Learning to Find Bugs

(Work in progress)

Michael Pradel
TU Darmstadt

Joint work with Koushik Sen and Rohan Bavishi

Automated Bug Detection

Hundreds of bug detectors

- One analysis for each bug pattern
- E.g., Google's Error Prone framework: 150+ different analyses

Thousands of bug patterns

- Existing bug detectors miss most bugs

Automated Bug Detection

Hundreds of bug detectors

- One analysis for each bug pattern
- E.g., Google's Error Prone framework: 150+ different analyses

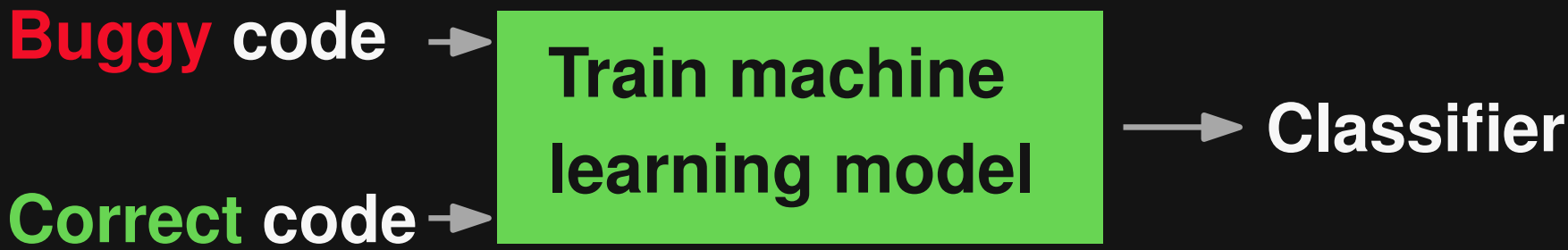
Thousands of bug patterns

- Existing bug detectors miss most bugs

Manually creating and tuning bug detectors doesn't scale

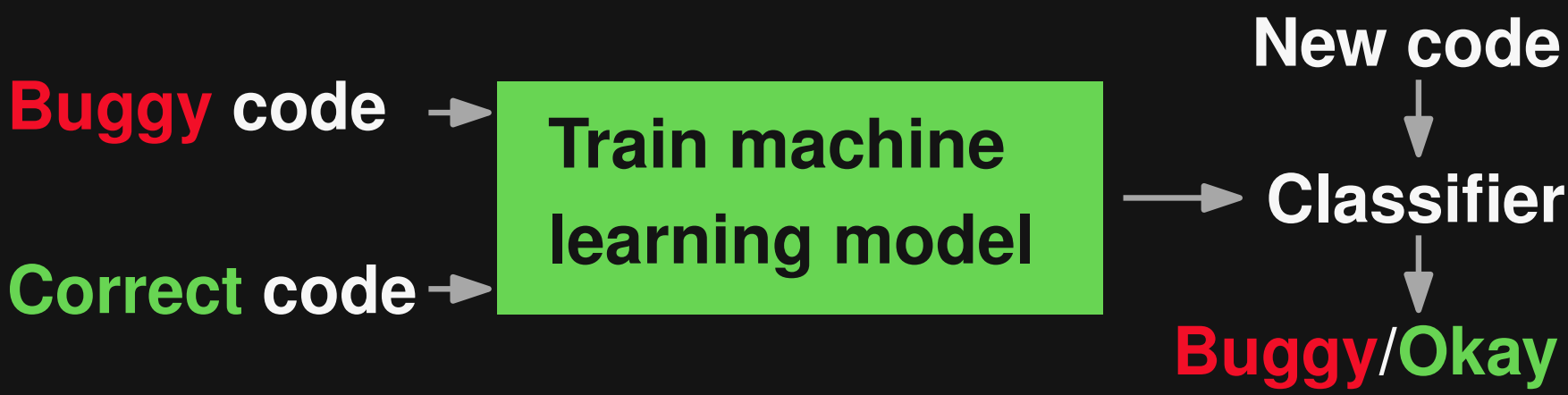
Learning to Find Bugs

Train a model to identify instances of bug patterns:



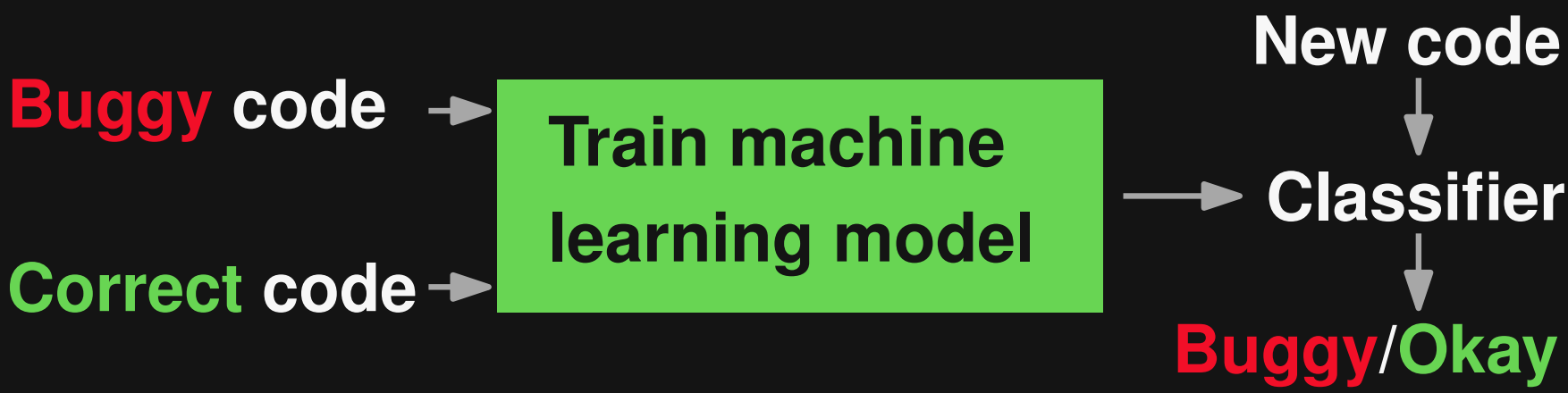
Learning to Find Bugs

Train a model to identify instances of bug patterns:



Learning to Find Bugs

Train a model to identify instances of bug patterns:



Problem of writing program analysis

↓

Problem of finding training examples

Here: Name-based Bug Detection

What's wrong with this code?

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

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

Here: Name-based Bug Detection

What's wrong with this code?

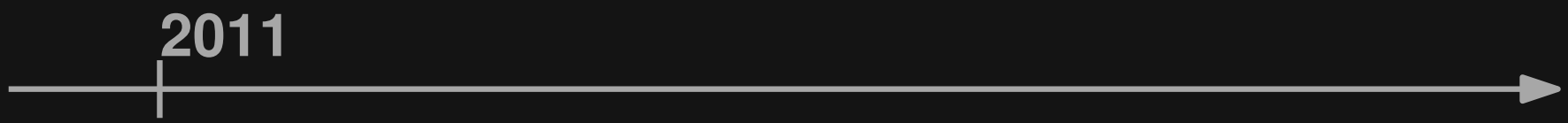
```
function setPoint(x, y) { ... }

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

Incorrect order of arguments

Prior Work

Name-based bug detection

- Find unusual and likely incorrect arguments
- Exploit similarities of identifier names



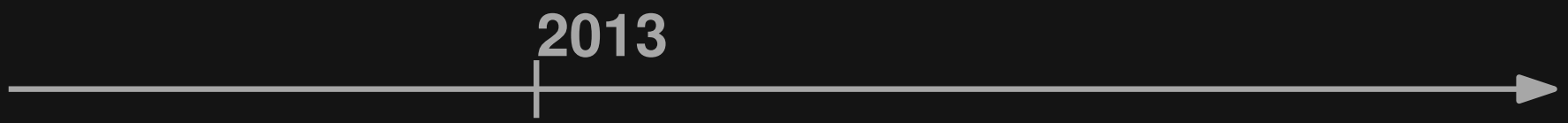
First name-based bug detector [ISSTA'11]

- Finds incorrectly ordered, equally typed arguments
- Compares call sites of same method

Prior Work

Name-based bug detection

- Find unusual and likely incorrect arguments
- Exploit similarities of identifier names



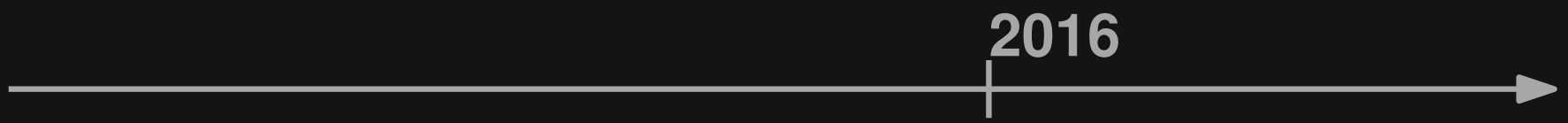
Improved analysis [TSE'13]

- Improved precision
- Effective for multiple languages (Java, C, C++)

Prior Work

Name-based bug detection

- Find unusual and likely incorrect arguments
- Exploit similarities of identifier names



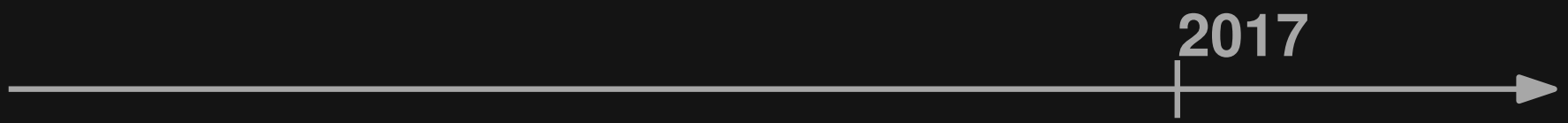
Generalized analysis [ICSE'16]

- Apply to arbitrary arguments
- Heuristic pruning of false positives

Prior Work

Name-based bug detection

- Find unusual and likely incorrect arguments
- Exploit similarities of identifier names



Adopted by Google [OOPSLA'17]

- Default check in Error Prone framework
- Found 2000+ new bugs

Problem Solved?

Various hand-tuned heuristics

- **Detect more bugs**
  - Special check for `assertEquals` calls
- **Reduce false positives**
  - Hard-coded method names that suggest that swapping is intended, e.g., `transpose`

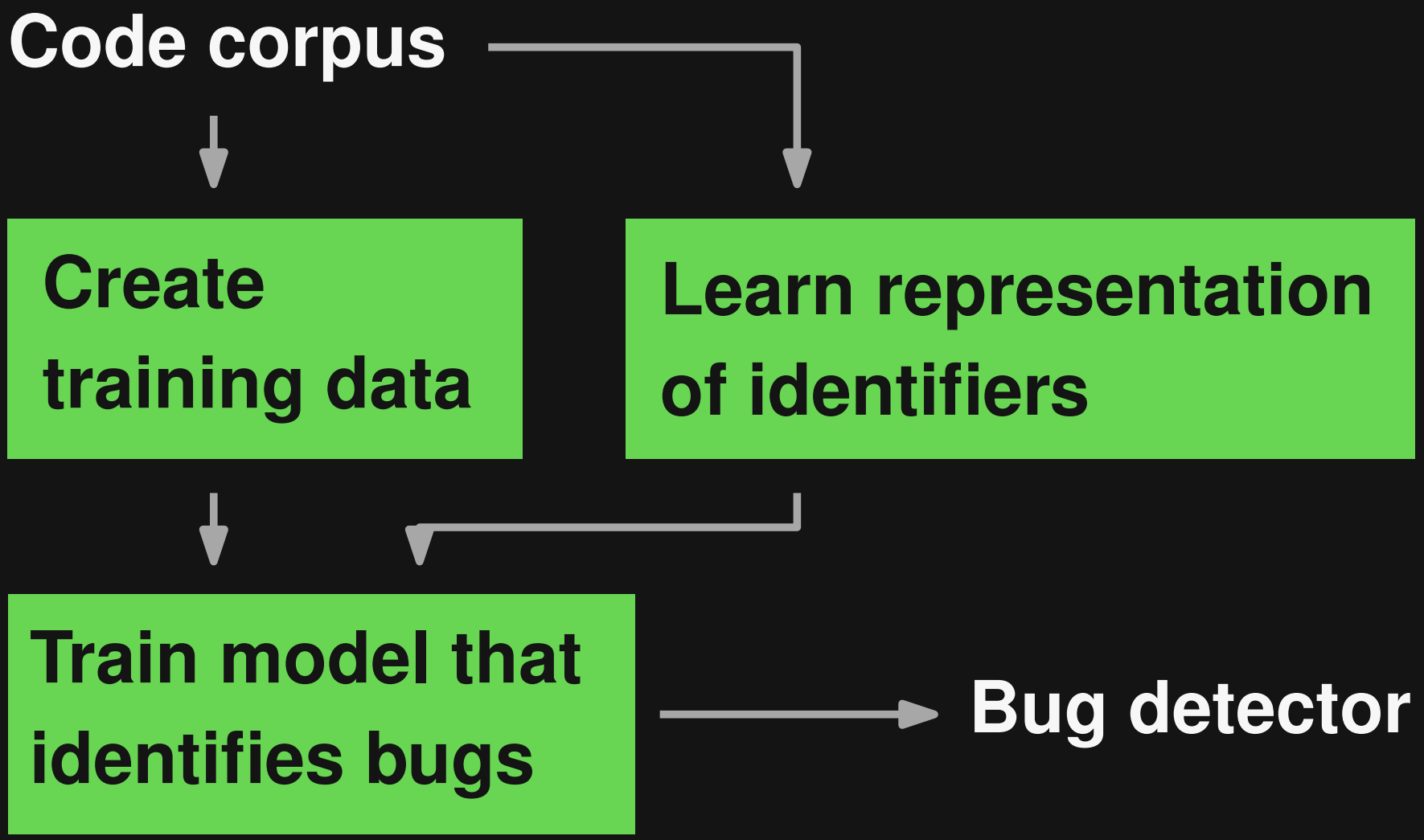
Problem Solved?

Various hand-tuned heuristics

- **Detect more bugs**
  - Special check for `assertEquals` calls
- **Reduce false positives**
  - Hard-coded method names that suggest that swapping is intended, e.g., `transpose`

Goal: Replace hand-tuned analysis with trained machine learning model

This Work: Overview



Creating Training Data

Program transformation that seeds bugs

For swapped arguments:

- Visit every function call with ≥ 2 arguments
- Positive example: Original order of arguments
- Negative example: Swap first two arguments

```
setPoint(x, y);   ⟶   setPoint(y, x);
```

Representing Identifiers

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`

Background: Word Embeddings

Word embeddings in NLP

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

Word2Vec: Learn from corpus of text

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

AST Context

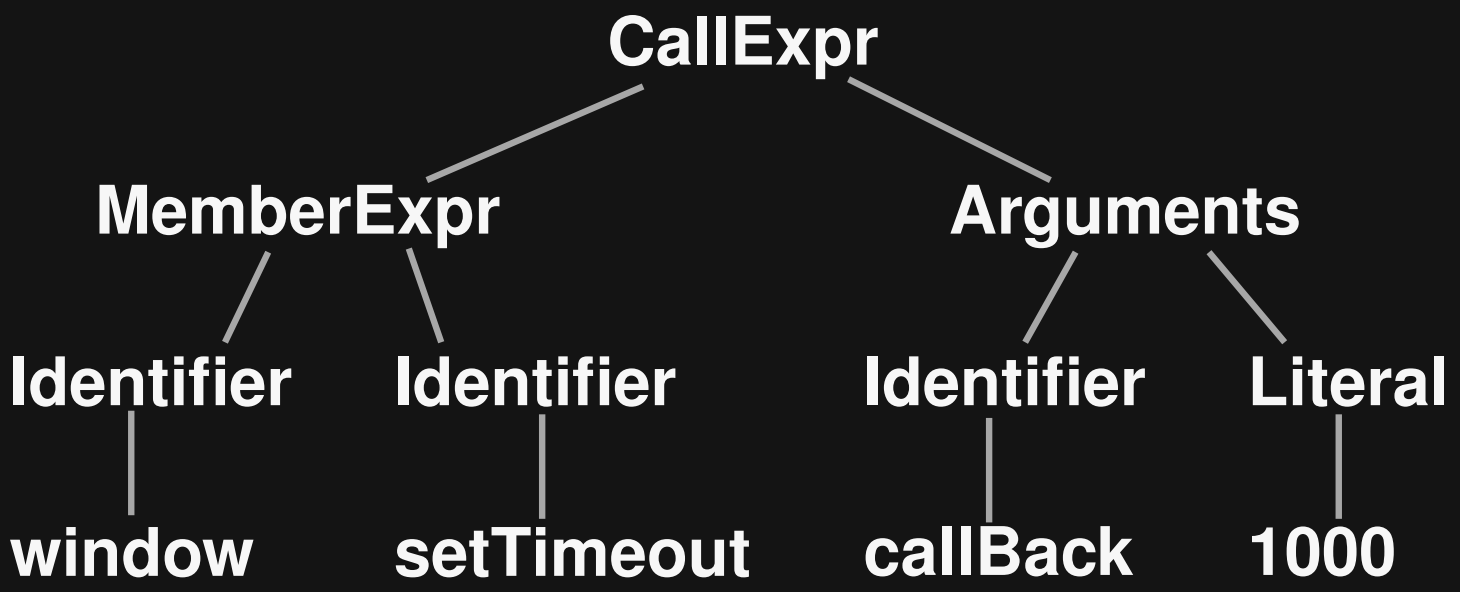
What's the context of an identifier?

Our approach: AST-based context

- Surrounding nodes:
  Parent, grandparent, siblings, etc.
- Extract node types, node contents, and relative positioning

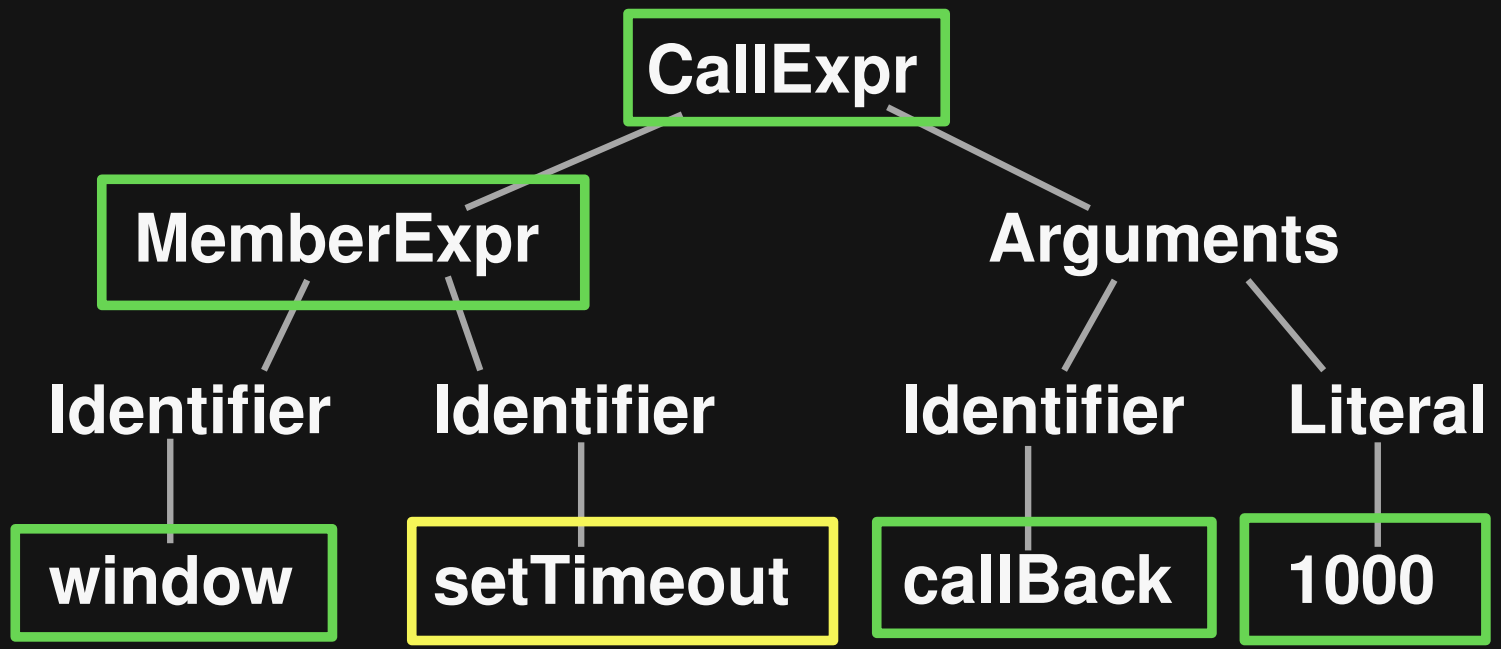
AST Context: Example

```
window.setTimeout(callback, 1000);
```



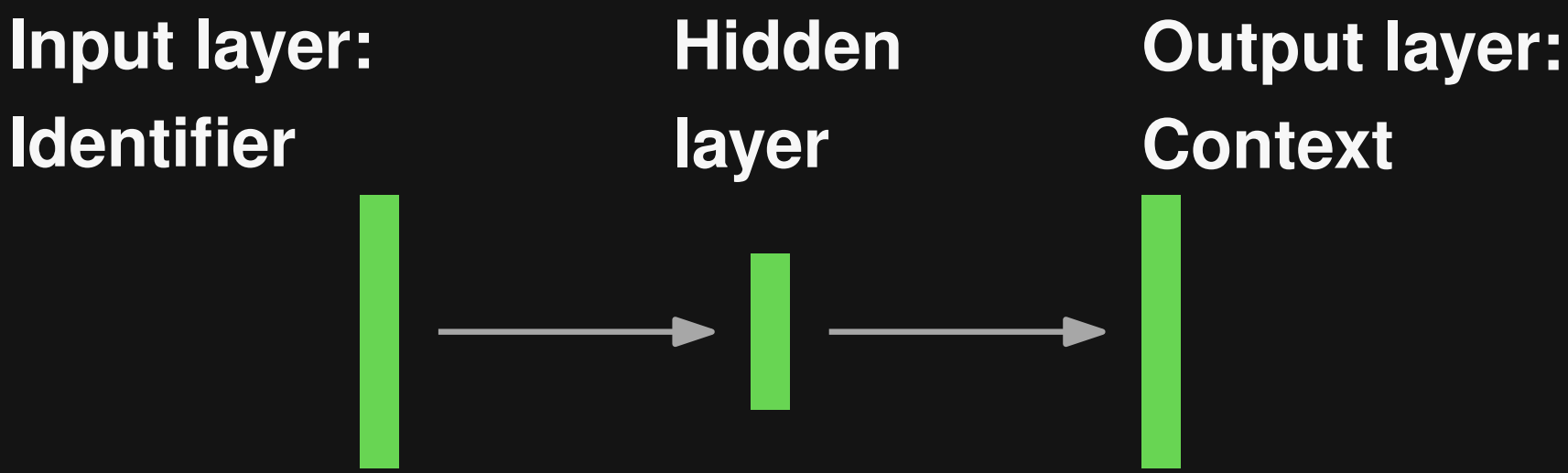
AST Context: Example

```
window.setTimeout(callback, 1000);
```



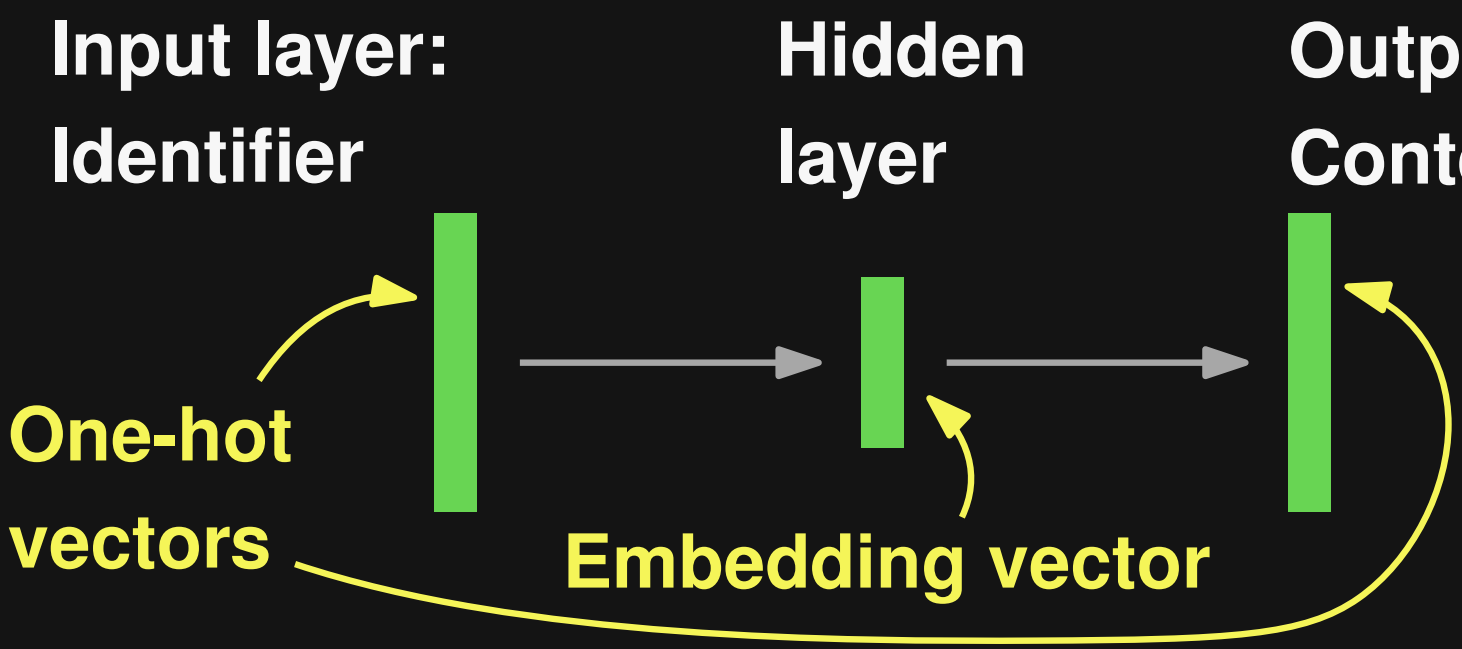
Learning Embeddings

- Train neural network to predict context from identifier
- Use hidden layer as representation for identifier



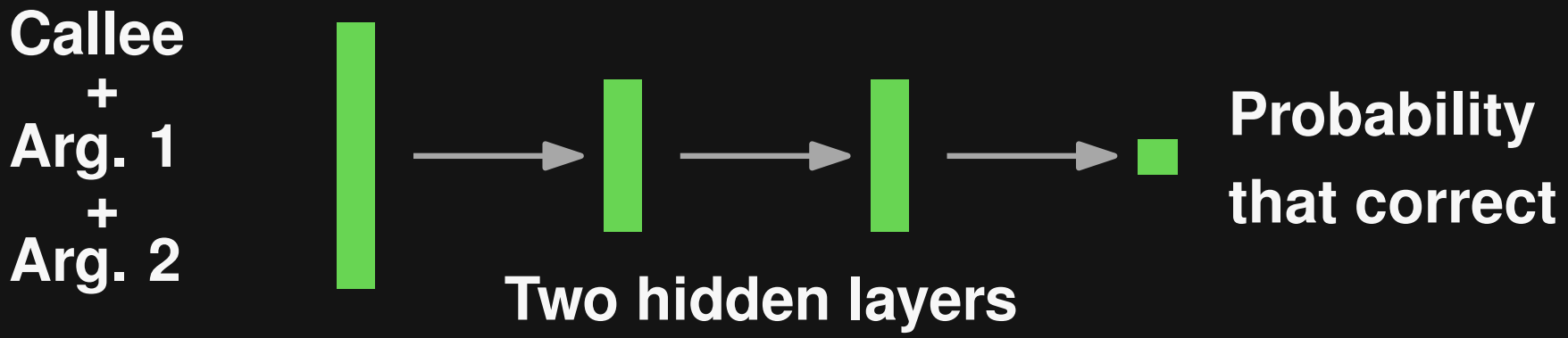
Learning Embeddings

- Train neural network to predict context from identifier
- Use hidden layer as representation for identifier



Training the Bug Detector

- Given: Embeddings of callee and two arguments
- Train neural network:
  Predict whether correct or wrong



Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values
- Incorrect binary operators
- Swapped operands of binary operations

Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values

```
var callback = function() { .. }
```

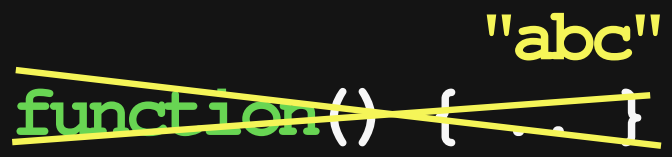
- Incorrect binary operators

- Swapped operands of binary operations

Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values



- Incorrect binary operators

- Swapped operands of binary operations

Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values

```
var callback = function() { ... }
```

"abc"

- Incorrect binary operators

```
if (x == undefined) ...
```

- Swapped operands of binary operations

Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values

```
var callback = function() { ... }
```

"abc"

- Incorrect binary operators

```
if (x == undefined) ...
```

>

- Swapped operands of binary operations

Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values

```
var callback = function() { ... }
```

"abc"

- Incorrect binary operators

```
if (x == undefined) ...
```

>

- Swapped operands of binary operations

```
bytes[i + 1] >> 4
```

Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values

```
                                          "abc"
var callback = function() { ... }
```

- Incorrect binary operators

```
               >
if (x == undefined) ...
```

- Swapped operands of binary operations

```
        4 >> bytes[i + 1]
bytes[i + 1] >> 4
```

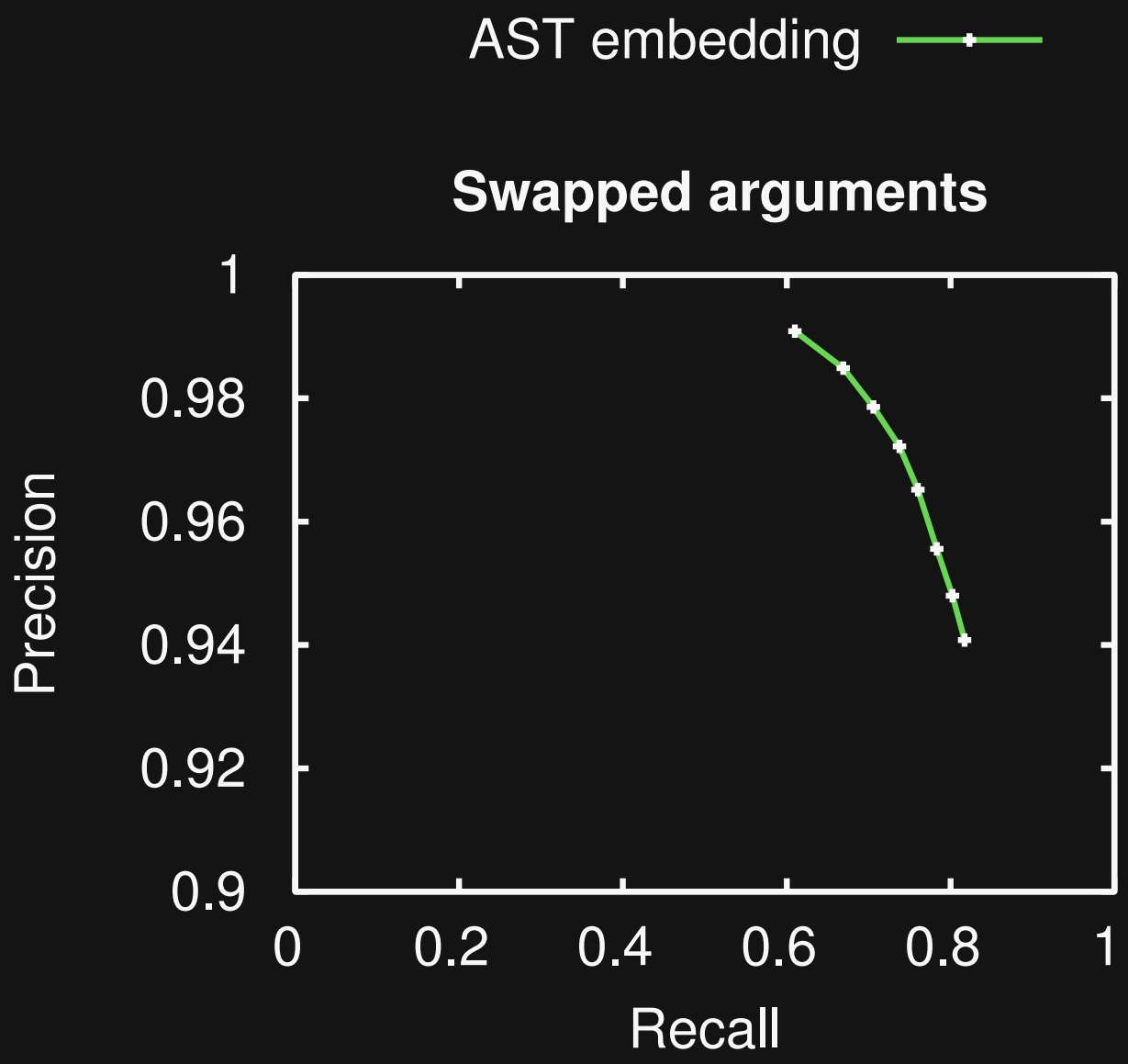
Evaluation: Setup

- **100.000 JavaScript files from various projects**
  - 80.000 for training
  - 20.000 for validation
- **68 million lines of code**
  - 37.3 million occurrences of identifiers
  - 10.1 million occurrences of literals

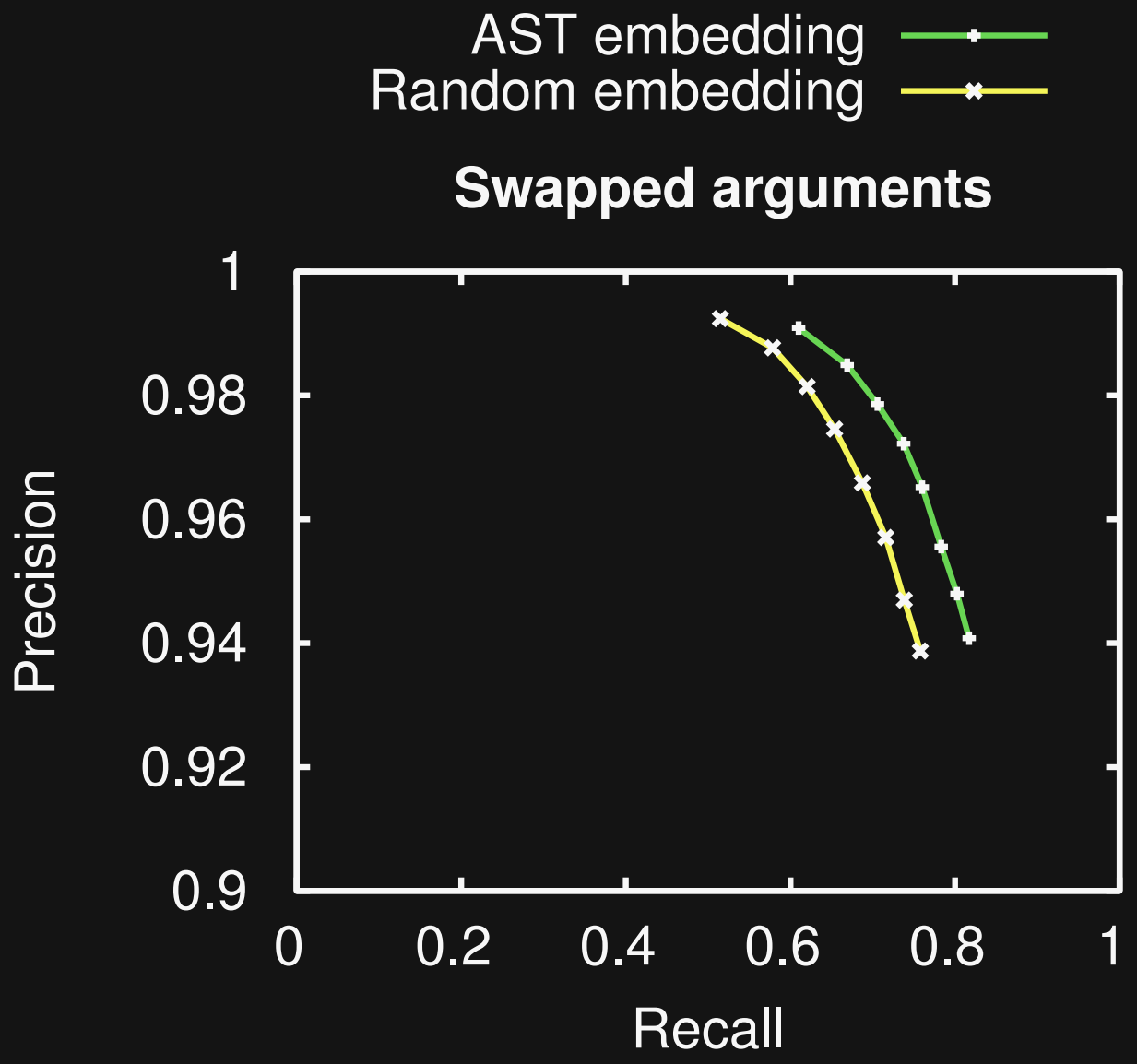
Examples of Bugs

```
// Callback must come before the
// number of milliseconds to wait
setTimeout(50, dojo.lang.hitch(this,
  function(){ ... }));

// First argument must be smaller than
// the second argument
array.slice(3, 0);
```

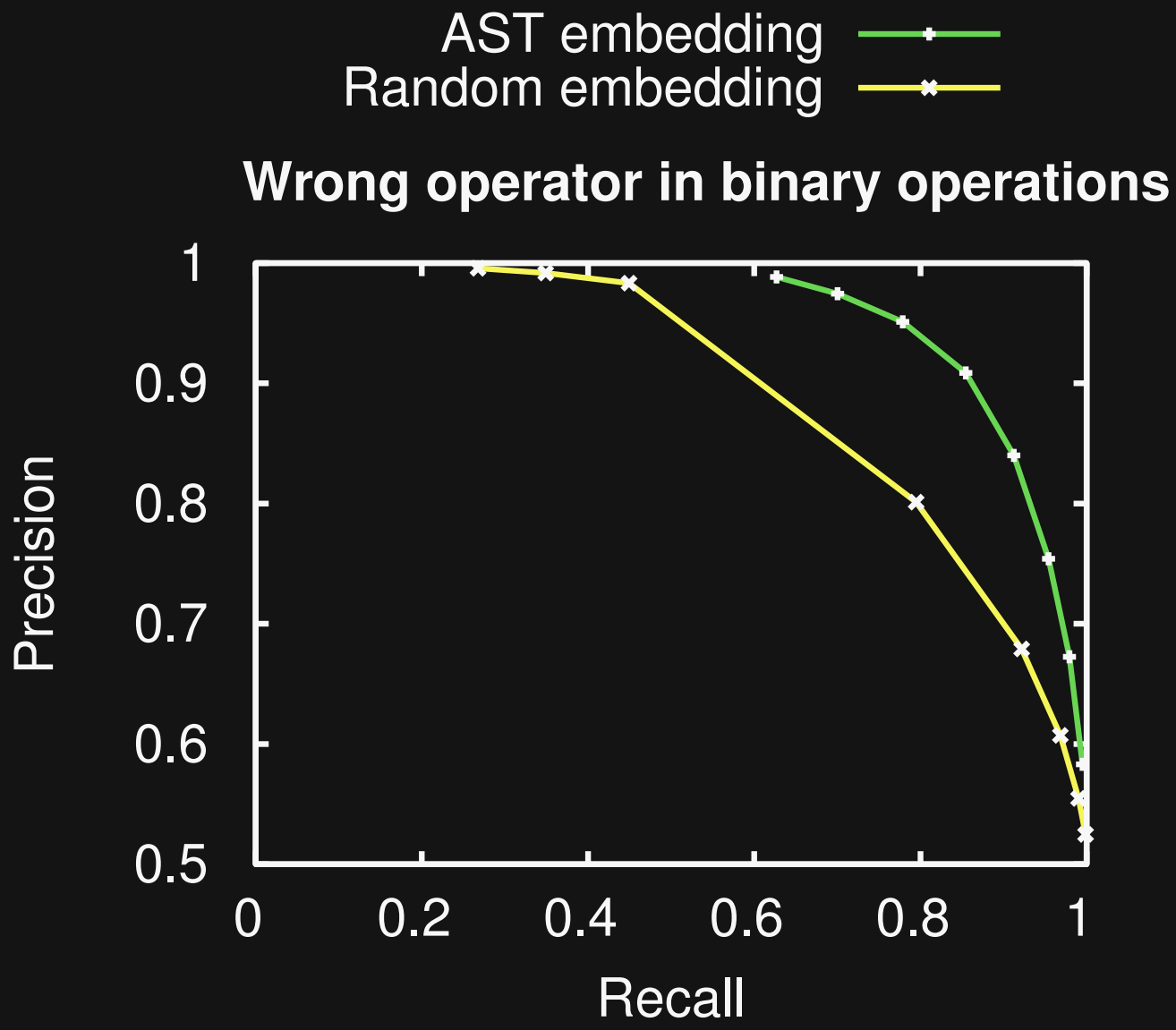
Precision and Recall



Precision and Recall



Precision and Recall



Open Challenges

Better representation of identifiers

- Same name $\not\Rightarrow$ Same meaning

Ensure that seeded bugs are realistic

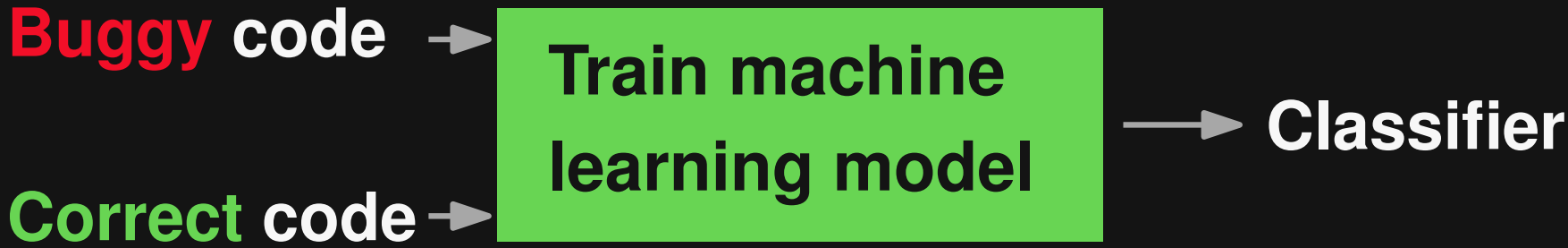
- Learn bug patterns from version histories?

Generalize to more bug patterns

- Train a model per bug pattern

Conclusion

Replace manually written program analyses with trained machine learning models



Precision and recall match or exceed manually written analyses