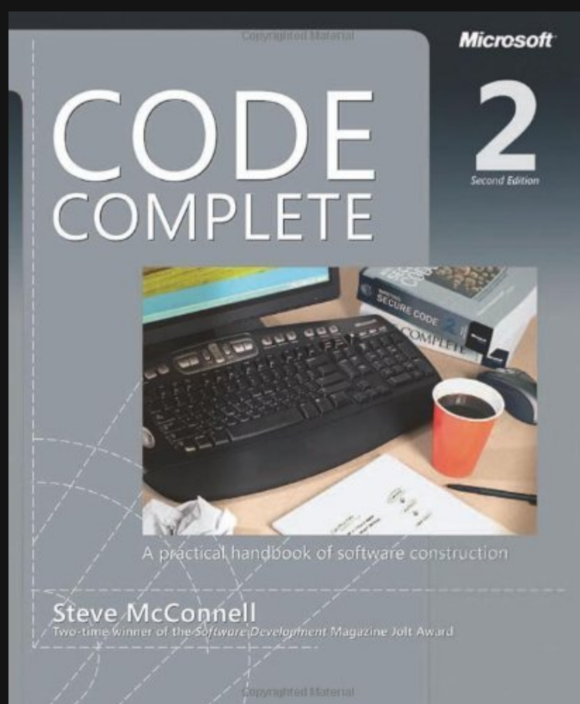
DeepBugs: A Learning Approach to Name-based Bug Detection

Michael Pradel and Koushik Sen

Presented at OOPSLA 2018

Software has bugs

Software has bugs



0.5-25/KLoC in delivered software

Static Bug Detection

- Lightweight static analysis
- General framework & set of checkers for specific bug patterns

Infer

Google
Error Prone

SpotBugs

The Problem

- **Existing bug detectors miss most bugs (see our ASE'18 paper)**

- **Main reasons:**
  - Bugs are domain-specific
  - Bug detectors cover only a small fraction of all bug patterns

Traditional Approach

How to create a new bug detector?

Human expert

Time-consuming process

Program analysis

Traditional Approach

How to create a new bug detector?

Human expert

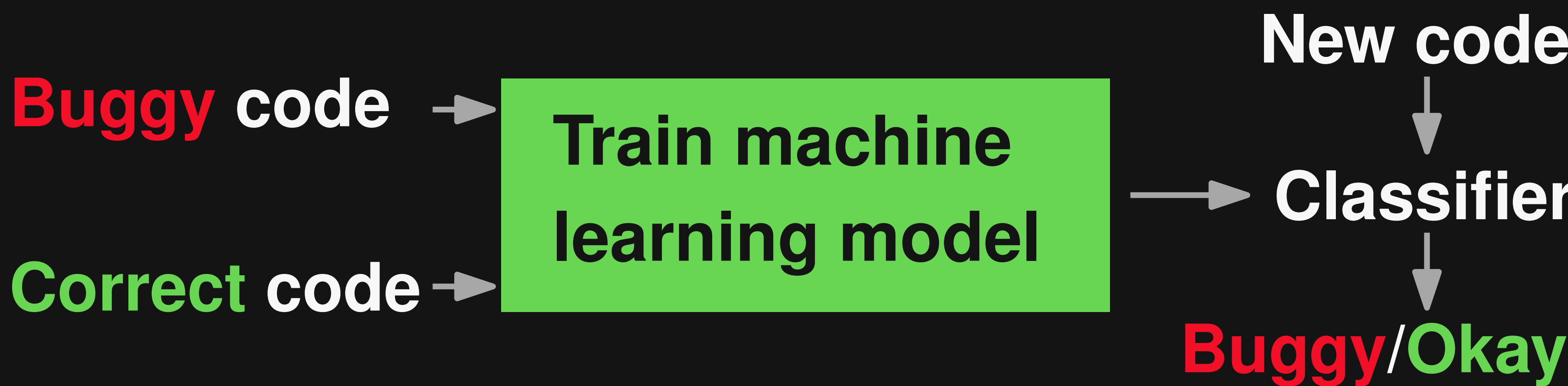
Time-consuming process

Program analysis

- **Heuristics**, e.g., to avoid spurious warnings
- Carefully **tuned algorithms**, e.g., to ensure scalability

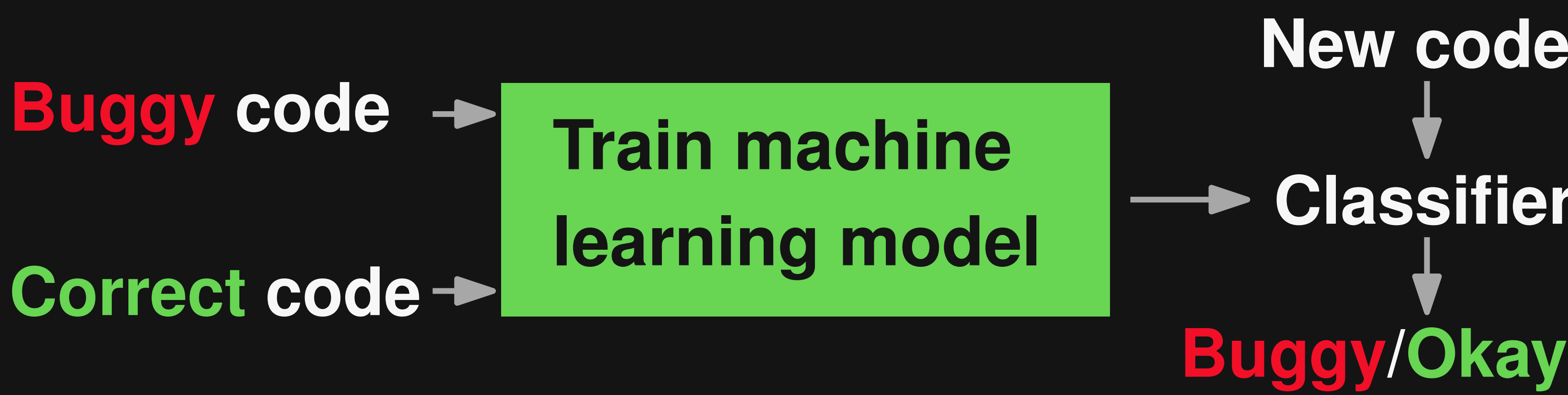
Learning to Find Bugs

Train a model to distinguish correct from buggy code



Learning to Find Bugs

Train a model to distinguish correct from buggy code

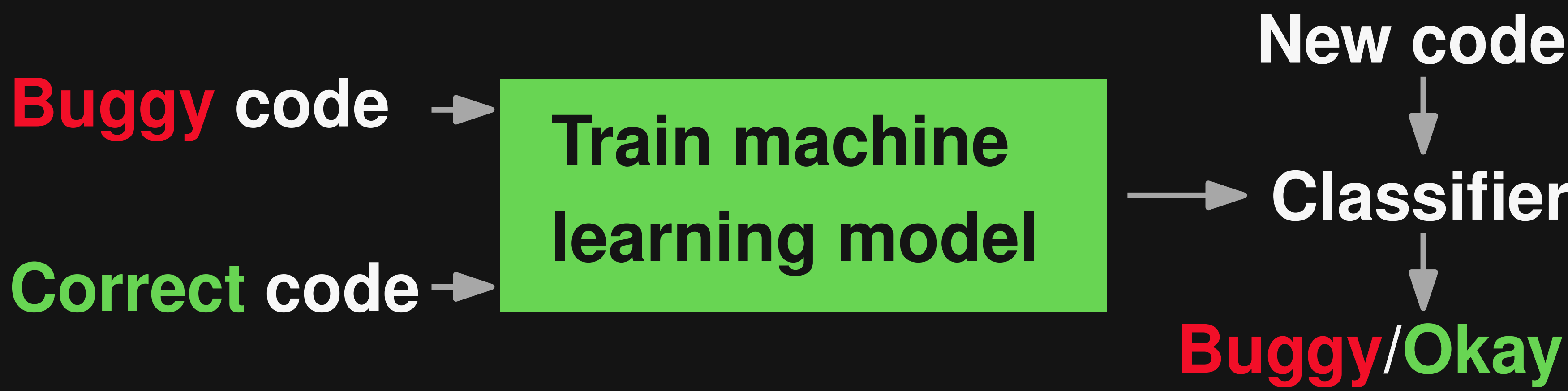


How to get training data?

- Here: Insert artificial bugs via simple program transformations
- Other options, e.g., from version histories

Learning to Find Bugs

Train a model to distinguish correct from buggy code



How to represent code?

- Here: Embeddings of natural language elements in code
- Other options, e.g., token-based, graph-based

Benefits of Learning Bug Detectors

Simplifies the problem

- Before: Writing a program analysis
- Now: Providing examples of buggy and correct code

Catches otherwise missed bugs

- Learns conventions from corpora of existing code
- ML can handle natural language in code, which expresses domain-specific knowledge

Name-related Bugs

What’s wrong with this code?

```
function setPoint(x, y) { ... }

var x_dim = 23;
var y_dim = 5;
setPoint(y_dim, x_dim);
```

Name-related Bugs

What's wrong with this code?

```
function setPoint(x, y) { ... }

var x_dim = 23;
var y_dim = 5;
setPoint(y_dim, x_dim);
```

Incorrect order of arguments

Name-related Bugs (2)

What's wrong with that code?

```
for (j = 0; j < params; j++) {
  if (params[j] == paramVal) {
    ...
  }
}
```

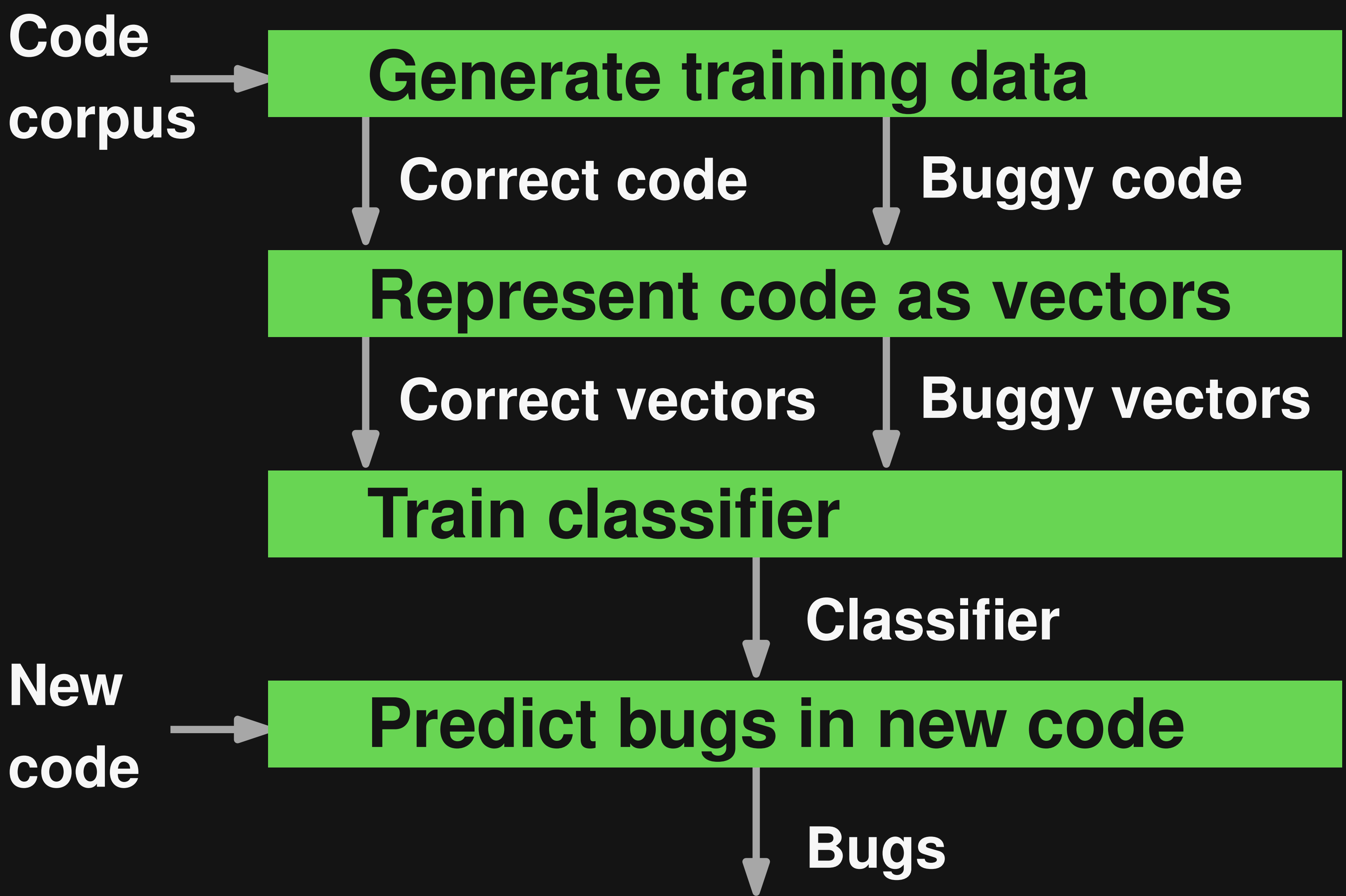
Name-related Bugs (2)

What's wrong with that code?

```
for (j = 0; j < params; j++) {
  if (params[j] == paramVal) {
    ...
  }
}
```

Should be `params.length`

Overview of DeepBugs



Generating Training Data

Simple **code transformations** to **inject artifical bugs** into given corpus

Generating Training Data

Simple **code transformations** to **inject artifical bugs** into given corpus

1) Swapped arguments

```
setPoint(x, y)  →  setPoint(y, x)
```

Generating Training Data

Simple code transformations to inject artifical bugs into given corpus

2) Wrong binary operator

`i <= length` → `i % length`

Randomly selected operator

Generating Training Data

Simple code transformations to inject artifical bugs into given corpus

3) Wrong binary operand

```
bits << 2        →        bits << next
```

Randomly selected operand that occurs in same file

Representing Code as Vectors

- **Insight: Natural language in identifiers conveys semantics of code**

- **Compute word embeddings of identifier names**
  - Train Word2Vec* on corpus of code
  - Tokens ≈ words

Efficient Estimation of Word Representations in Vector Space
(Mikolov et al., 2013)

Word Embeddings

- **Known problem in natural language processing**
- **Word embeddings**
  - Continuous vector representation for each word
  - Similar words have similar vectors

Word2Vec

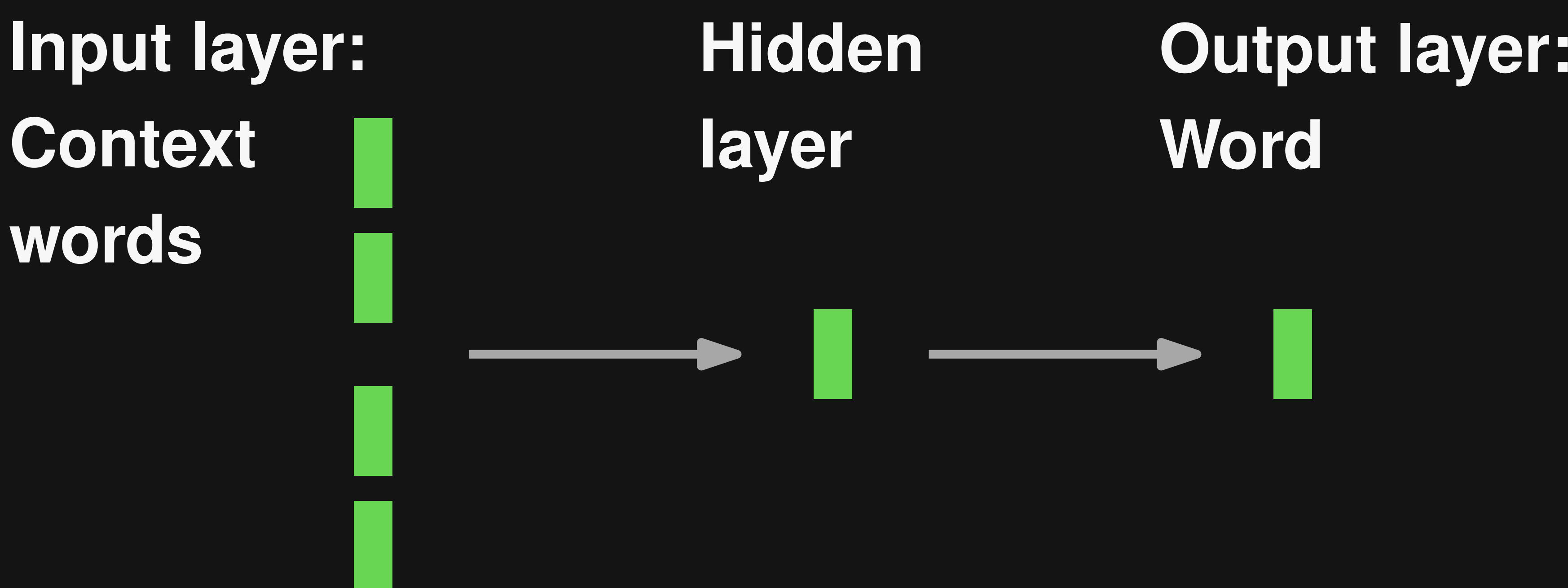
Learn embeddings from corpus of text

- ”You shall know a word by the company it keeps”
- Context: Surrounding words in sentences

Word2Vec

Learn embeddings from corpus of text

- "You shall know a word by the company it keeps"
- Context: Surrounding words in sentences



Word2Vec for Source Code

Natural language

- Sentences
- Words

Programming language

- Program
- Tokens

Word2Vec for Source Code

Natural language

- Sentences
- Words

Programming language

- Program
- Tokens

```
function setPoint(x, y) { ... }

var x_dim = 23;
var y_dim = 5;
setPoint(y_dim, x_dim);
```

Word2Vec for Source Code

Natural language		Programming language
▪ Sentences	→	▪ Program
▪ Words	→	▪ Tokens

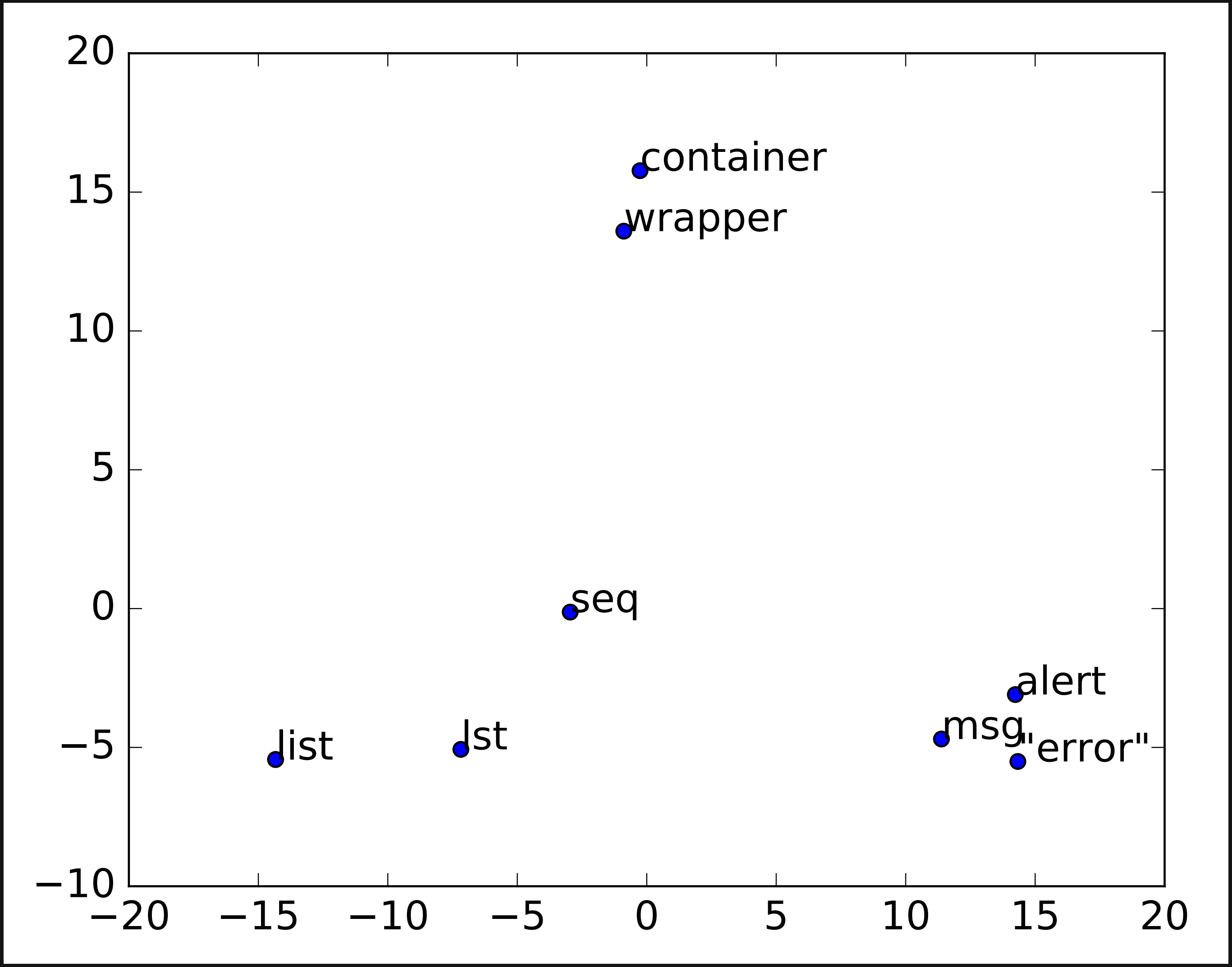
```
function setPoint(x, y) { ... }

var x_dim =
var y_dim =
setPoint(y_dim, x_dim);
```

Context of `x`:

`function - setPoint - ( - , - y - )`

Example: Embeddings

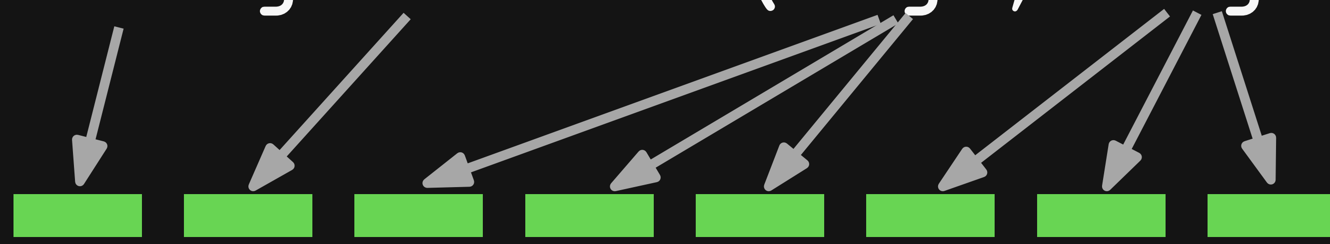


Code Snippets as Vectors

Concatenate embeddings of names in code snippet

1) Swapped arguments

```
someObj.someFun(arg1, arg2)
```

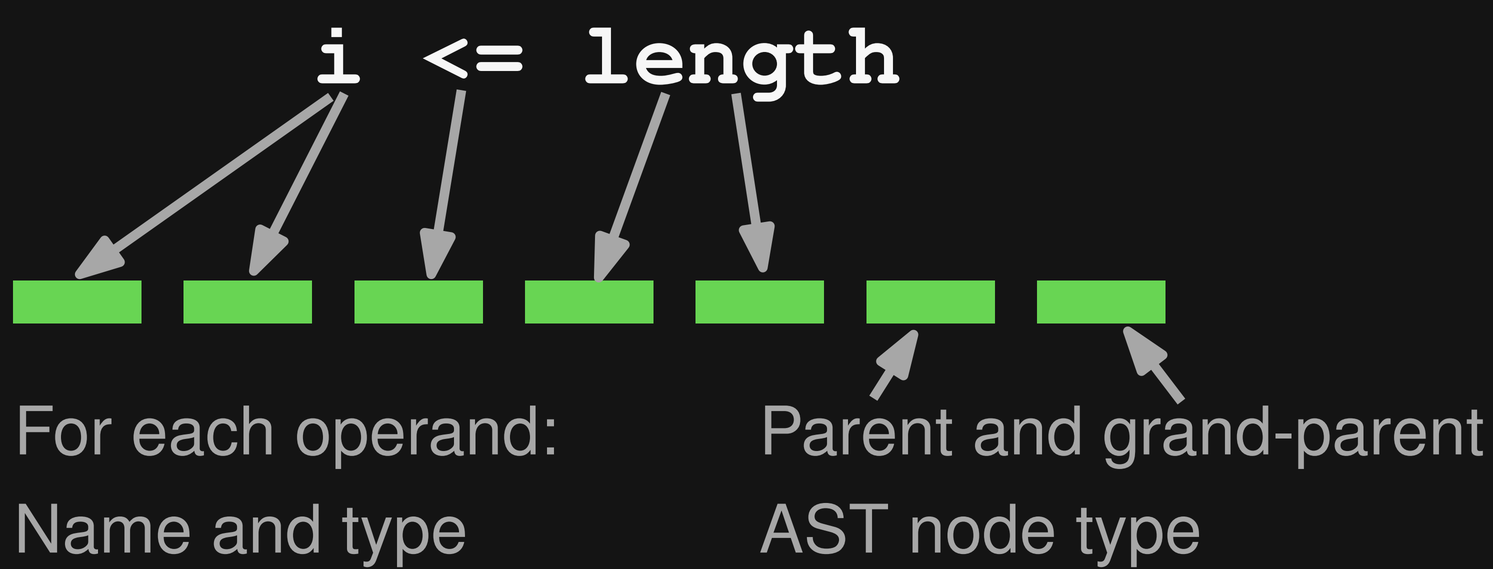


For each argument: Name, type, and formal parameter name

Code Snippets as Vectors

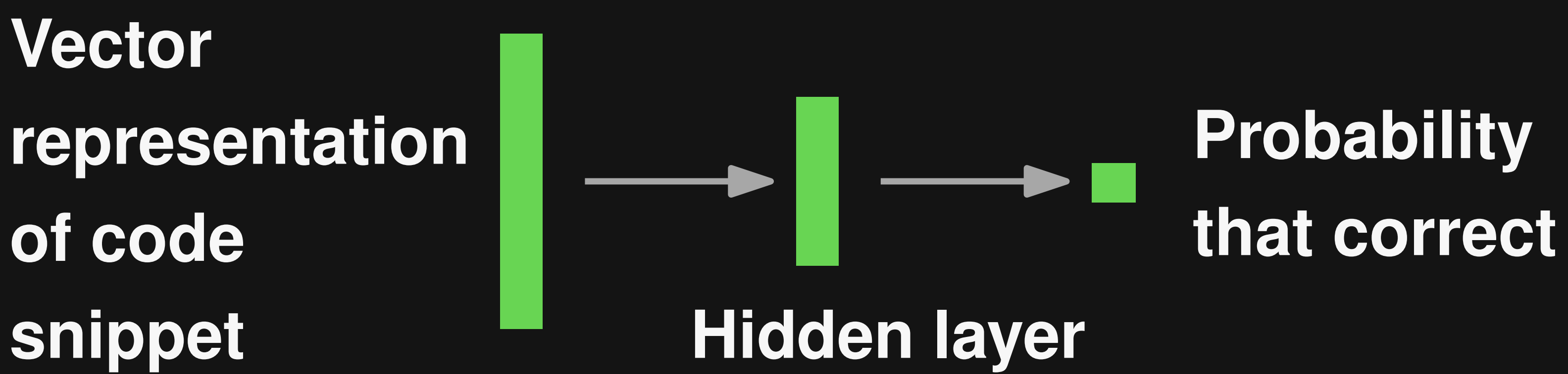
Concatenate embeddings of names in code snippet

2) + 3) Wrong binary operator/operation



Learning the Bug Detector

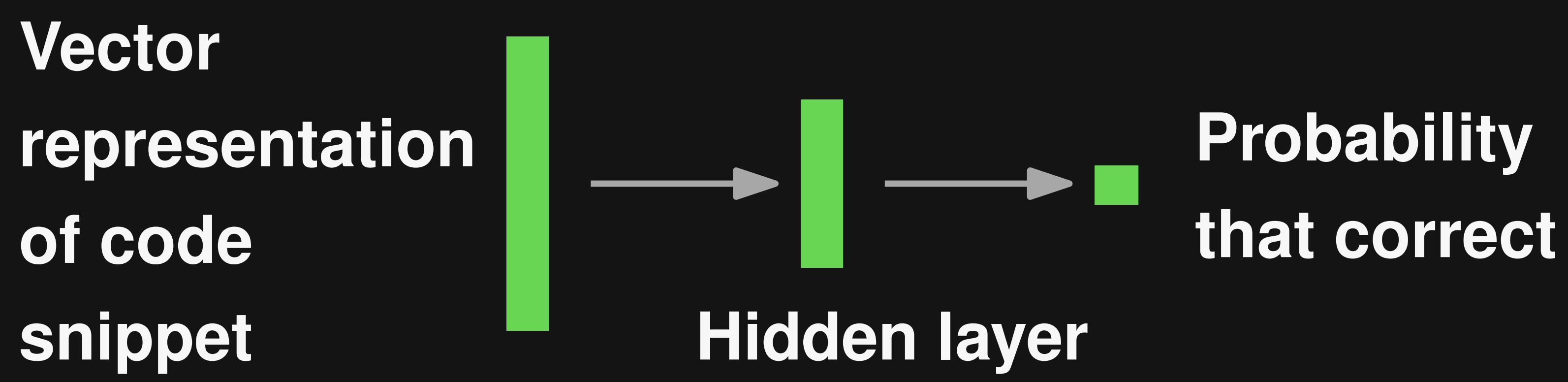
- Given: Vector representation of code snippet
- Train neural network: Predict whether correct or wrong



Hidden layer: size=200, dropout=0.2

RMSprop optimizer with binary cross-entropy as loss function

Predicting Bugs in New Code



- Represent **code snippet as vector**
- **Sort warnings** by predicted probability that code is incorrect

Evaluation: Setup

68 million lines of JavaScript code

- 150k files [Raychev et al.]
- 100k files for training, 50k files for validation

Bug detector	Examples	
	Training	Validation
Swapped arguments	1,450,932	739,188
Wrong binary operator	4,901,356	2,322,190
Wrong binary operand	4,899,206	2,321,586

Accuracy of Classifier

Bug detector	Validation accuracy
Swapped arguments	94.70%
Wrong binary operator	92.21%
Wrong binary operand	89.06%

Examples of Detected Bugs

```
// From Angular.js
browserSingleton.startPoller(100,
    function(delay, fn) {
        setTimeout(delay, fn);
    });
```

Examples of Detected Bugs

```
// From Angular.js
browserSingleton.startPoller(100,
    function(delay, fn) {
        setTimeout(delay, fn);
    });
```

First argument must be callback function

Examples of Detected Bugs

```
// From DSP.js
for(var i = 0; i<this.NR_OF_MULTIDELAYS; i++){
  // Invert the signal of every even multiDelay
  mixSampleBuffers(outputSamples, ...,
      2%i==0, this.NR_OF_MULTIDELAYS);
}
```

Examples of Detected Bugs

```
// From DSP.js
for(var i = 0; i<this.NR_OF_MULTIDELAYS; i++){
  // Invert the signal of every even multiDelay
  mixSampleBuffers(outputSamples, ...,
      2%i==0, this.NR_OF_MULTIDELAYS);
}
```

Should be `i%2==0`

Precision

Bug detector	Inspected	Bugs	Code quality	False pos.
Swapped args.	50	23	0	27
Wrong bin. operator	50	37	7	6
Wrong bin. operand	50	35	0	15
Total	150	95	7	48

Precision

Bug detector	Inspected	Bugs	Code quality	False pos.
Swapped args.	50	23	0	27
Wrong bin. operator	50	37	7	6
Wrong bin. operand	50	35	0	15
Total	150	95	7	48

68% true positives. High, even compared to manually created bug detectors

Importance of Embeddings

How many true positives do we miss with random embeddings?

- Misses 11 out of 102 true positives
- Example:

```
transform = is(obj, value) | is(func, value);
```

Importance of Embeddings

How many true positives do we miss with random embeddings?

- **Misses 11 out of 102 true positives**
- **Example:**

```
transform = is(obj, value) | is(func, value);
```

Bitwise OR for logical OR of booleans: Inefficient and error-prone

Efficiency

- **Data extraction and learning:**
  28 minutes – 59 minutes
  (depending on bug detector)

- **Prediction of bugs:**
  Less than 20ms per JavaScript file

48 Intel Xeon E5-2650 CPU cores, 64GB of memory, 1 NVIDIA Tesla P100 GPU

Open Challenges

- **Bug detection based on other code representations**
  - Token-based, graph-based, etc.
  - One representation for many bug patterns
- **Support more bug patterns**
  - Learn code transformations from version histories
  - A single model for multiple bug patterns

Conclusion

- **Bug detection as a learning problem**
  - Classify code as buggy or correct
- **DeepBugs: Name-based bug detector**
  - Exploit natural language information to detect otherwise missed bugs
  - Learning from seeded bugs yields classifier that detects real bugs

DeepBugs: A Learning Approach to Name-based Bug Detection
(Pradel & Sen, OOPSLA'18)