Neural Software Analysis: Learning Developer Tools from Code

Michael Pradel

Software Lab – University of Stuttgart

Joint work with Koushik Sen, Georgios Gousios, Jason Liu, and Satish Chandra

Developers Need Tools

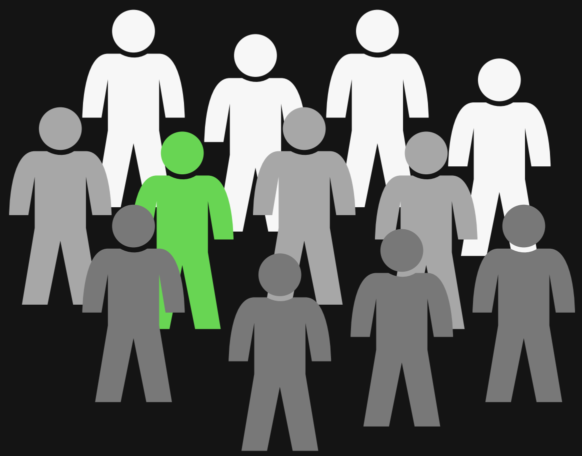
Key feature of humans:

Ability to develop tools

Software development tools, e.g., compilers, bug detection, code completion, documenting software

Creating Developer Tools

Today:



Manually crafted rules,

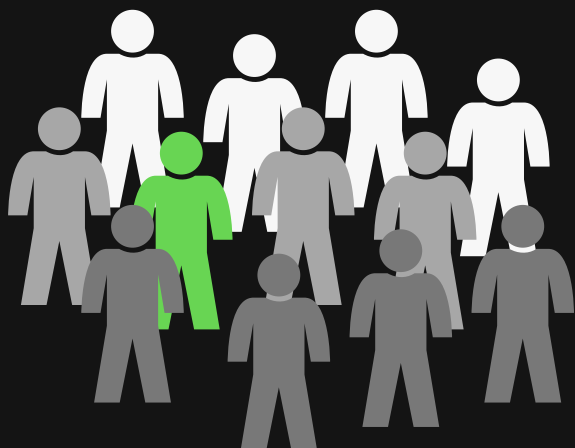
Years of work,

Experts only

E.g., few, predefined kinds of bugs

Creating Developer Tools

Today:



Manually crafted rules,
Years of work,
Experts only

E.g., few, predefined kinds of bugs

Vision:

Automatic,
Learned in minutes,
Widely accessible

E.g., many more kinds of bugs

Learning Developer Tools

Insight: Lots of **data** about **software development** to **learn** from

Source code
Execution traces
Documentation
Bug reports
etc.

→ **Machine Learning** →

Predictive tool

Learning Developer Tools

Insight: Lots of **data** about **software development** to **learn** from

Source code
Execution traces
Documentation
Bug reports
etc.

→ **Machine Learning** →

New code, execution, etc.

↓

Predictive tool

↓

Information useful for developers

This Talk

Two examples of learned developer tools *

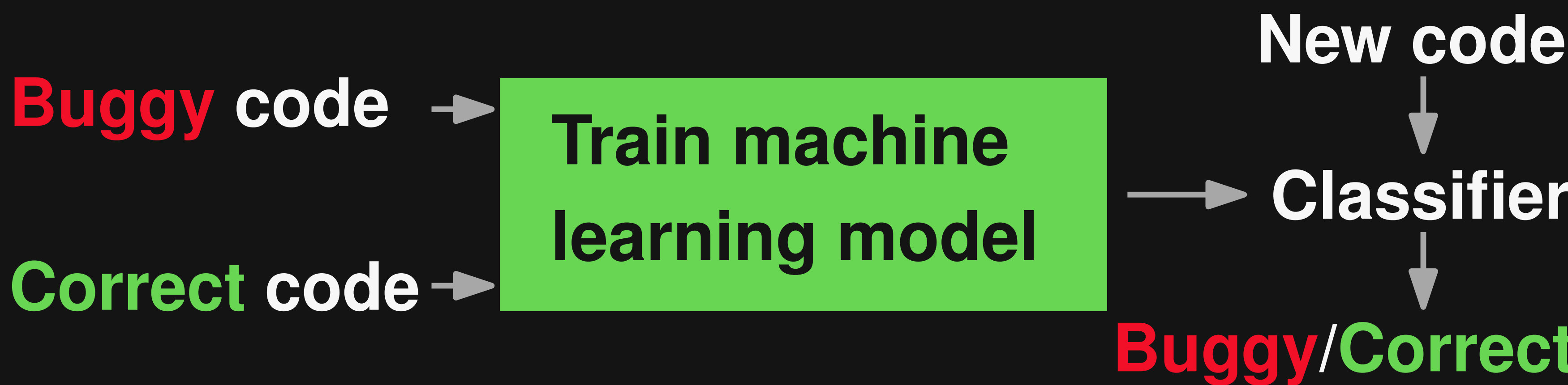
1) DeepBugs: Learning to find bugs

2) TypeWriter: Learning to predict types

Part of larger research agenda:
ERC Starting Grant on “Learning to Find Software Bugs”

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 → Train machine learning model

Correct code → Train machine learning model

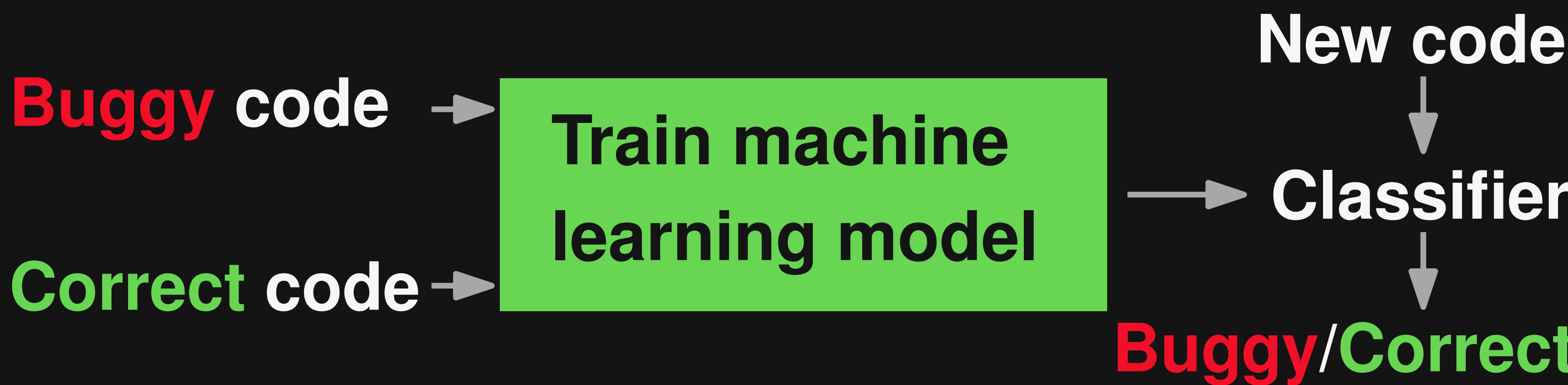
Train machine learning model → Classifier

New code → Classifier → Buggy/Correct

How to get the training data?

Learning to Find Bugs

Train a model to distinguish correct from buggy code



How to get the training data?

How to represent the code?

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) {
    ...
  }
}
```

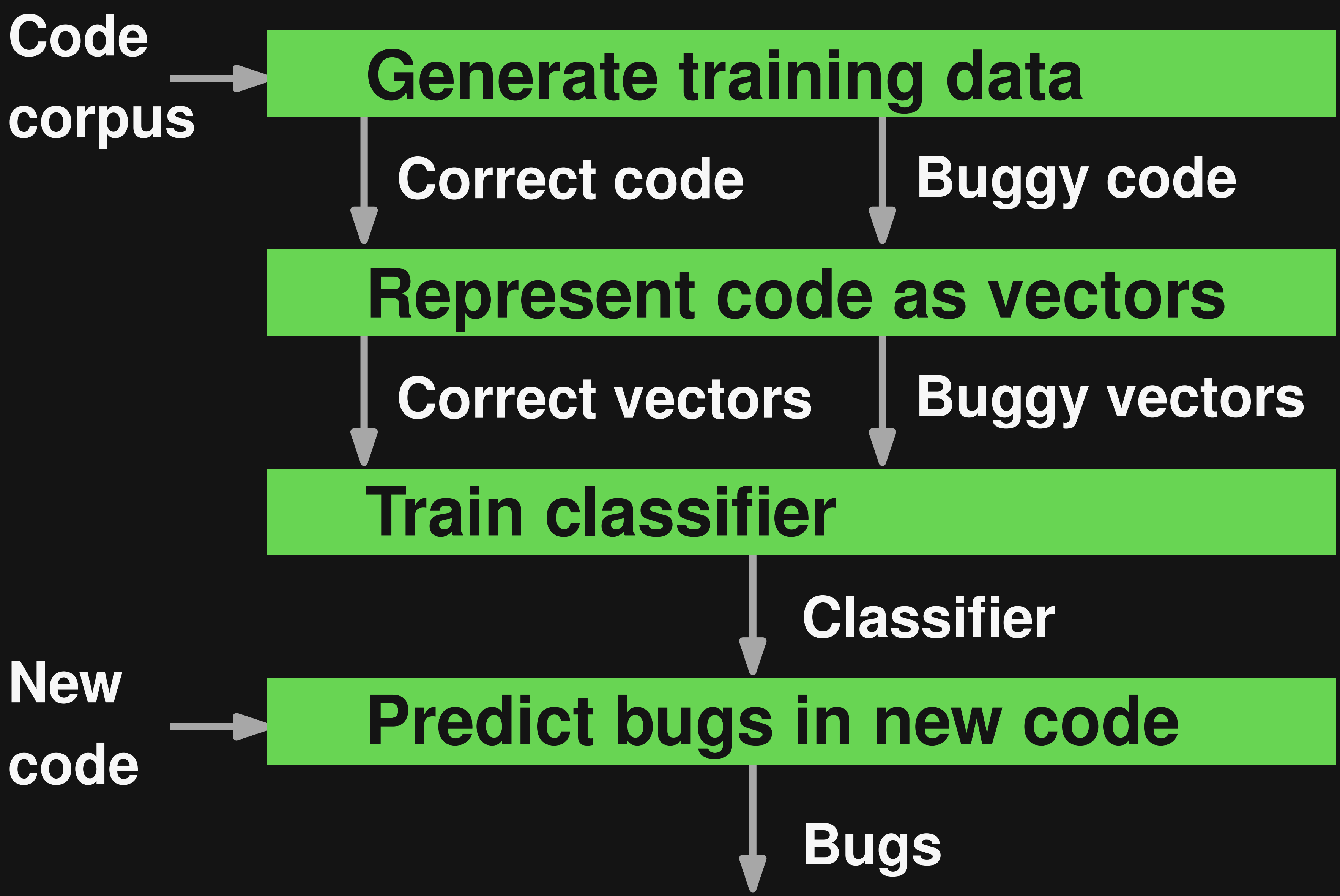
Name-related Bugs (2)

What's wrong with that code?

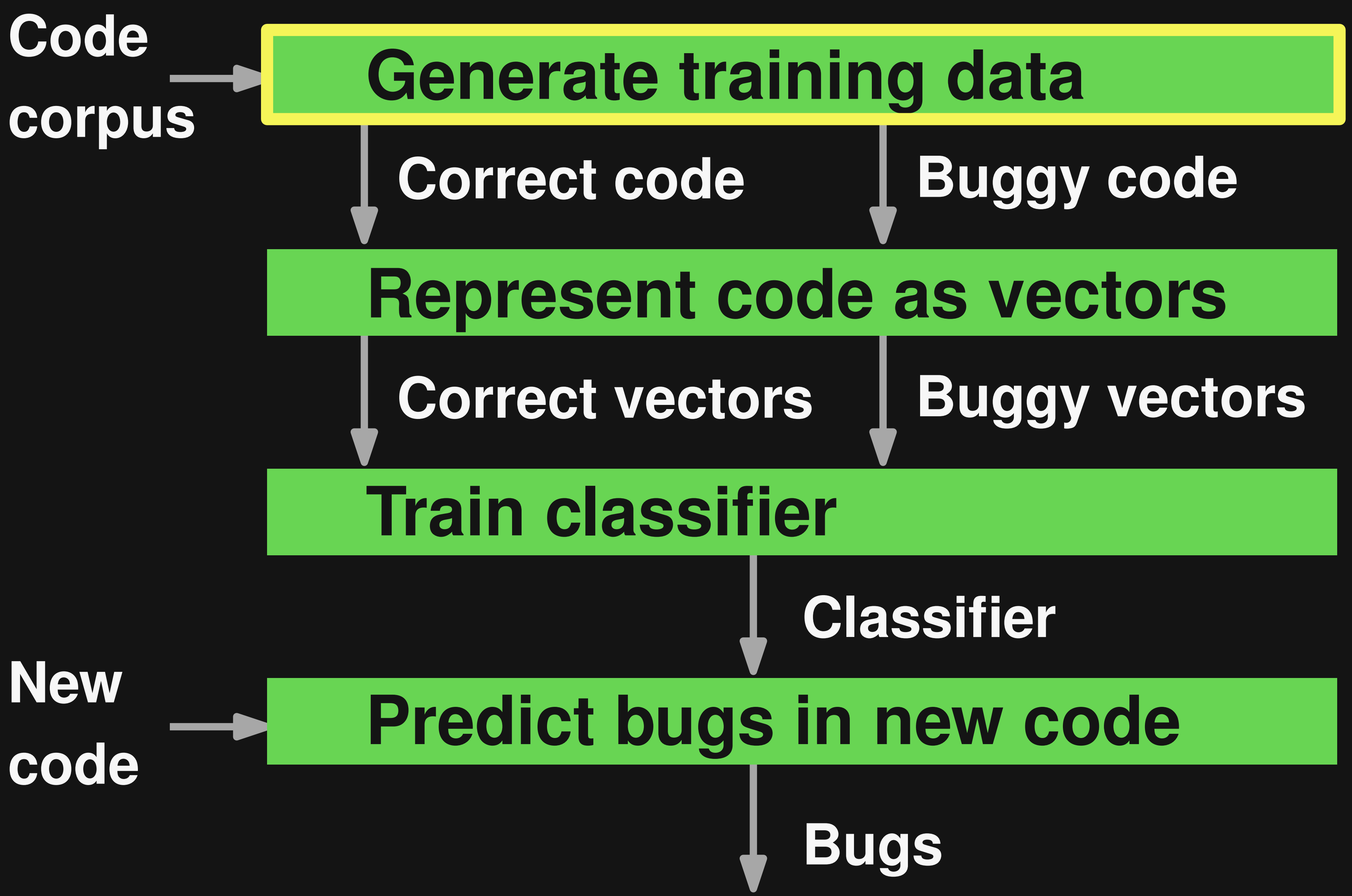
```
for (j = 0; j < params; j++) {
  if (params[j] == paramVal) {
    ...
  }
}
```

Should be `params.length`

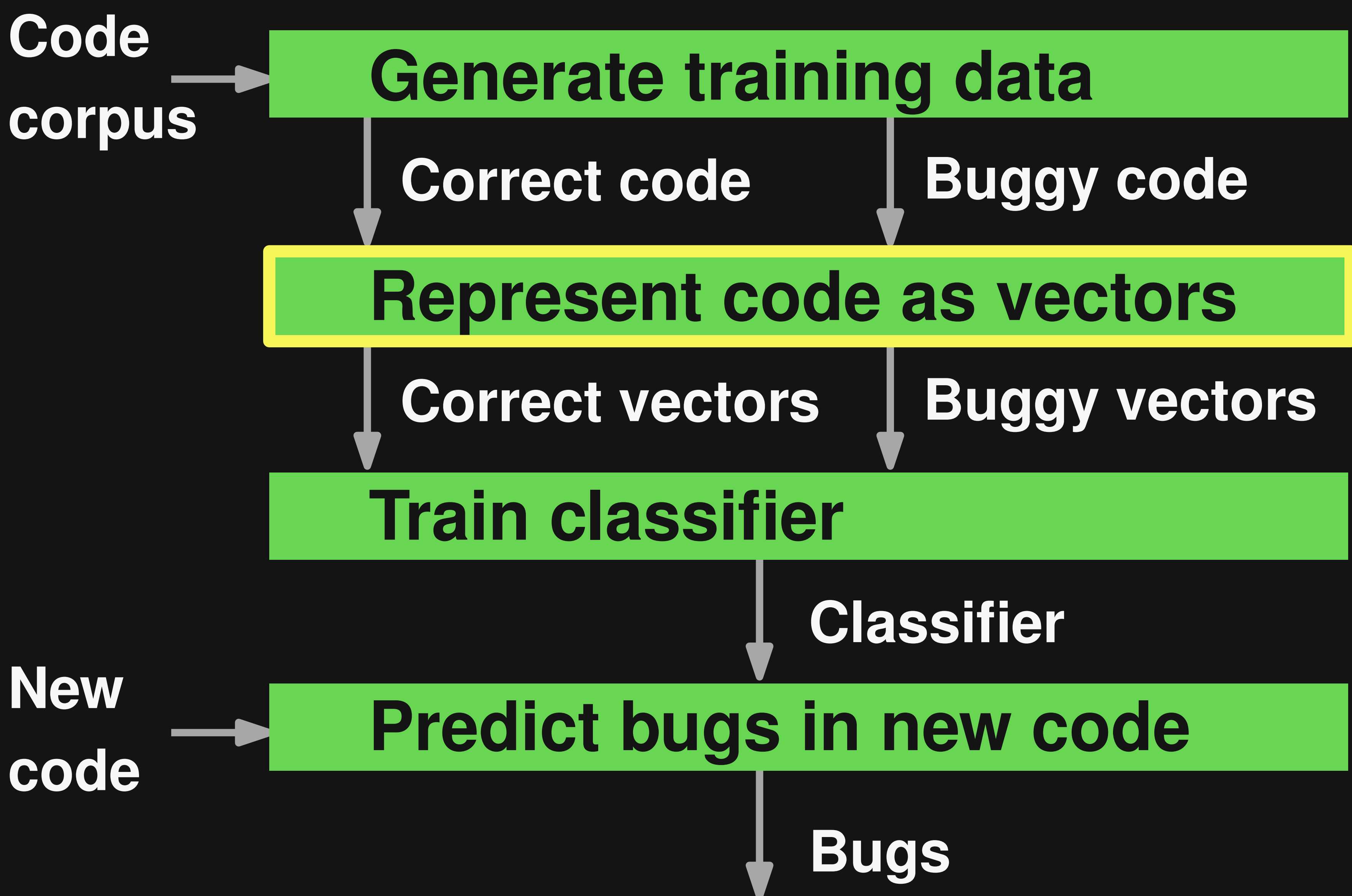
Overview of DeepBugs



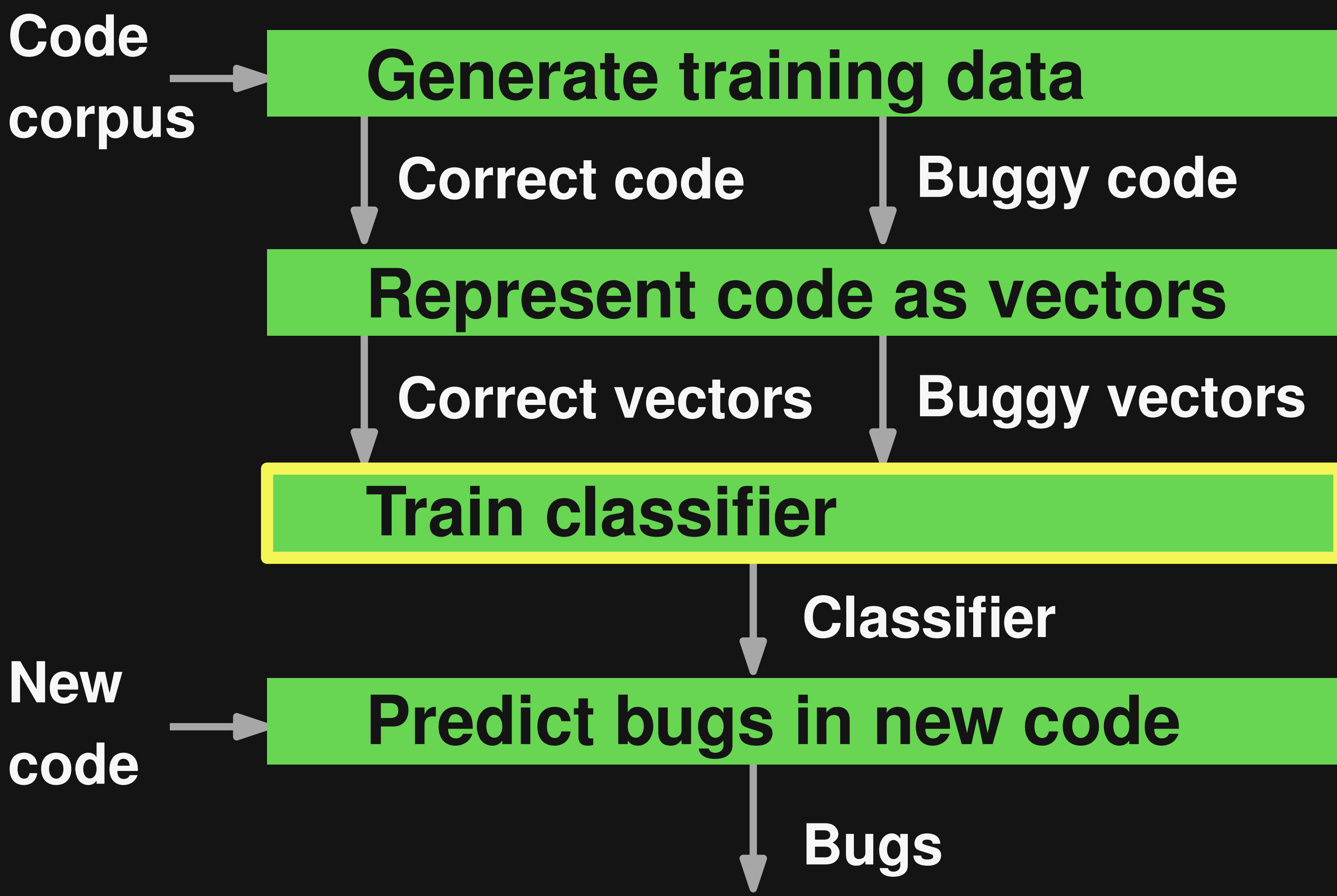
Overview of DeepBugs



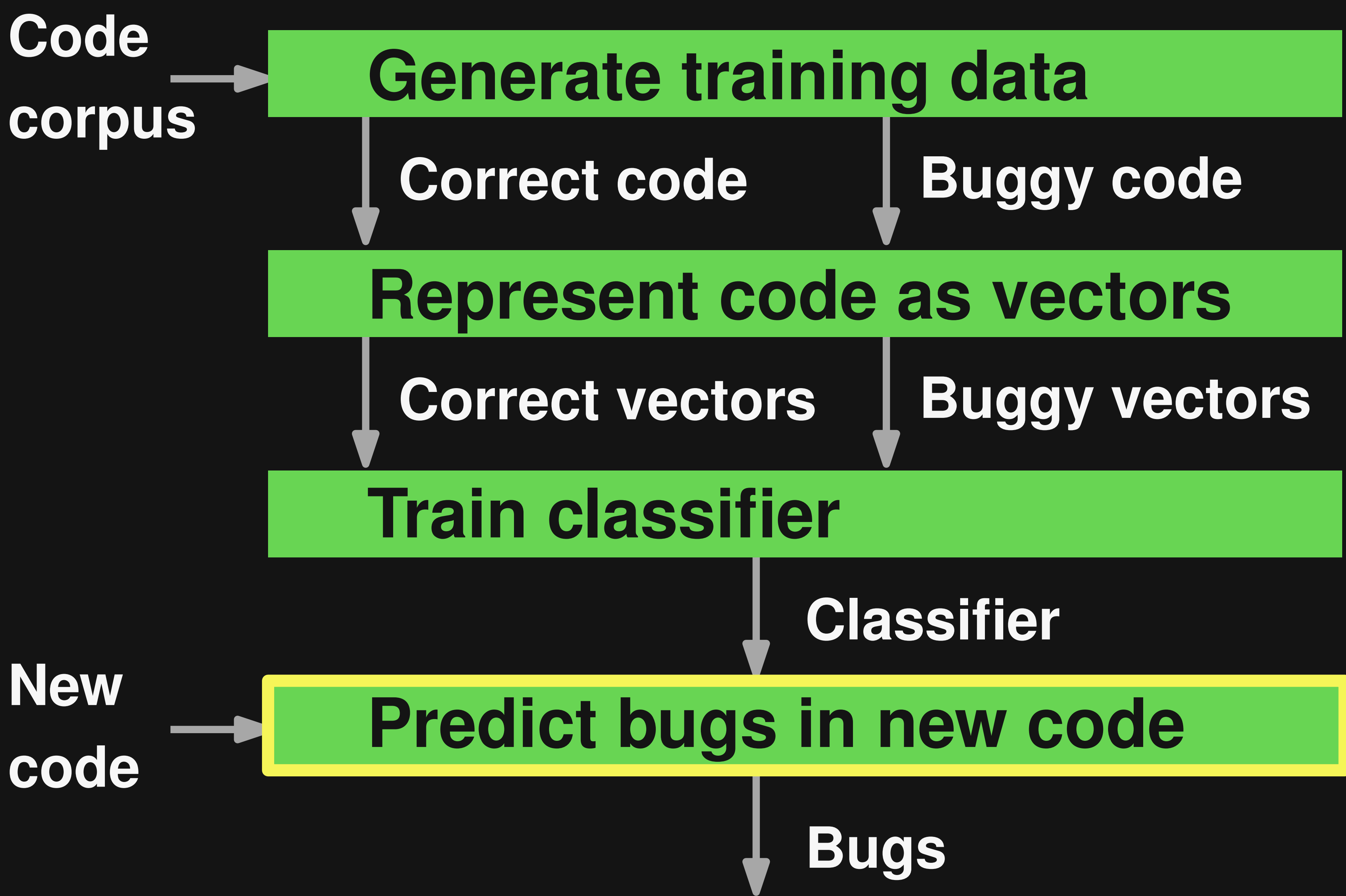
Overview of DeepBugs



Overview of DeepBugs



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
```

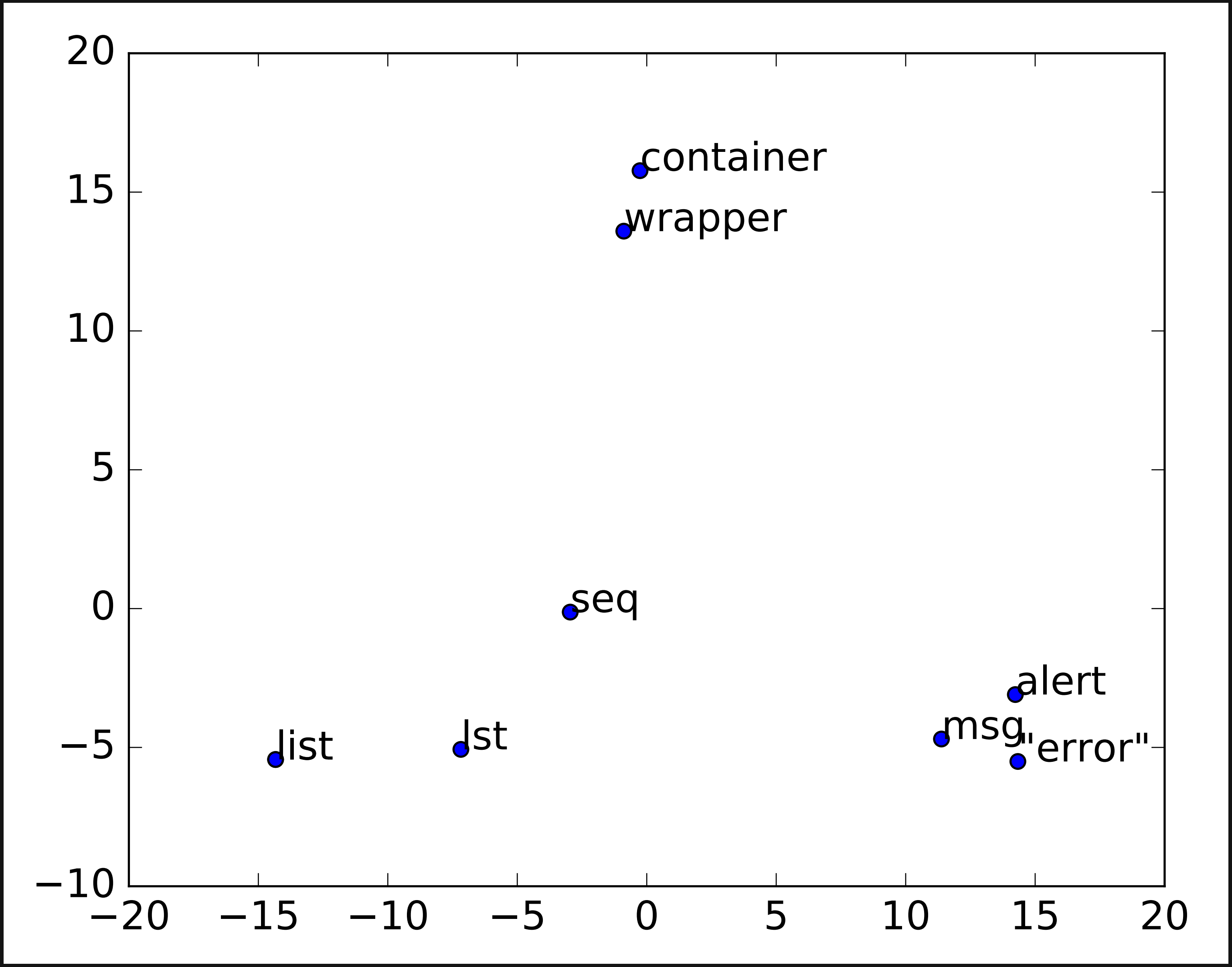
Randomly selected operand that occurs in same file

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 $\approx$ words)
  - Similar identifiers $\Rightarrow$ Similar vectors

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

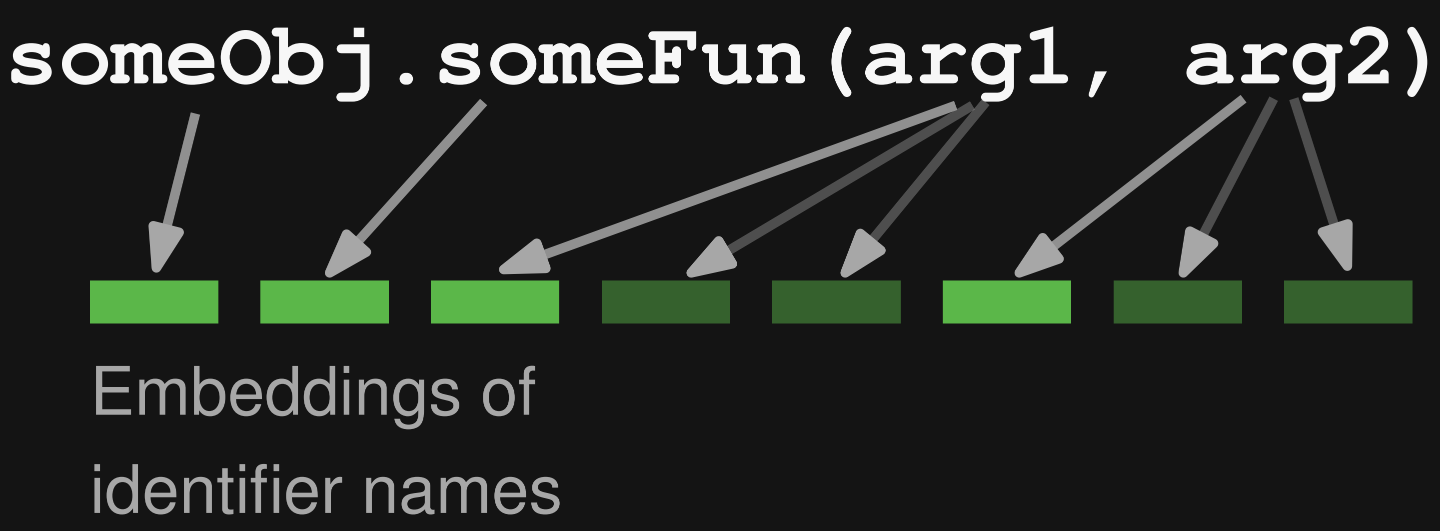
Example: Embeddings



Code Snippets as Vectors

Concatenate embeddings of names in code snippet

1) Swapped arguments

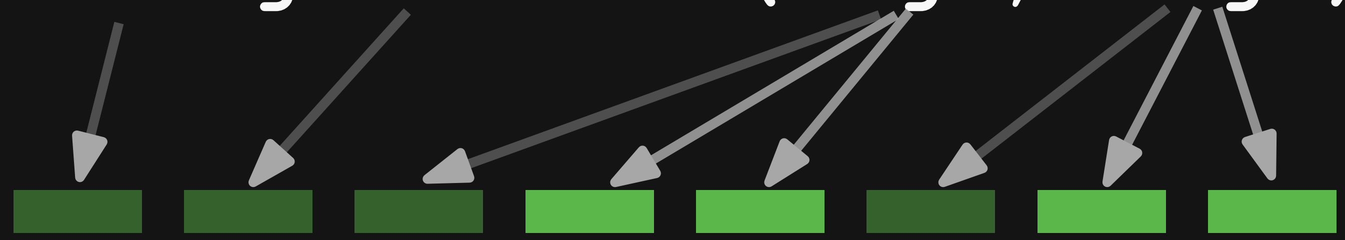


Code Snippets as Vectors

Concatenate embeddings of names in code snippet

1) Swapped arguments

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

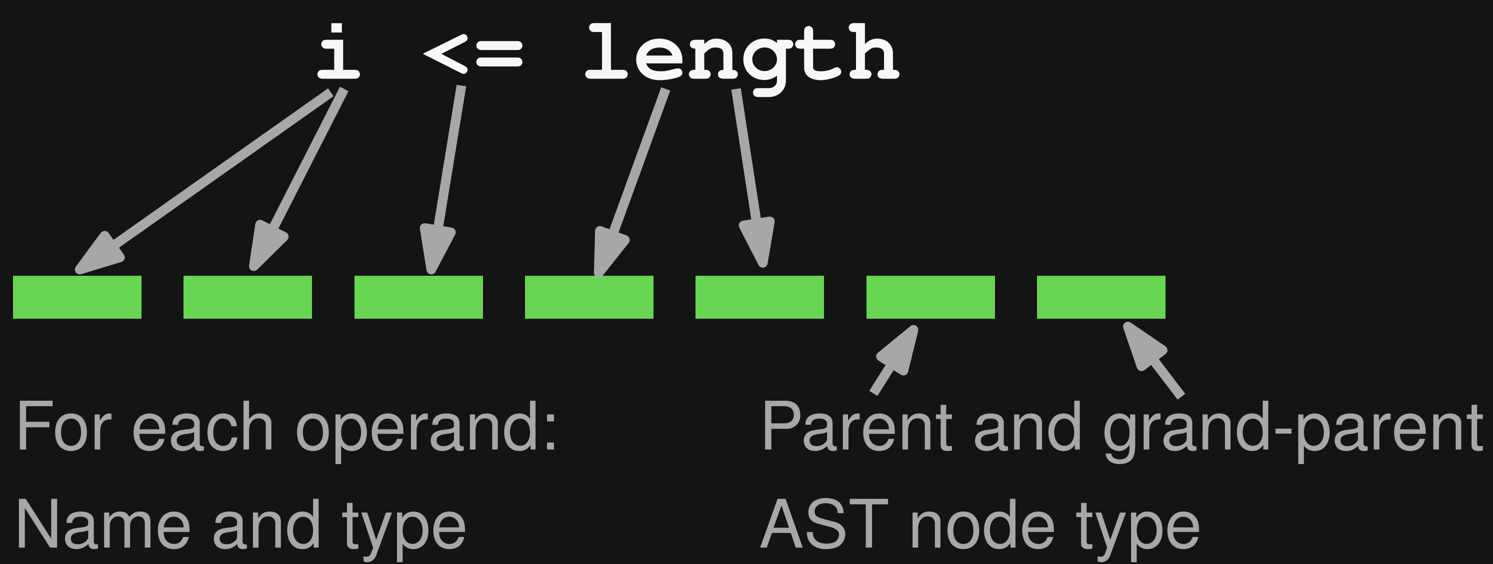


For each argument:
Type and formal parameter name

Code Snippets as Vectors

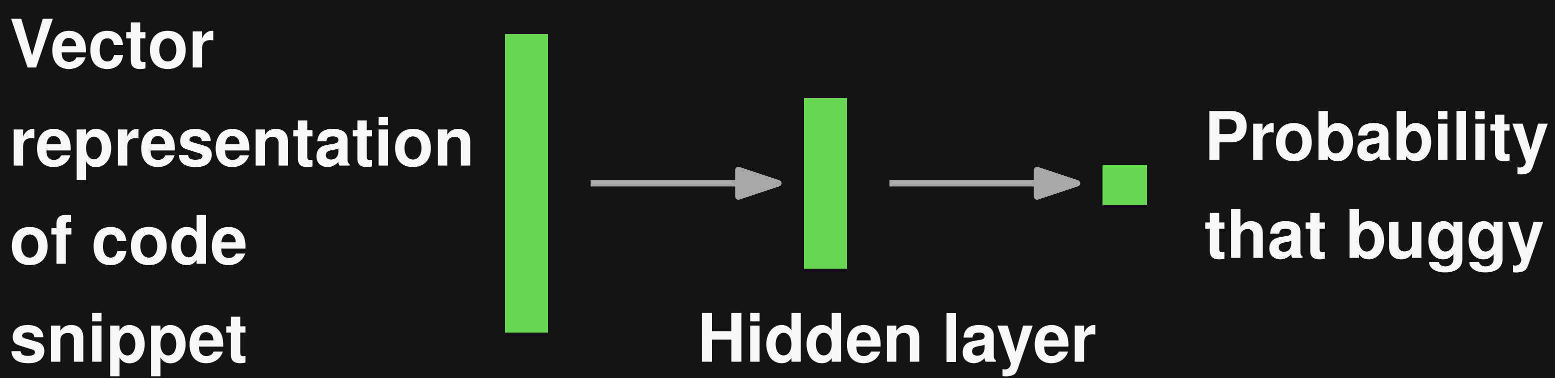
Concatenate embeddings of names in code snippet

2) + 3) Wrong binary operator/operation



Learning the Bug Detector

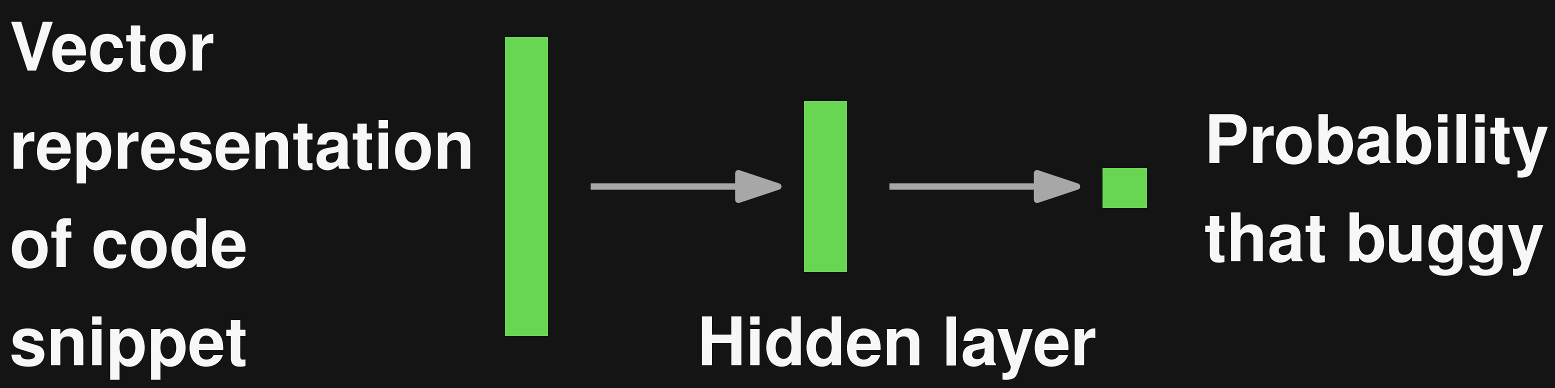
- **Given: Vector representation of code snippet**
- **Train neural network:**
  Predict whether correct or wrong



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 buggy

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

Summary: DeepBugs

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

Details: OOPSLA'18 paper

Code: https://github.com/michaelpradel/DeepBugs

This Talk

Two examples of learned developer tools *

1) DeepBugs: Learning to find bugs

2) TypeWriter: Learning to predict types

Part of larger research agenda:
ERC Starting Grant on "Learning to Find Software Bugs"

Why Infer Types?

- **Dynamically typed languages:**
  Extremely popular
- **Lack of type annotations:**
  - Type errors
  - Hard-to-understand APIs
  - Poor IDE support
- **Gradual types to the rescue**

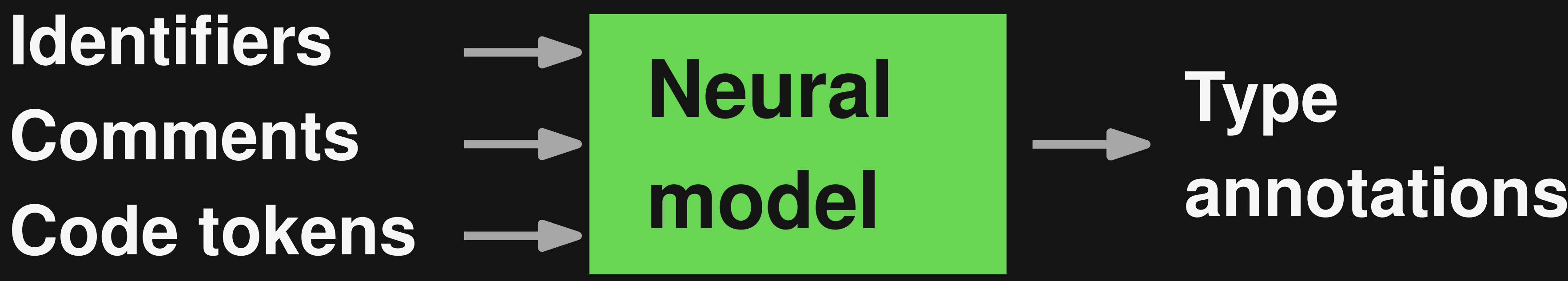
Why Infer Types?

- **Dynamically typed languages:**
  Extremely popular
- **Lack of type annotations:**
  - Type errors
  - Hard-to-understand APIs
  - Poor IDE support
- **Gradual types to the rescue**

But: Annotating types is painful

Probabilistic Type Prediction

E.g., neural model to predict types



Popular models:

- *Deep Learning Type Inference*, FSE'18
- *NL2Type: Inferring JavaScript Function Types from Natural Language Information*, ICSE'19

Example

```python
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Example

```python
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
  """
  Args:
    color (s                                  d return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Top-most predictions:
Type errors

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
  """
  Args:
    color (s                              d return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Correct predictions

Predictions:
1) int
2) $\boxed{\text{str}}$
3) bool

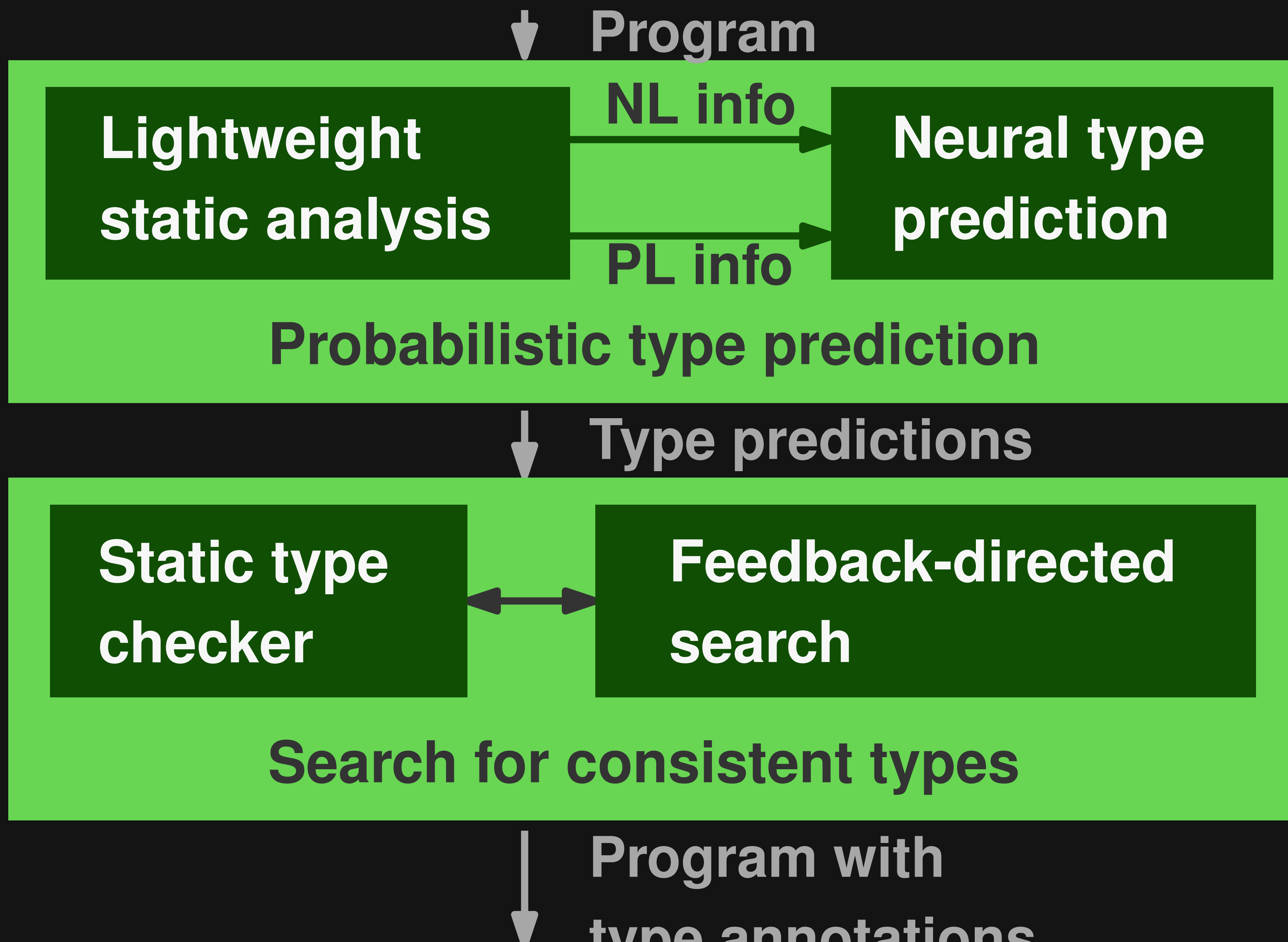
Predictions:
1) str
2) $\boxed{\text{Optional[str]}}$
3) None

Predictions:
1) $\boxed{\text{List[str]}}$
2) List[Any]
3) str

Challenges

- **Imprecision**
  - Some predictions are wrong
  - Developers must decide which suggestions to follow
- **Combinatorial explosion**
  - For each missing type: One or more suggestions
  - Exploring all combinations: Practically impossible

Overview of TypeWriter



Extracting NL and PL Info

- **NL information**
  - Names of functions and arguments
  - Function-level comments
- **PL information**
  - Occurrences of the to-be-typed code element
  - Types made available via imports

Extracting NL Information

```python
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting NL Information

Identifiers associated with the to-be-typed program element

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting NL Information

Function-level comments

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting PL Information

```python
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting PL Information

Tokens around occurrences of the to-be-typed code element

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting PL Information

Tokens around occurrences of the to-be-typed code element

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting PL Information

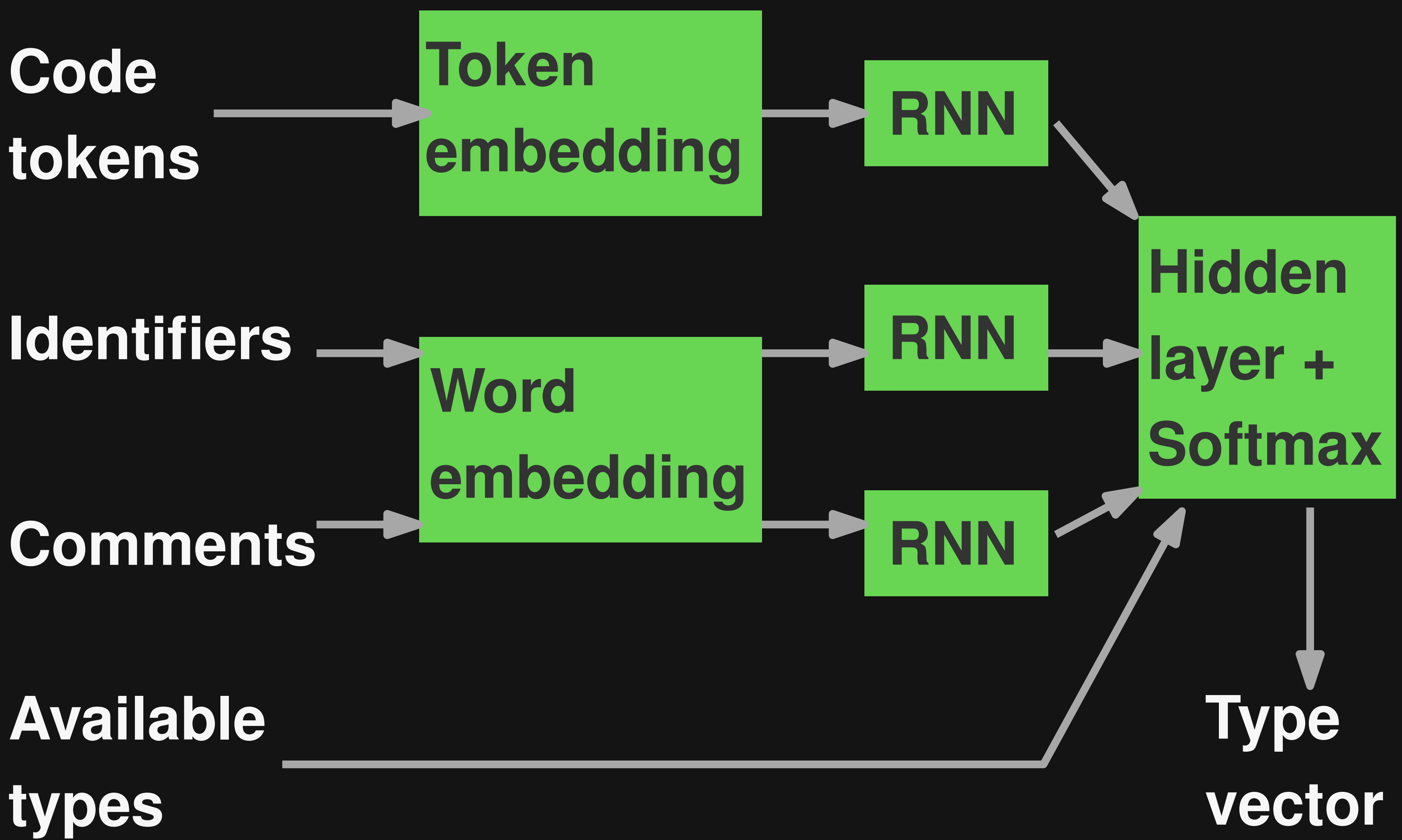
```
from ab import de
import x.y.z

def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Types made available via imports

Neural Type Prediction Model



Searching for Consistent Types

- **Top-k predictions for each missing type**
  - Filter predictions using gradual type checker
  - E.g., pyre and mypy for Python, flow for JavaScript
- **Combinatorial search problem**
  - For type slots S and k predictions per slot: $(k+1)^{|S|}$ possible type assignments

Searching for Consistent Types

- **Top-k predictions for each missing type**
  - Filter predictions using gradual type checker
  - E.g., pyre and mypy for Python, flow for JavaScript
- **Combinatorial search problem**
  - For type slots S and k predictions per slot: $(k+1)^{|S|}$ possible type assignments

Too large to explore exhaustively!

Exploring the Search Space

Tree of variants of program P

$$\begin{tikzcd}[column sep=small]
 & & P_{origTypes} \arrow[dl] \arrow[dr] & \\
 & P_{addedTypes1} \arrow[dl] \arrow[d] & & P_{addedTypes2} \\
P_{addedTypes3} & P_{addedTypes4} & &
\end{tikzcd}$$

$\longrightarrow$ **... add, remove, or replace types**

Exploring the Search Space

Tree of variants of program P

$$
\begin{tikzcd}
& & P_{origTypes} \arrow[dl] \arrow[dr] & \\
& P_{addedTypes1} \arrow[dl] \arrow[dr] & & P_{addedTypes2} \\
P_{addedTypes3} & & P_{addedTypes4} &
\end{tikzcd}
$$

Which variants to explore first?

$\longrightarrow$ **... add, remove, or replace types**

Feedback Function

- **Goal: Minimize missing types without introducing type errors**

- **Feedback score (lower is better):**

  $v \cdot n_{missing} + w \cdot n_{errors}$

Feedback Function

- Goal: **Minimize missing types** without introducing **type errors**

- **Feedback score** (lower is better):

$$v \cdot n_{missing} + w \cdot n_{errors}$$

Default: $v = 1$, $w = 2$,
i.e., higher weight for errors

Example

```python
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Example

```python
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Predictions:

1) int
2) str
3) bool

Predictions:

1) str
2) Optional[str]
3) None

Predictions:

1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Predictions:

1) int
2) str
3) bool

Predictions:

1) str
2) Optional[str]
3) None

Predictions:

1) List[str]
2) List[Any]
3) str

Evaluation: Setup

- **Code corpora**
  - Facebook's Python code
  - 5.8 millions lines of open-source code
- **Types**
  - Millions of argument and return types
  - 6-12% already annotated
  - Trivial types (e.g., type of `self`) ignored

Effectiveness of Neural Model

Approach	Top-1		
	Prec	Rec	F1
TypeWriter	65%	59%	62%

Open-source corpus; Combined prediction (arg. and return types)

Effectiveness of Neural Model

Approach	Top-1			Top-3			Top-5		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
TypeWriter	65%	59%	62%	80%	71%	75%	85%	75%	80%

Open-source corpus; Combined prediction (arg. and return types)

Effectiveness of Neural Model

Approach	Top-1			Top-3			Top-5		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
TypeWriter	65%	59%	62%	80%	71%	75%	85%	75%	80%
NL2Type	59%	55%	57%	73%	67%	70%	79%	71%	75%
Frequencies	12%	20%	15%	19%	35%	25%	22%	39%	28%

Open-source corpus; Combined prediction (arg. and return types)

Effectiveness of Search

Strategy	Top-k	Annotations	
		Type-correct	Ground truth match
Greedy search	1		
	3		
	5		
Non-greedy search	1		
	3		
	5		

Ground truth: 306 annotations in 47 fully annotated files

Exploring up to $7 \cdot |S|$ states

Effectiveness of Search

Strategy	Top-k	Annotations	
		Type-correct	Ground truth match
Greedy search	1	215 (70%)	194 (63%)
	3	230 (75%)	196 (64%)
	5	231 (75%)	198 (65%)
Non-greedy search	1	216 (71%)	195 (64%)
	3	178 (58%)	148 (48%)
	5	164 (54%)	141 (46%)

Ground truth: 306 annotations in 47 fully annotated files

Exploring up to $7 \cdot |S|$ states

Effectiveness of Search

Strategy	Top-k	Annotations	
		Type-correct	Ground truth match
Greedy search	1	215 (70%)	194 (63%)
	3	230 (75%)	196 (64%)
	5	231 (75%)	198 (65%)
Non-greedy search	1	216 (71%)	195 (64%)
	3	178 (58%)	148 (48%)
	5	164 (54%)	141 (46%)
Pyre Infer	—	106 (35%)	78 (25%)

Subsumes practically all types

Ground truth: 306 annotations in 47 fully annotated files

Exploring up to $7 \cdot |S|$ states

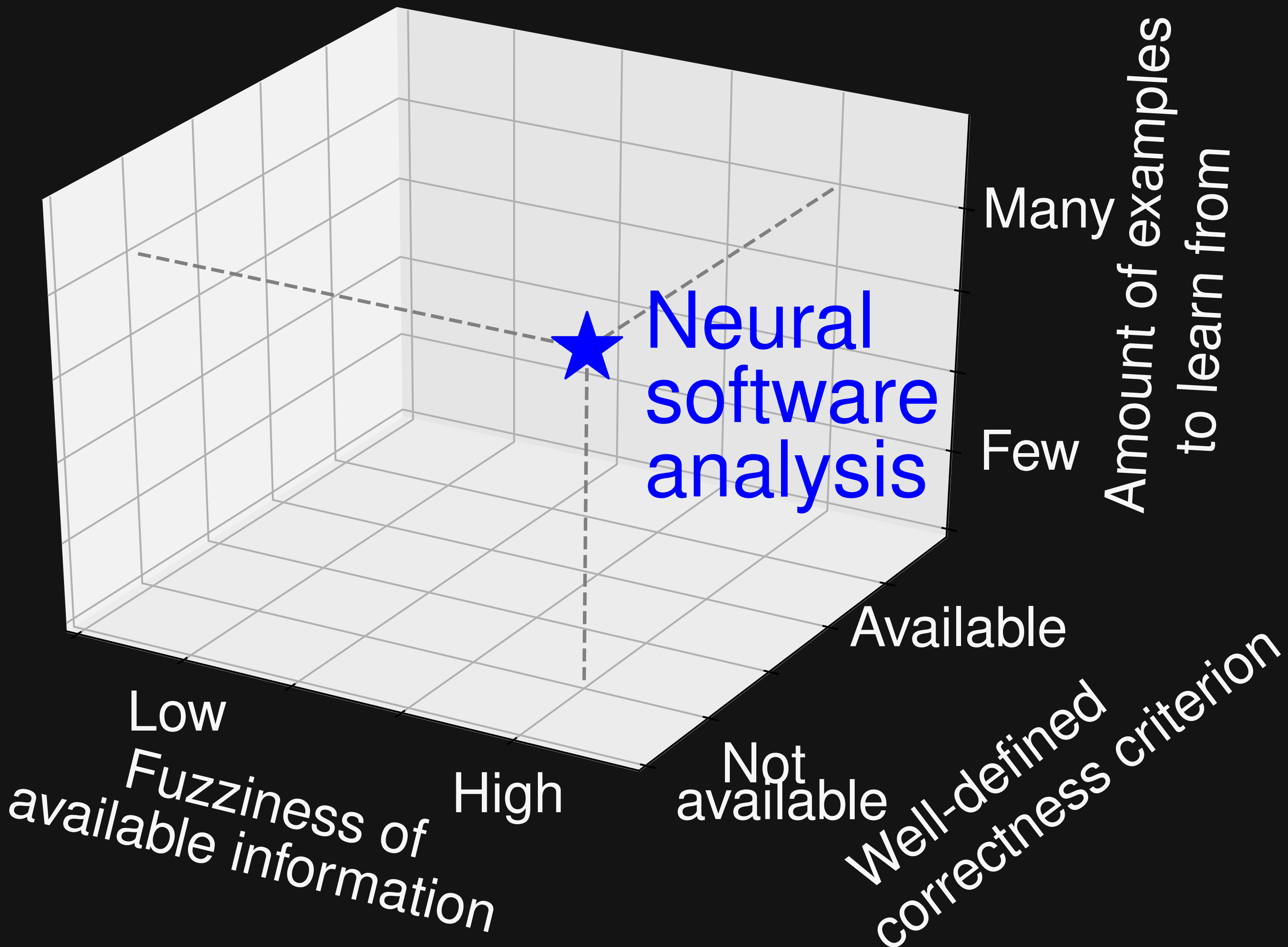
Summary: TypeWriter

Neural type prediction with search-based validation

- Probabilistic type prediction based on NL and PL information
- Ensure type correctness of added types via gradual type checker
- TypeWriter tool in use at Facebook

Neural Software Analysis

When to (not) use it?



Conclusion

Learning developer tools

- DeepBugs: Learning to find bugs
- TypeWriter: Learning to predict types
- Many open challenges
  - Better representations of software artifacts
  - Better models to reason about programs
  - Learn other kinds of developer tools
  - Understand predictions