DeepBugs: A Learning Approach to Name-based Bug Detection

Michael Pradel
TU Darmstadt, software-lab.org

Joint work with Koushik Sen

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

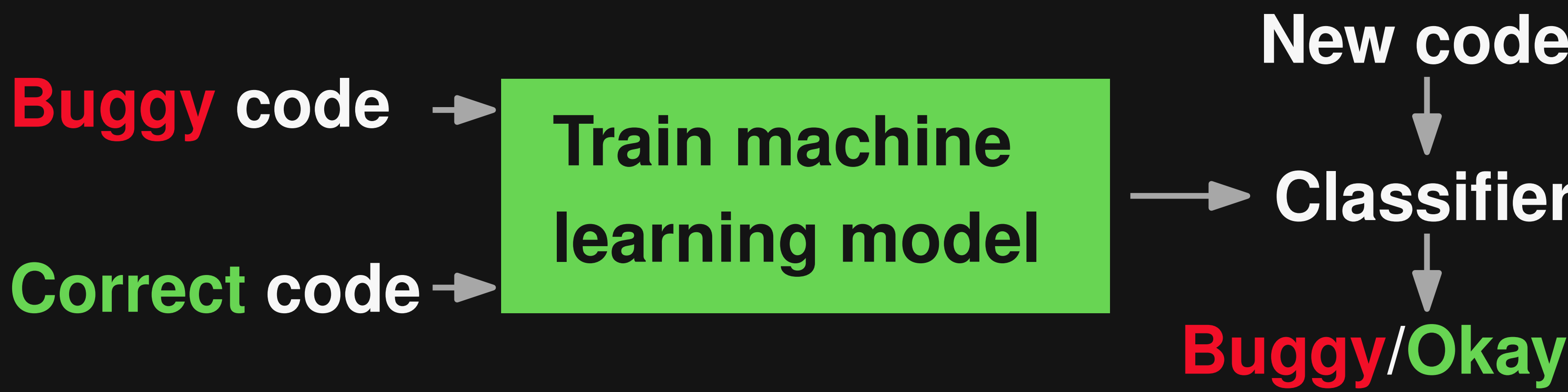
Time-consuming process

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

Buggy code →

Correct code →

Train machine learning model → Classifier

New code ↓ Classifier ↓ Buggy/Okay

How to get training data?

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

Learning to Find Bugs

Train a model to distinguish correct from buggy code

Buggy code

Correct code

Train machine learning model

New code

Classifier

Buggy/Okay

How to represent code?

- Token-based, AST-based, graph-based, etc.
- Here: Embeddings of natural language elements in code

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) {
    ...
  }
}
```

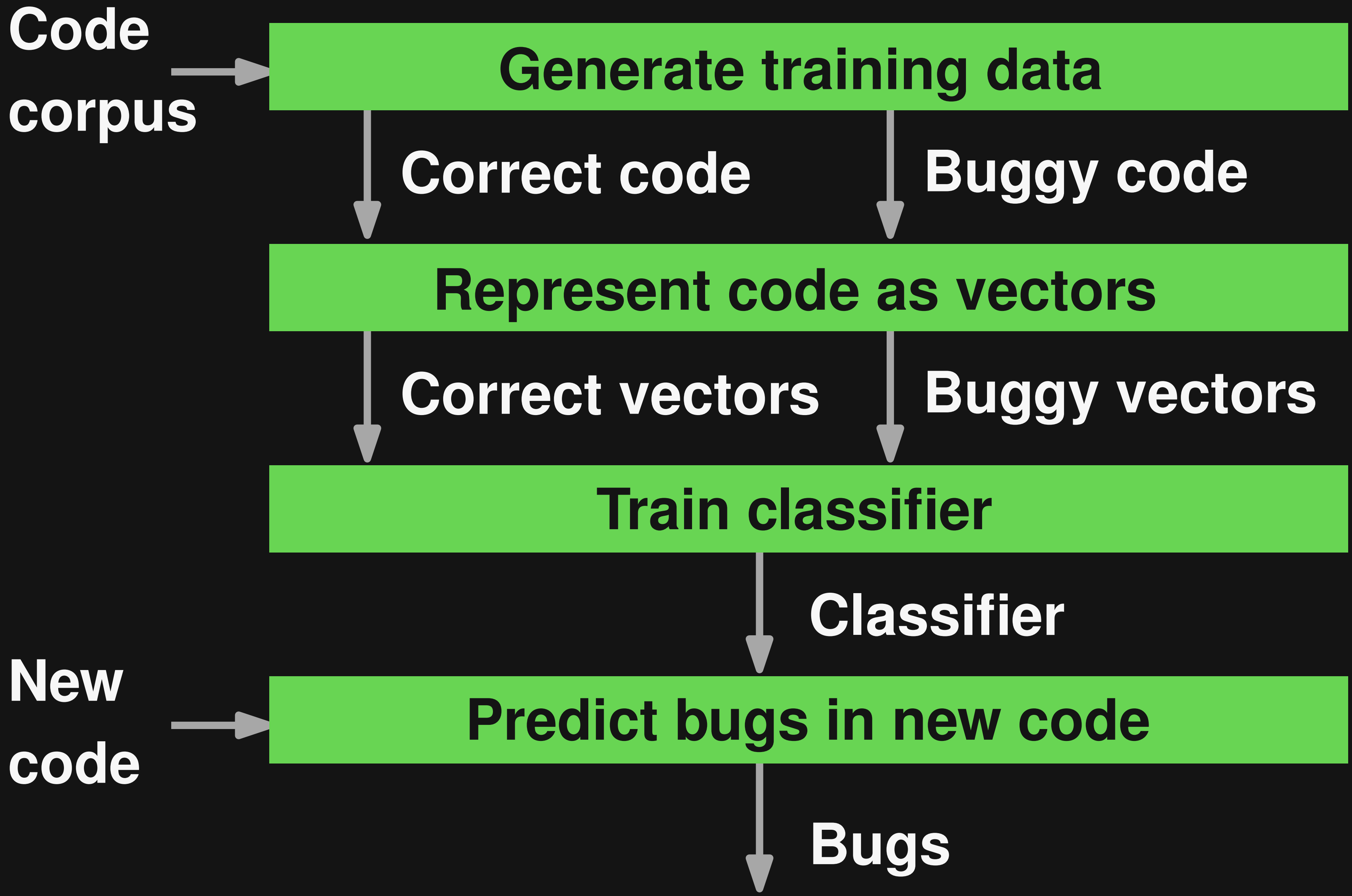
Name-related Bugs (2)

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

Goal: **Exploit natural language information** in **identifier names**

How to reason about identifier names?

- **Prior work: Lexical similarity**
  - `x` similar to `x_dim`
- **Want: Semantic similarity**
  - `x` similar to `width`
  - `list` similar to `seq`

Word2Vec

Word embeddings

- Continuous vector representation for each word
- Similar words have similar vectors

Learn embeddings from corpus of text

- Context: Surrounding words in sentences

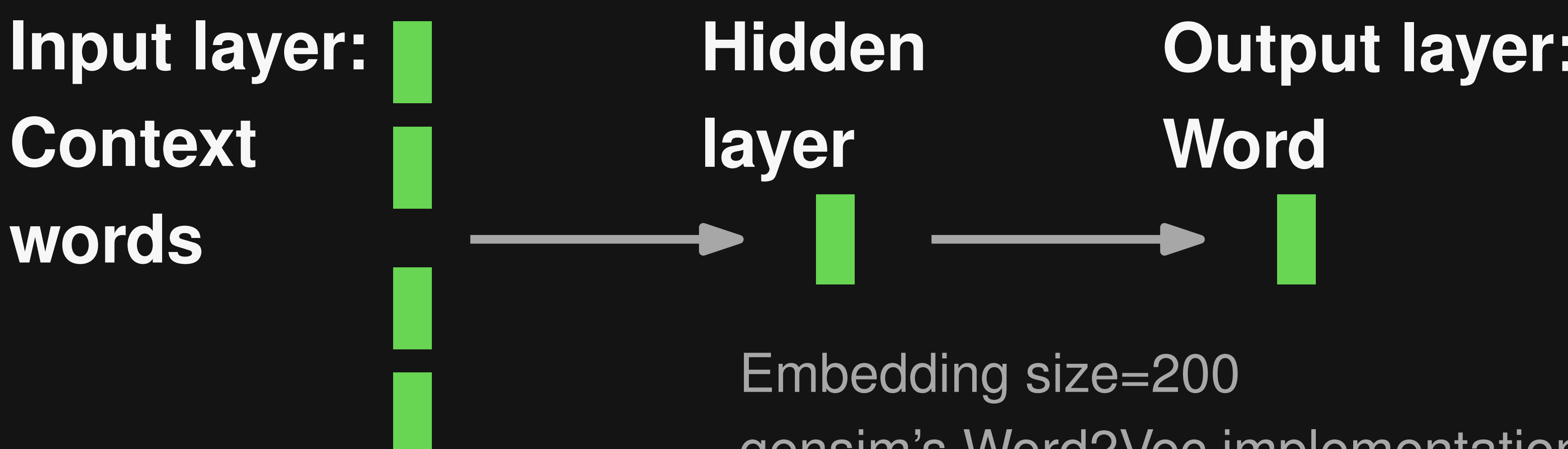
Word2Vec

Word embeddings

- Continuous vector representation for each word
- Similar words have similar vectors

Learn embeddings from corpus of text

- Context: Surrounding words in sentences



Embedding size=200

gensim's Word2Vec implementation

Word2Vec for Source Code

Natural language		Programming language
Sentences	→	Program
Words	→	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

- Sentences
- Words

Programming language

- Program
- Tokens

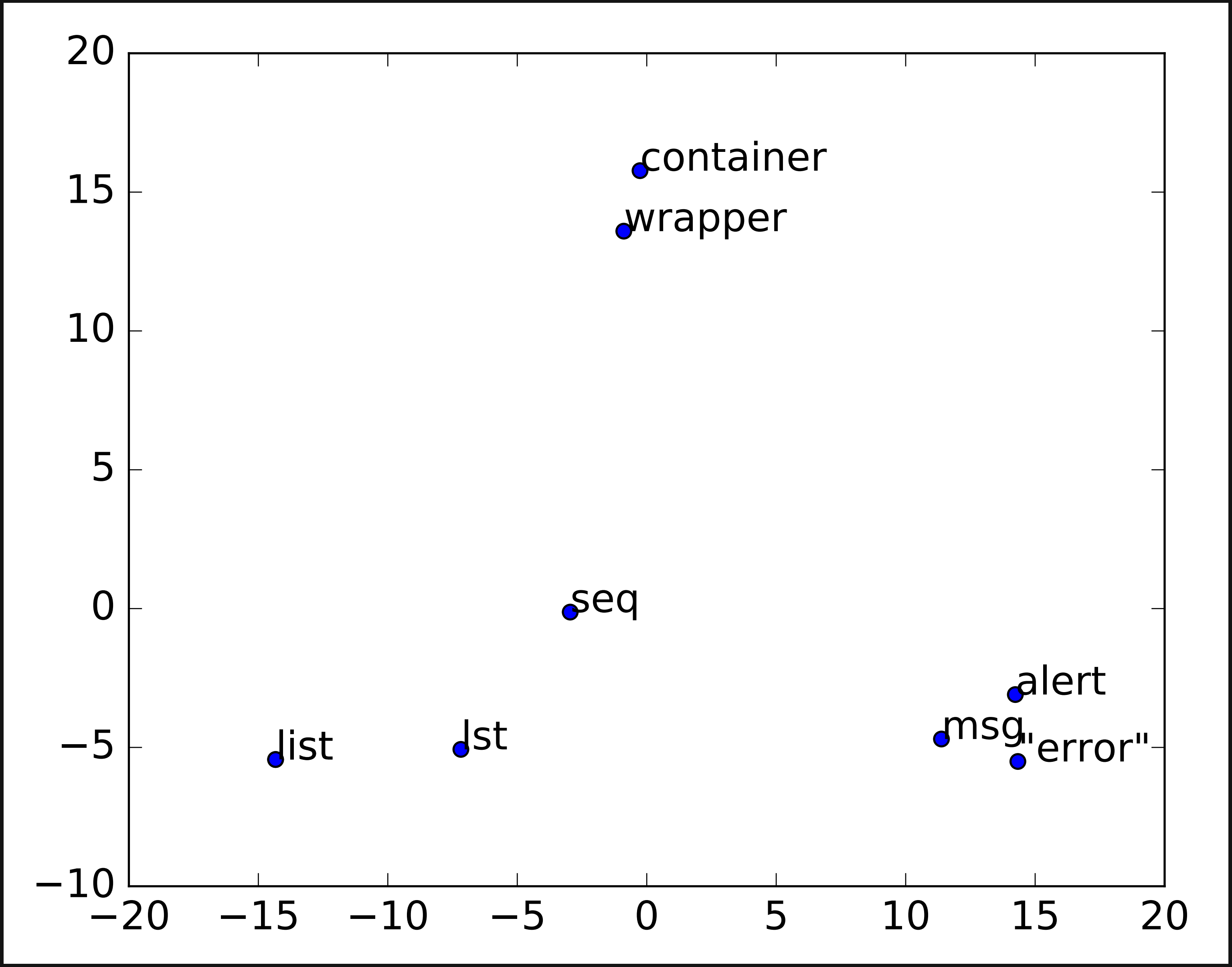
```
function setPoint(x, y) { ... }
```

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

Context of `x`:

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

Example: Embeddings

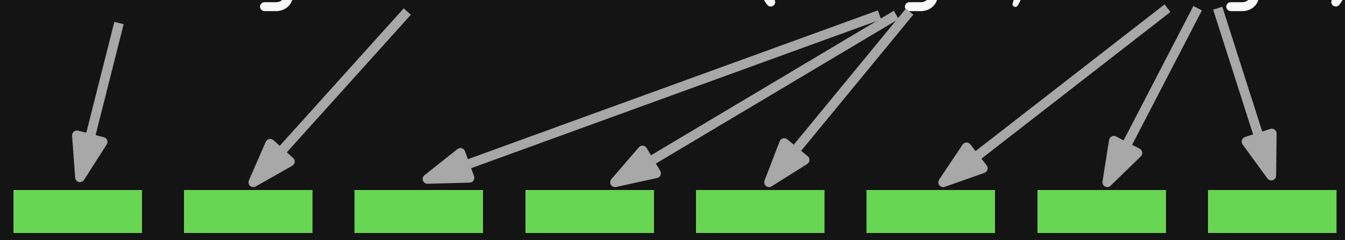


Code Snippets as Vectors

Concatenate embeddings of names in code snippet

1) Swapped arguments

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



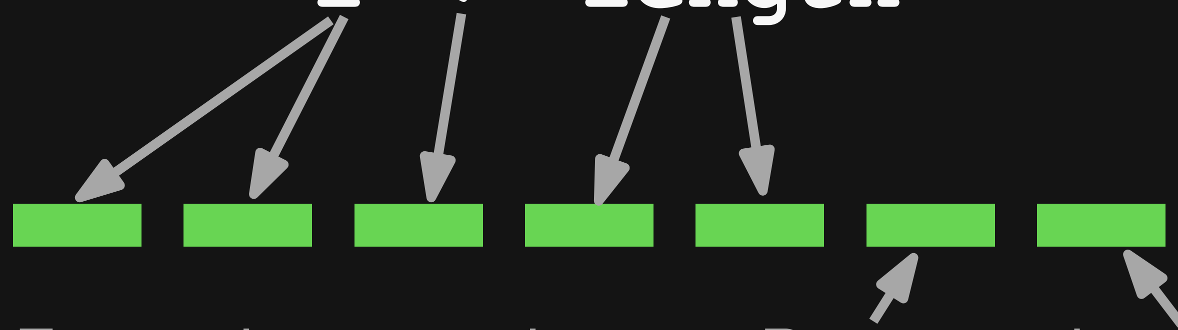
For each argument: Name, type, and formal parameter name

Code Snippets as Vectors

Concatenate embeddings of names in code snippet

2) + 3) Wrong binary operator/operation

```
i <= length
```

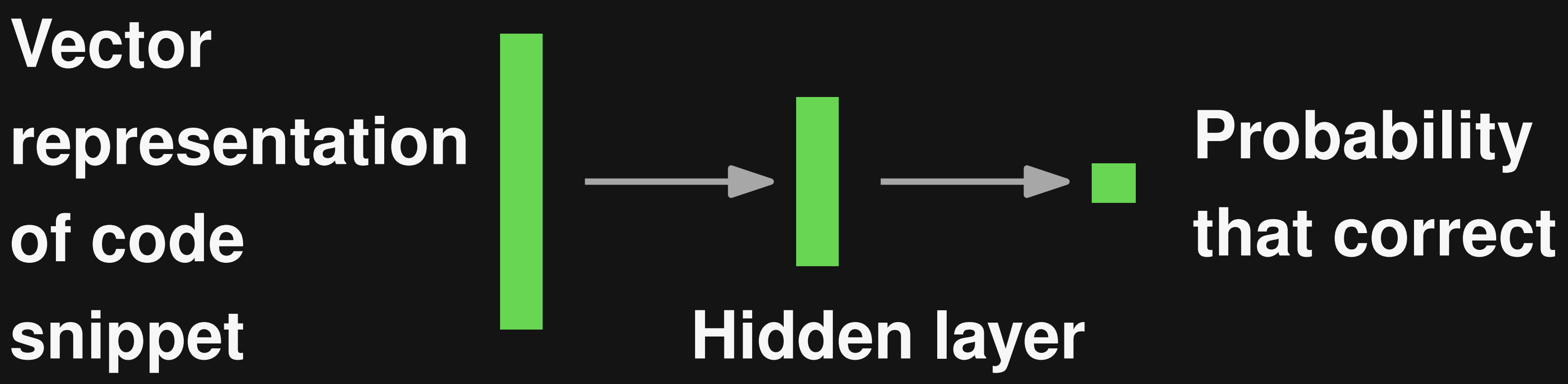


For each operand:
Name and type

Parent and grand-parent
AST node type

Learning the Bug Detector

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

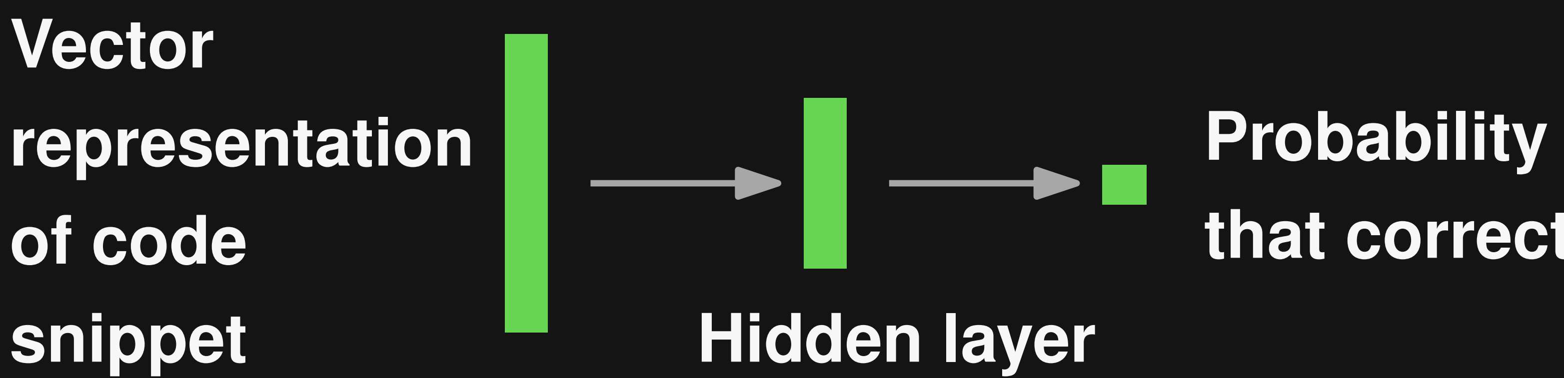


Hidden layer: size=200, dropout=0.2

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

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

Accuracy of Classifier

Validation accuracy (after training)

Swapped arguments	94.70%
Wrong binary operator	92.21%
Wrong binary operand	89.06%

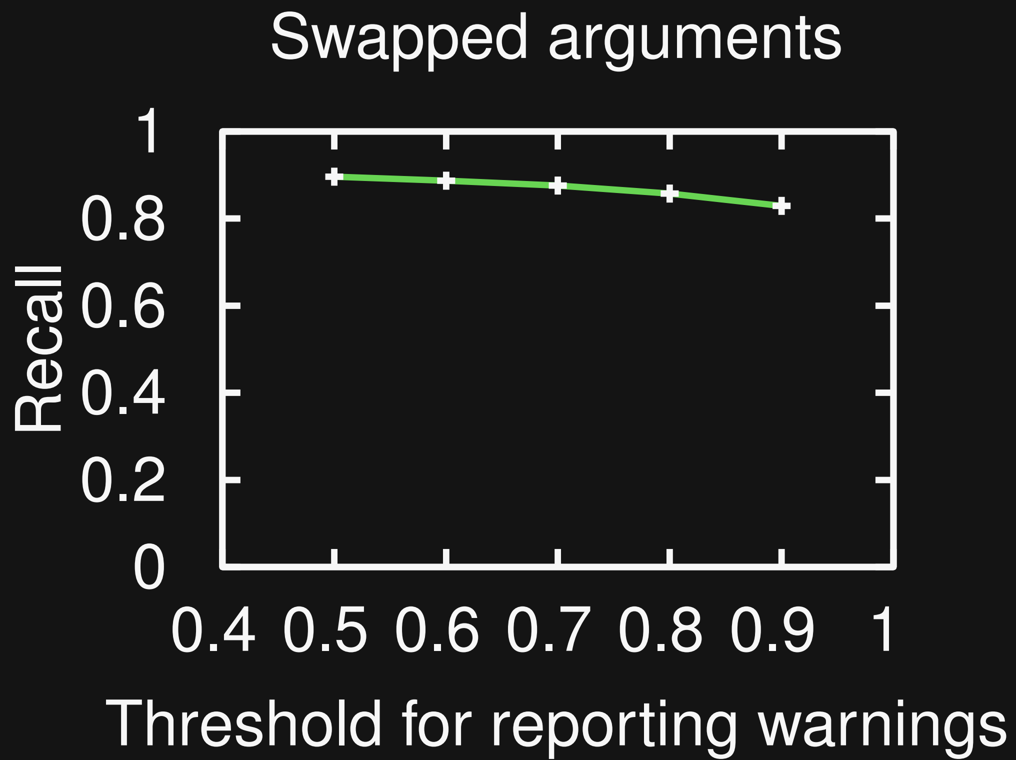
Accuracy of Classifier

Validation accuracy (after training)

	Embedding	
	Random	Learned
Swapped arguments	93.88%	94.70%
Wrong binary operator	89.15%	92.21%
Wrong binary operand	84.79%	89.06%

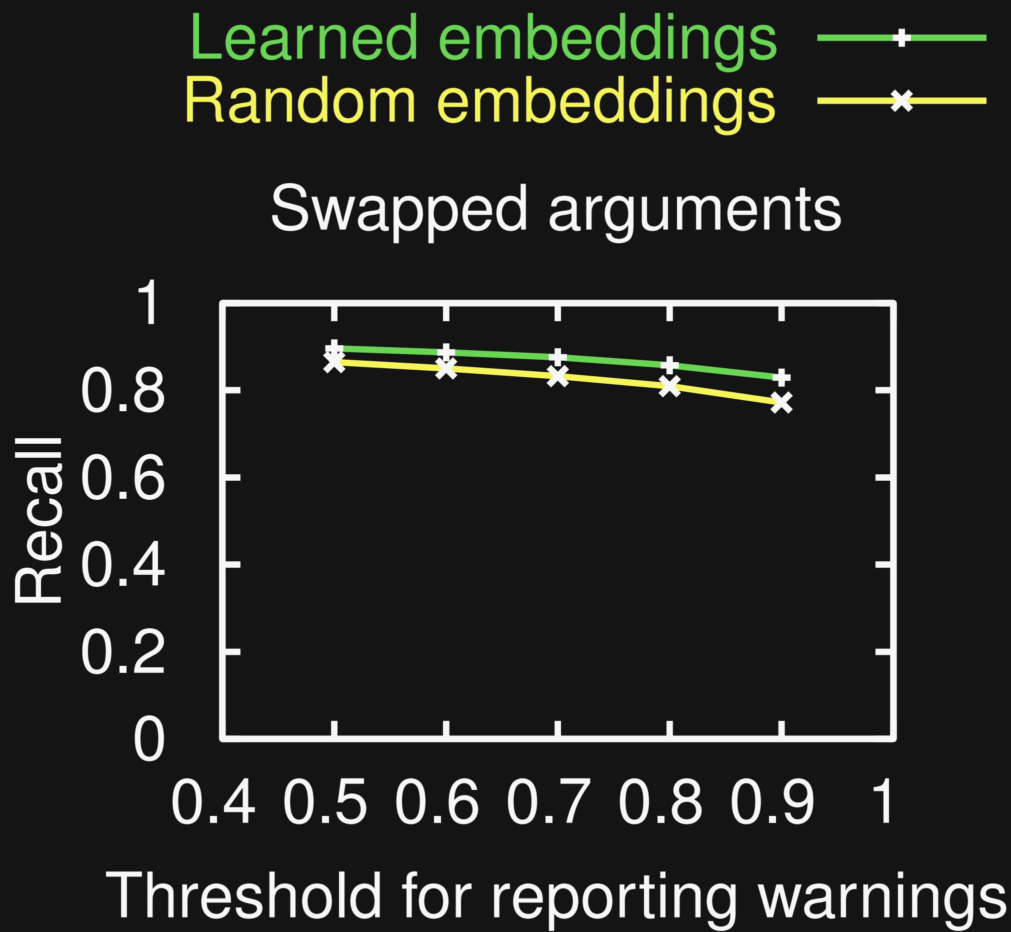
Recall of Seeded Bugs

How many of all seeded bugs are found?



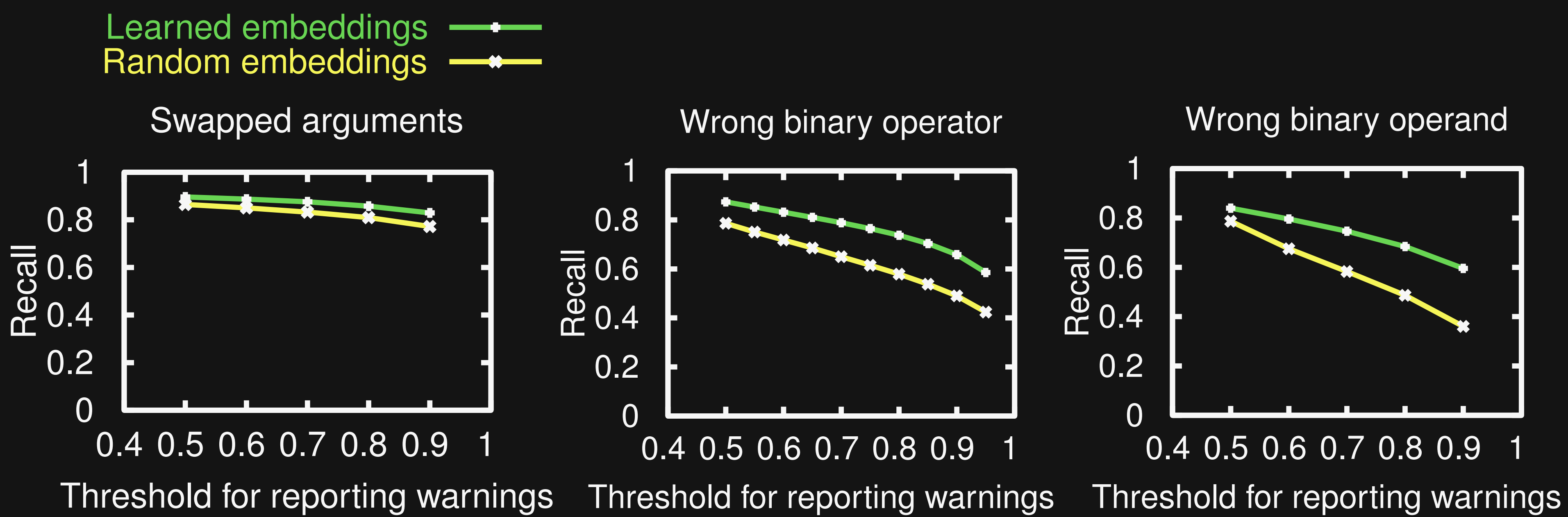
Recall of Seeded Bugs

How many of all seeded bugs are found?



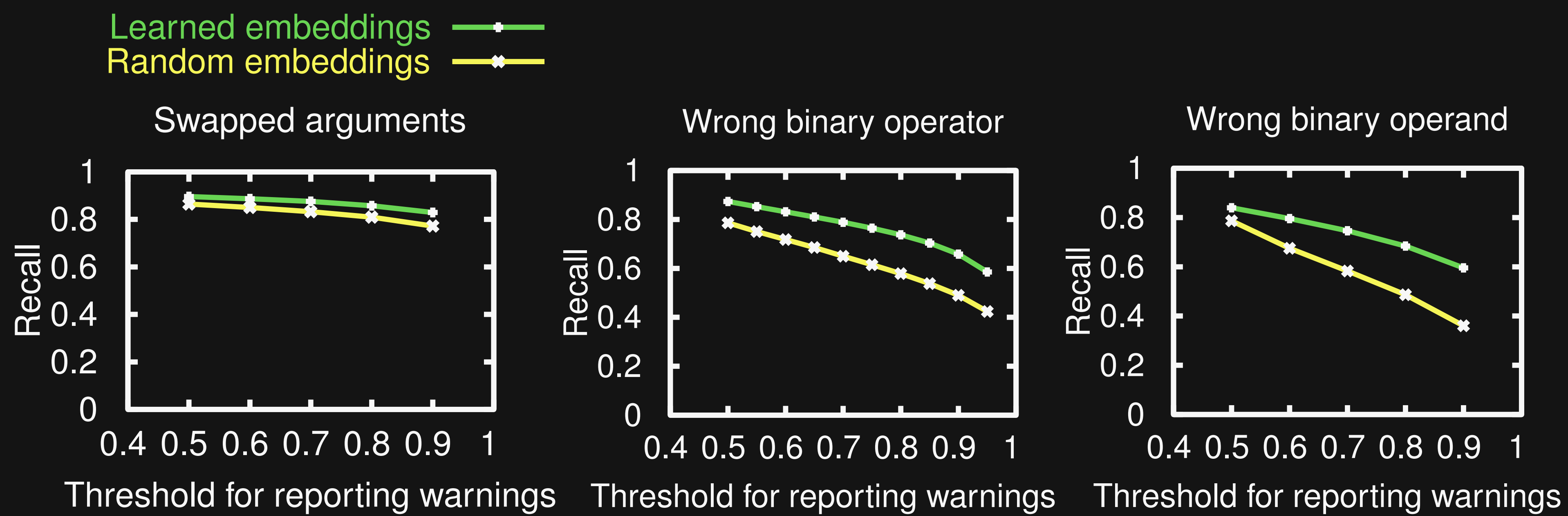
Recall of Seeded Bugs

How many of all seeded bugs are found?



Recall of Seeded Bugs

How many of all seeded bugs are found?



Embeddings enable generalization across similar names

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
  - Train one model per bug pattern

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

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

ASE'18: *How Many of All Bugs Do We Find? A Study of Static Bug Detectors* (Habib & Pradel)