average
- Hard-attention – only use highest ranked context
- Train-soft, predict-hard – only marginal improvement
- Element-wise soft attention – don't compress with fully connected layer
    - Better performance but hard to interpret

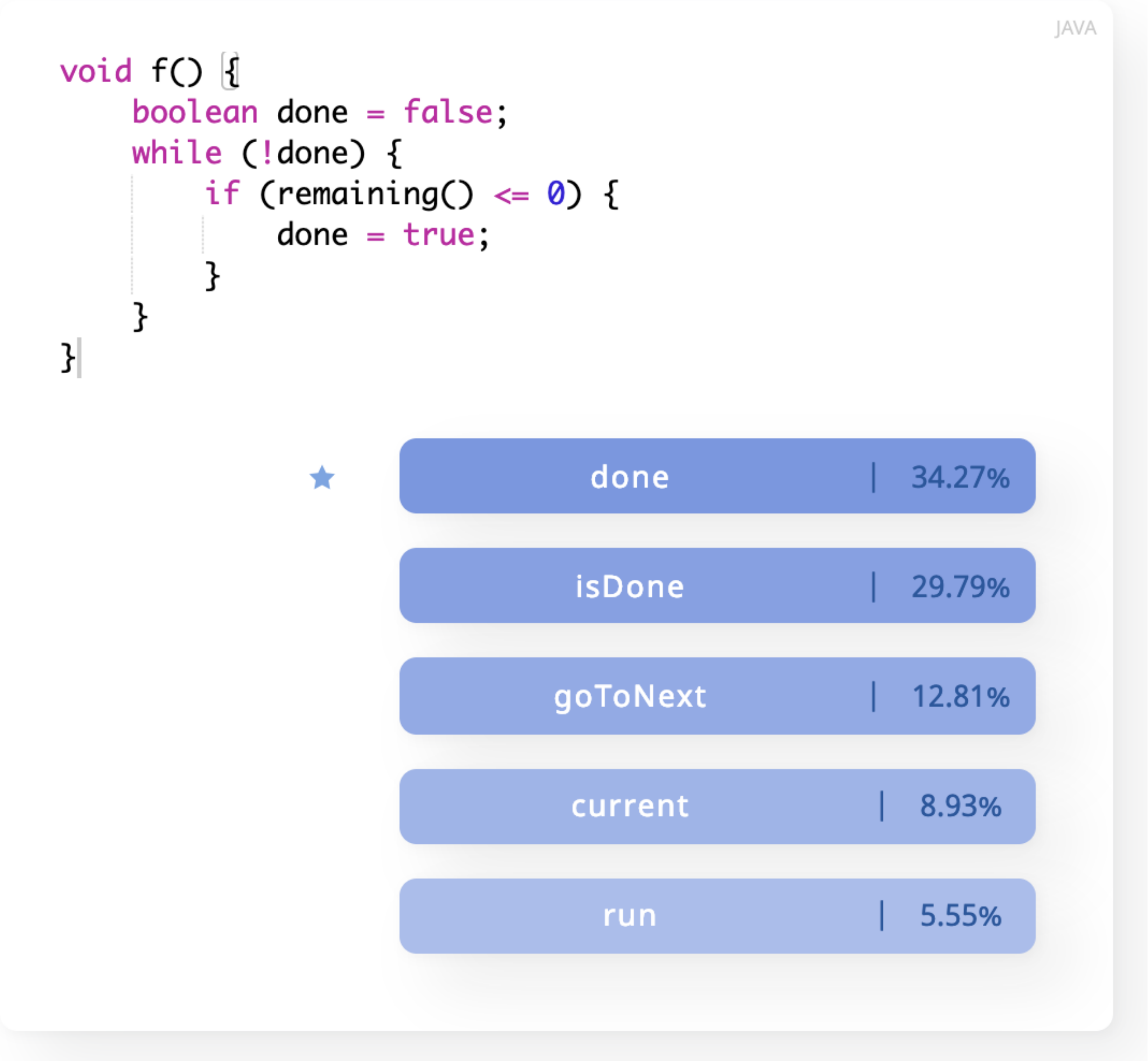
Table 4. Comparison of model designs.

Model Design	Precision	Recall	F1
No-attention	54.4	45.3	49.4
Hard attention	42.1	35.4	38.5
Train-soft, predict-hard	52.7	45.9	49.1
Soft attention	**63.1**	**54.4**	**58.4**
Element-wise soft attention	**63.7**	**55.4**	**59.3**

Data Ablation Study: evaluating components

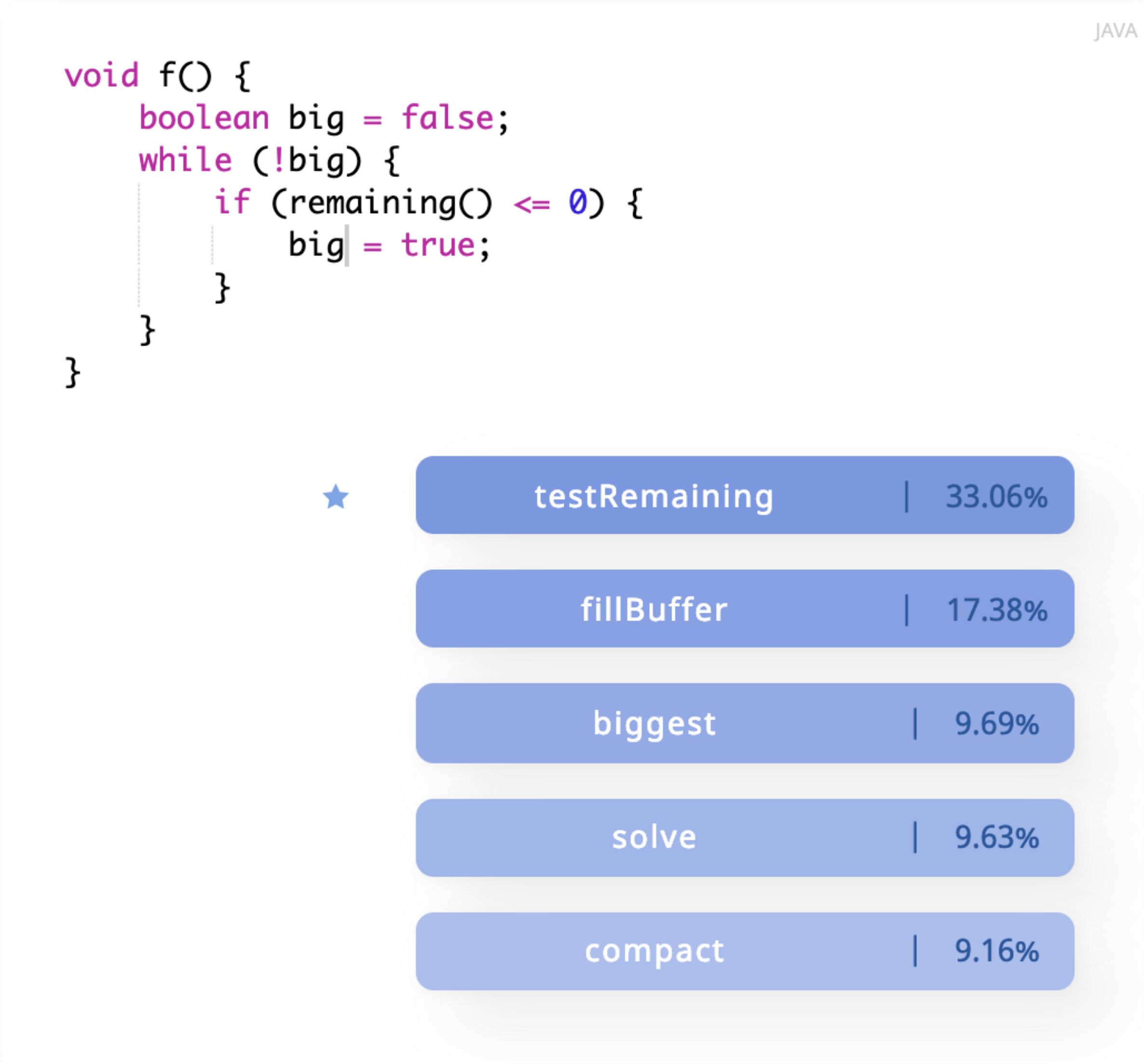
- Note performance of ‘No-value’ is significantly worse suggesting not robust with respect to changing variable names.
- The addition of “only-values” and ”no-values” is less than ”Full” suggesting some kind of synergy

Path-context input		Precision	Recall	F1
Full:	$\langle x_s, p, x_t \rangle$	63.1	54.4	58.4
Only-values:	$\langle x_s, \_\_, x_t \rangle$	44.9	37.1	40.6
No-values:	$\langle \_\_, p, \_\_ \rangle$	12.0	12.6	12.3
Value-path:	$\langle x_s, p, \_\_ \rangle$	31.5	30.1	30.7
One-value:	$\langle x_s, \_\_, \_\_ \rangle$	10.6	10.4	10.7



Variable Names

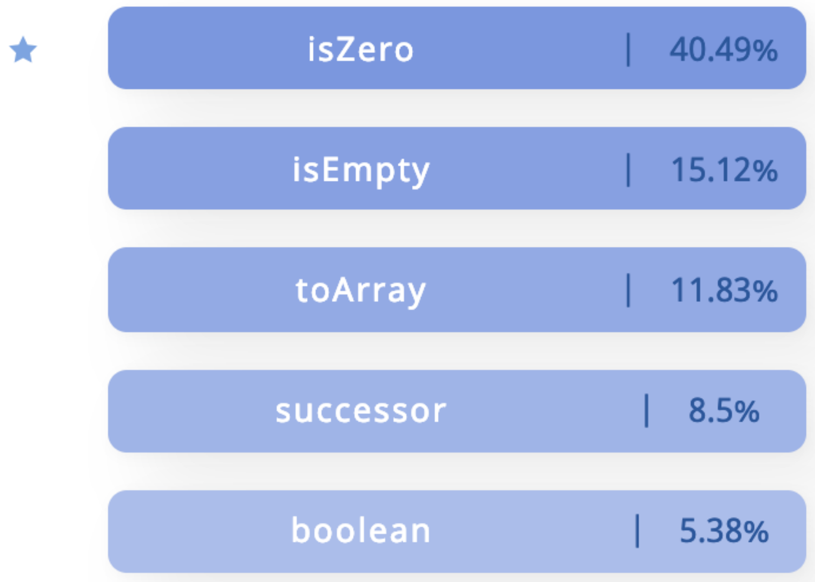
- Not that surprising given some of the examples of methods and the absence of semantic/execution information.



Variable Names

- Not that surprising given some of the examples of methods and the absence of semantic/execution information.
- Biggest indicates dependency on names

```java
void f() {
    boolean b = false;
    while (!b) {
        if (a() <=  0 ) {
            b = true;
        }
    }
}
```



Variable Names

- Not that surprising given some of the examples of methods and the absence of semantic/execution information.
- Biggest indicates dependency on names
- Completely obfuscated is basically hopeless

Interpretation: Analogies

Table 7. Semantic analogies between method names.

A :	B	C :	D
open :	connect	close :	disconnect
key :	keys	value :	values
lower :	toLowerCase	upper :	toUpperCase
down :	onMouseDown	up :	onMouseUp
warning :	getWarningCount	error :	getErrorCount
value :	containsValue	key :	containsKey
start :	activate	end :	deactivate
receive :	download	send :	upload

Limitations

- Closed label vocabulary:
  - Even though the labels for prediction can be composed rare names can’t be predicted.
  - Eg: imageFormatExceptionShouldProduceNotSuccessOperationResultWithMessage
- Sparsity
  - Variable names newArray and oldArray are treated as separate terminals.
  - AST paths that differ by a single node are considered completely different.
- Dependency on Variable Names
  - Potentially remedied by pipelining an upstream de-obfuscator tool

Questions?

- https://code2vec.org

Other prediction tasks:

Table 2. Accuracy comparison for variable name prediction, method name prediction, and full type prediction using CRFs.

Task	Previous works		AST Paths (this work)	Params (length/width)
Variable name prediction				
JavaScript	24.9% (no-paths)	60.0% (UnuglifyJS)	**67.3%**	7/3
Java	23.7% (rule-based)	50.1% (CRFs+4-grams)	**58.2%**	6/3
Python	35.2% (no-paths)		**56.7%** (top-7)	7/4
C#			**56.1%**	7/4
Method name prediction				
JavaScript	44.1% (no-paths)		**53.1%**	12/4
Java	16.5%, F1: 33.9 (Allamanis et al. [7])		**47.3%**, F1: **49.9**	6/2
Python	41.6% (no-paths)		**51.1%** (top-7)	10/6
Full type prediction				
Java	24.1% (naïve baseline)		**69.1%**	4/1

A General Path-Based Representation for Predicting Program Properties PLDI'18, June 18–22, 2018, Philadelphia, PA, USA

Variable Name Prediction

Stripped Names

```
function f(a, b, c) {
  b.open('GET', a, false);
  b.send(c);
}
```

AST Paths + CRFs

```
function f(url, request, callback) {
  request.open('GET', url, false);
  request.send(callback);
}
```

nice2predict.org

```
function f(source, req, n) {
  req.open("GET", source, false);
  req.send(n);
}
```