Natural and Programming Language Processing

Kick-Off of the Stuttgart ELLIS Unit

Michael Pradel

Software Lab – University of Stuttgart



Natural and Programming Language Processing




Natural and Programming Language Processing


Developers Need Tools

Key feature of humans:

Ability to develop tools

Software development tools,

e.g., compilers, bug detection, code completion

Creating Developer Tools

Traditional program analysis

- Manually crafted
- Years of work
- Precise, logical reasoning
- Heuristics to handle undecidability
- Challenged by large code bases

Creating Developer Tools

Traditional program analysis		Neural software analysis
Manually crafted; Years of work	→	Automatically learned within hours
Precise, logical reasoning	→	Data-driven prediction
Heuristics to handle undecidability	→	Learn instead of hard-code heuristics
Challenged by large code bases	→	Use big code to our benefit

Neural Software Analysis

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

Source code
Execution traces
Documentation
Bug reports
etc.

→ **Machine Learning** → Predictive tool

Neural Software Analysis, Pradel & Chandra, CACM'22

Neural Software Analysis

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

Source code
Execution traces
Documentation
Bug reports
etc.

→ **Machine Learning** →

New code, execution, etc.

↓

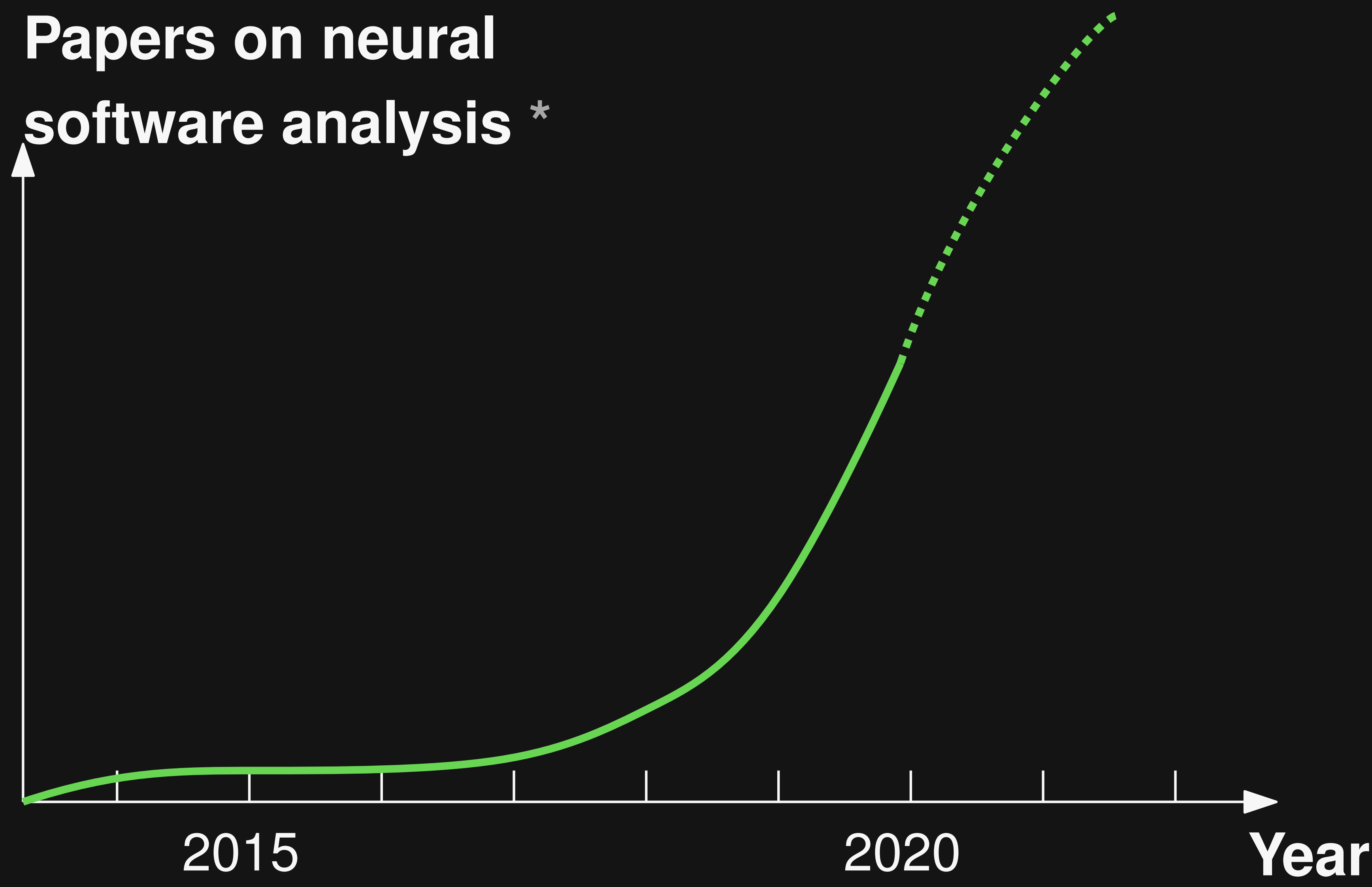
Predictive tool

↓

Information useful for developers

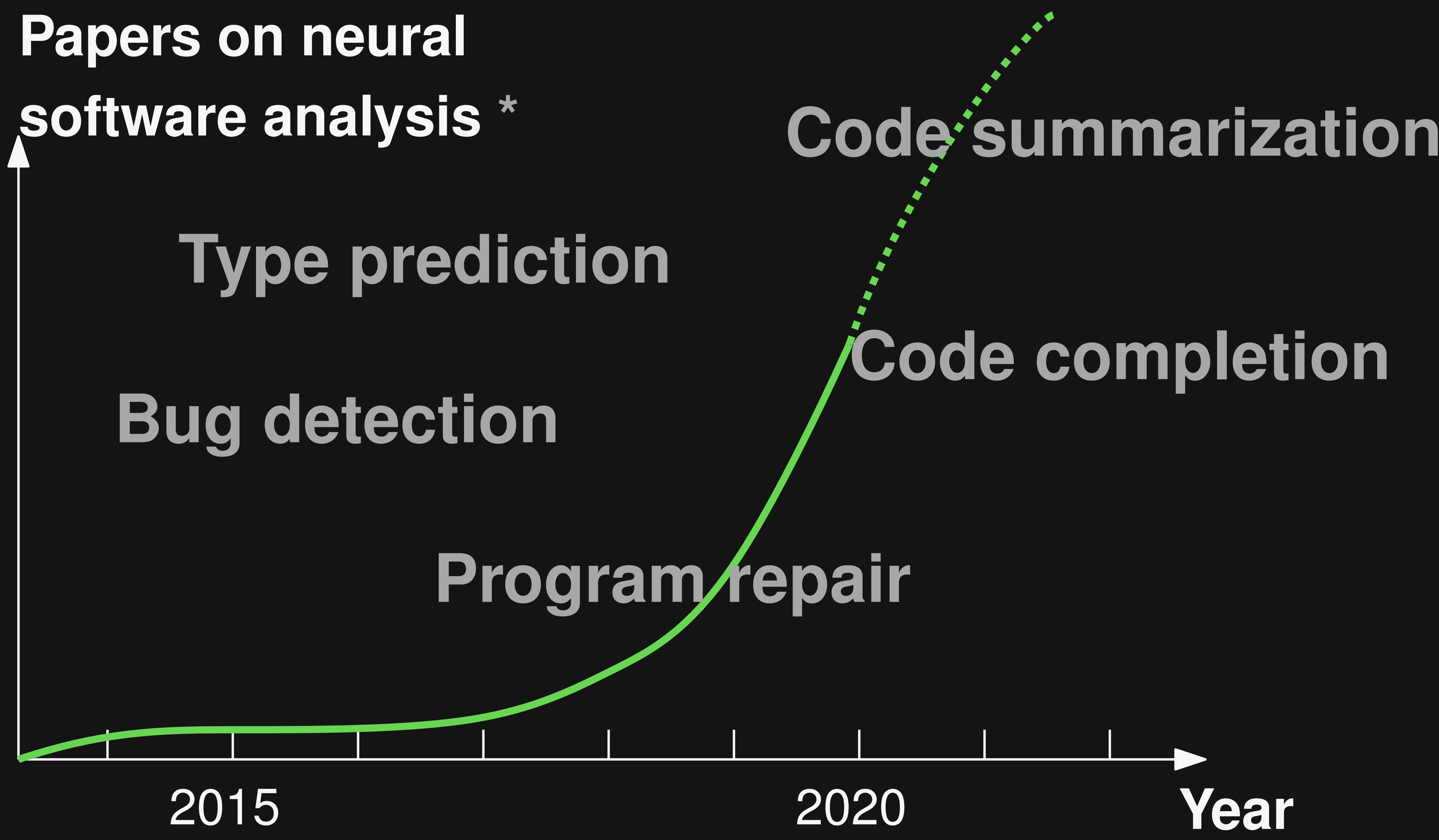
Neural Software Analysis, Pradel & Chandra, CACM'22

Join the Hype!



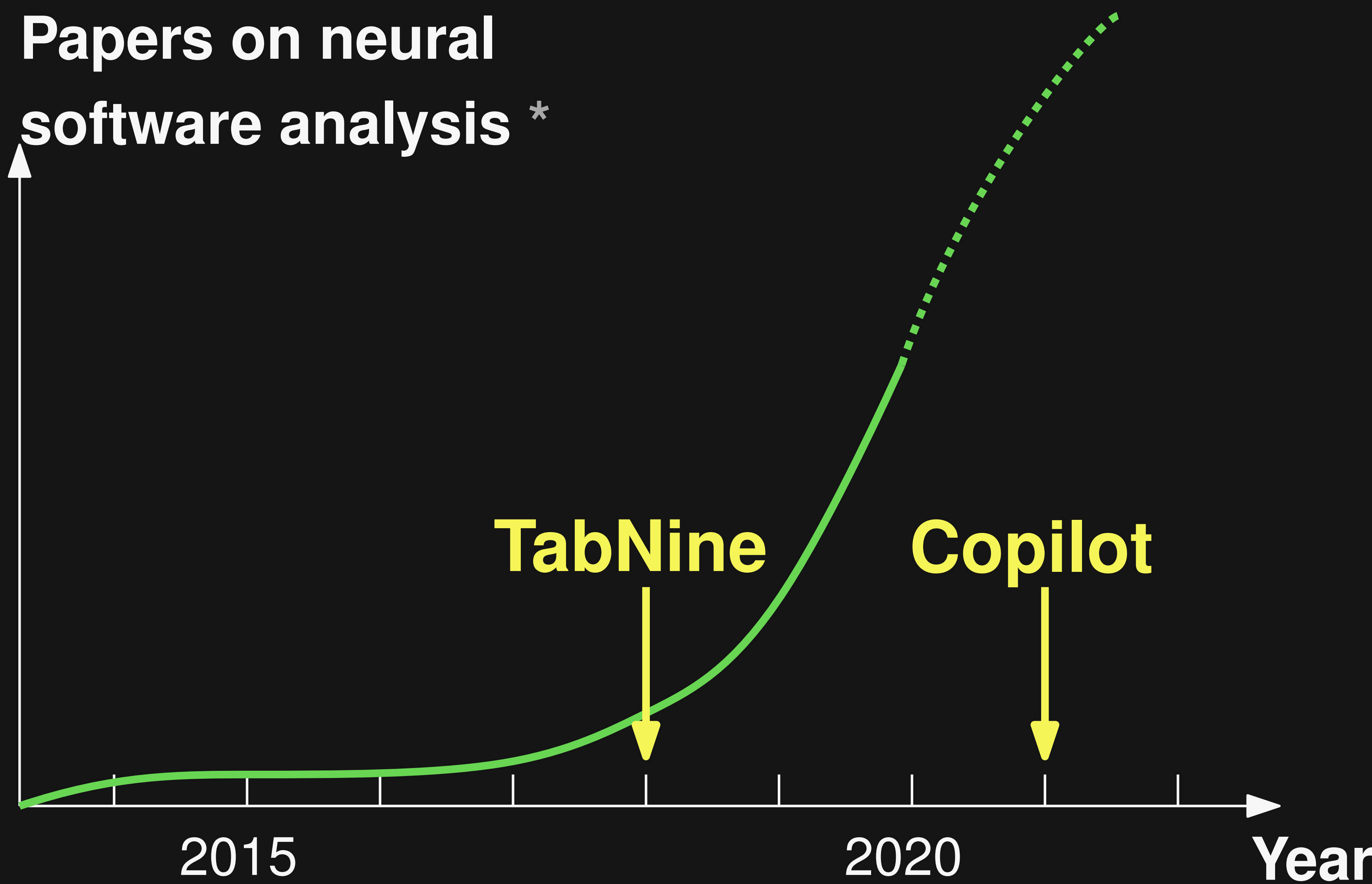
* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

Join the Hype!



* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

Join the Hype!



* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

This Talk

Examples of neural software analyses

1) Nalin: Name-value inconsistencies

2) TypeWriter: Type prediction

Open challenge

3) Understanding models of code

Motivation

```
train_size = 0.9 * iris.data.shape[0]
test_size = iris.data.shape[0] - train_size
train_data = data[0:train_size]
```

Motivation

Incorrect value: `135.0`, **should be** `135`

```
train_size = 0.9 * iris.data.shape[0]
test_size = iris.data.shape[0] - train_size
train_data = data[0:train_size]
```

Motivation

Incorrect value:

`135.0`, **should be** `135`

```
train_size = 0.9 * iris.data.shape[0]
test_size = iris.data.shape[0] - train_size
train_data = data[0:train_size]
```

```
file = os.path.exists('reference.csv')
if file == False:
  print('Warning: ...')
```

Motivation

Incorrect value:
`135.0`, **should be** `135`

```
train_size = 0.9 * iris.data.shape[0]
test_size = iris.data.shape[0] - train_size
train_data = data[0:train_size]
```

```
file = os.path.exists('reference.csv')
if file == False:
  print('Warning: ...')
```

Misleading name:
`file` **vs. boolean**

Motivation

Incorrect value:
`135.0`, **should be** `135`

```
train_size = 0.9 * iris.data.shape[0]
test_size = iris.data.shape[0] - train_size
train_data = data[0:train_size]
```

Commonality: Name and value are inconsistent

```
file = os.path.exists('reference.csv')
if file == False:
  print('Warning: ...')
```

Misleading name:
`file` **vs. boolean**

Goal

Finding name-value inconsistencies

Goal

Challenge 1:
Understand the
meaning of names

Finding name-value inconsistencies

Goal

Challenge 1:
Understand the
meaning of names

Challenge 2:
Understand the
meaning of values

Finding name-value inconsistencies

Goal

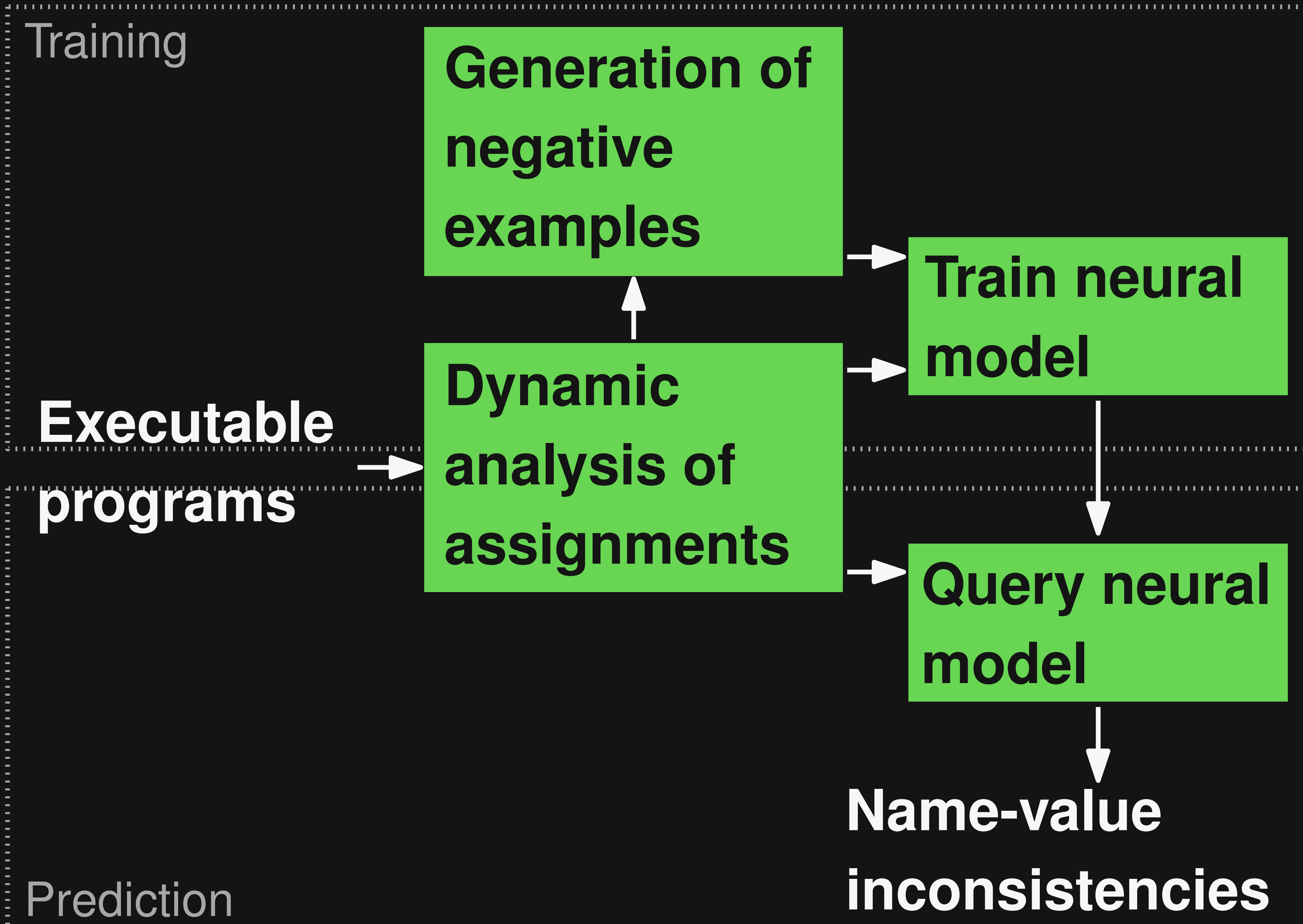
Challenge 1:
Understand the
meaning of names

Challenge 2:
Understand the
meaning of values

Finding name-value inconsistencies

Challenge 3:
Precisely pinpoint
unusual pairs

Overview of Nalin

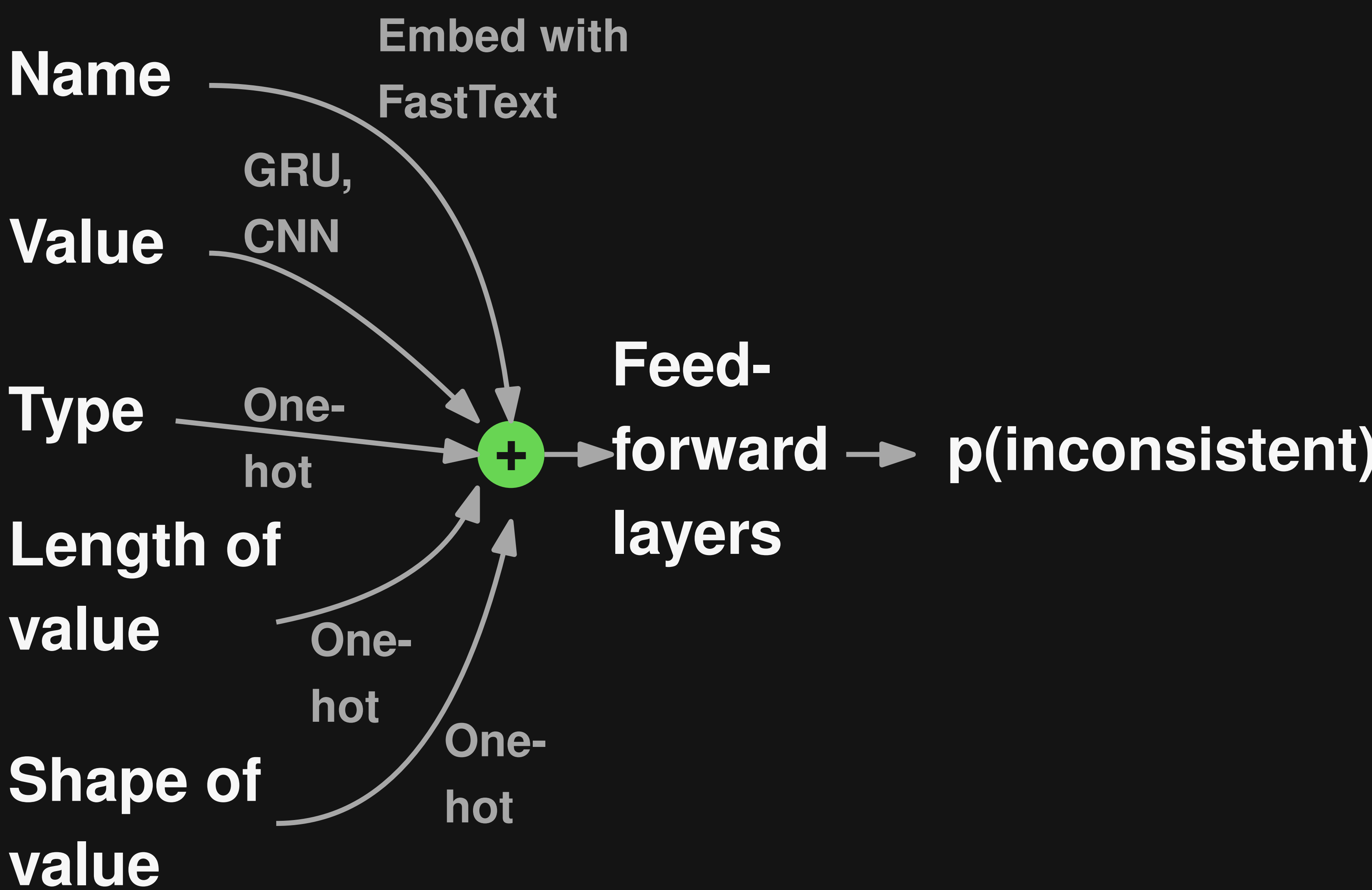


Analyzing Assignments

Data gathered via dynamic analysis:

Name	Value	Type	Length	Shape
age	23	int	null	null
probability	0.83	float	null	null
Xs_train	[[0.5 2.3]\n [..	ndarray	600	(600,2)
name	2.5	float	null	null
file_name	'example.txt'	str	11	null

Neural Classification Model



Two linear layers, 50% dropout, Adam optimizer, batch size=128

Evaluation

- **Experimental setup**
  - 947k name-value pairs (Jupyter notebooks)
- **Results**
  - Classifier: 89% F1-score
  - User study:

    Nalin points out hard-to-understand names
  - Complements static checkers

Kinds of Inconsistencies

30 inspected warnings

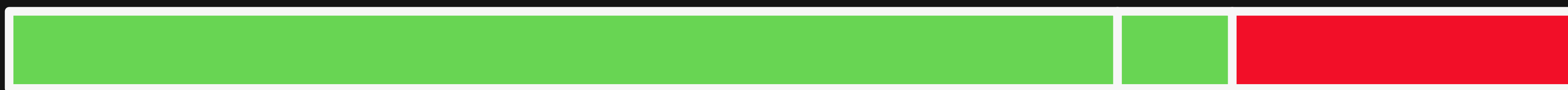
21 misleading names

2 incorrect values

7 false positives

Kinds of Inconsistencies

30 inspected warnings



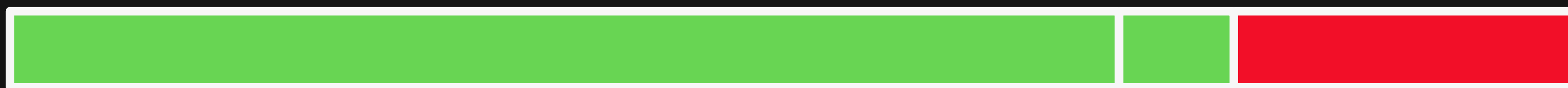
21 misleading names | **2 incorrect values** | **7 false positives**

```
name = 'Philip K. Dick'
...
name = 2.5
```

Unusual combination

Kinds of Inconsistencies

30 inspected warnings



21 misleading names

2 incorrect values

7 false positives

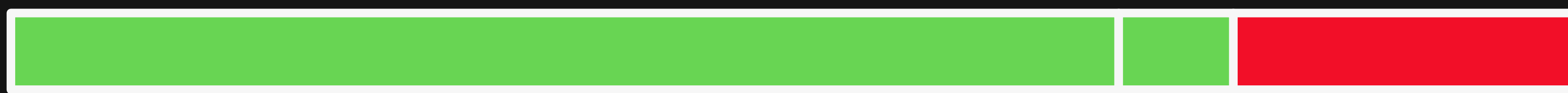
```
prob = get_betraying_probability(information)
if prob > 1/2:
  return D
```

Value: `"Corporate"`

Nalin: Learning from Runtime Behavior to Find Name-Value Inconsistencies in Jupyter Notebooks, ICSE'22

Kinds of Inconsistencies

30 inspected warnings



21 misleading names | **2 incorrect values** | **7 false positives**

```
dwarF = '/Users/iayork/Downloads/dwar\_2013\_2015.txt'
dwar = pd.read_csv(dwarF, sep=' ', header=None)
```

Model doesn’t understand the abbreviation (“F” means “file”)

Wouldn't a type checker find some of these problems?

Wouldn't a type checker find some of these problems?

Yes, but: Lots of code has no type annotations

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

Parameter type?

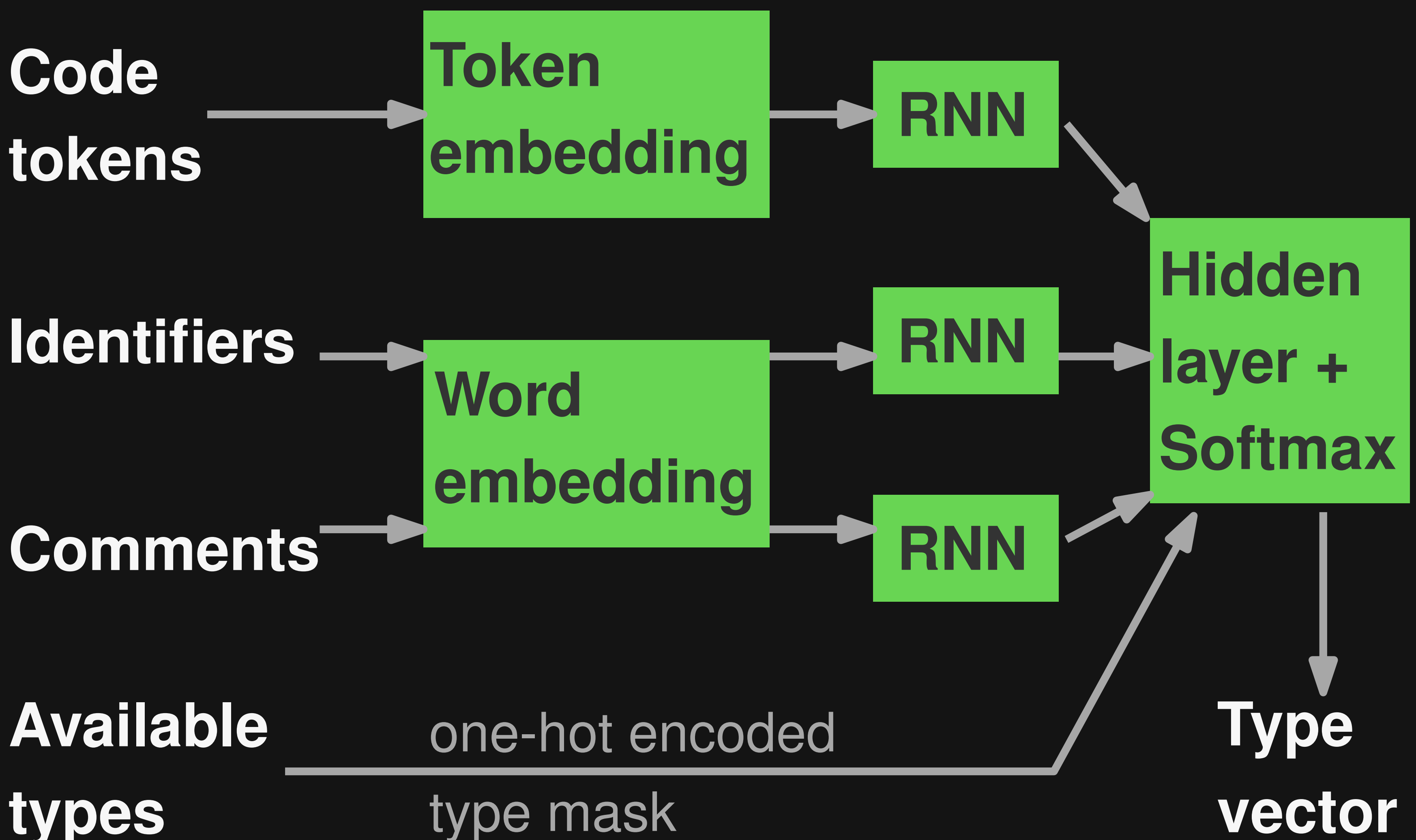
```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

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

Return type?

Return type?

TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

Neural Type Prediction Model



TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

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

TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

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

TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

Searching for Consistent Types

- **Top-k predictions for each missing type**
  - Filter predictions using gradual type checker (e.g., mypy or pyre)
- **Combinatorial search problem**
  - Feedback-directed search: Minimize type errors, maximize type annotations

TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

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

```
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

TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

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

TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

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

TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

Evaluation

- **Experimental setup**
  - Facebook's Python code
  - 5.8 millions lines of open-source code
- **Results**
  - Neural model:

    80% F1-score (top-5, individual annotations)
  - Neural model + search:

    Correctly adds 75% all annotations in a file
  - Subsumes traditional static type inference

Why Does It Work?

Developers use **meaningful names**

Source code is **repetitive**

Many programs available as **training data**

Probabilistic models + NL = ♡

What are these models actually learning?

Idea: Compare Humans & Models

Developers

vs.

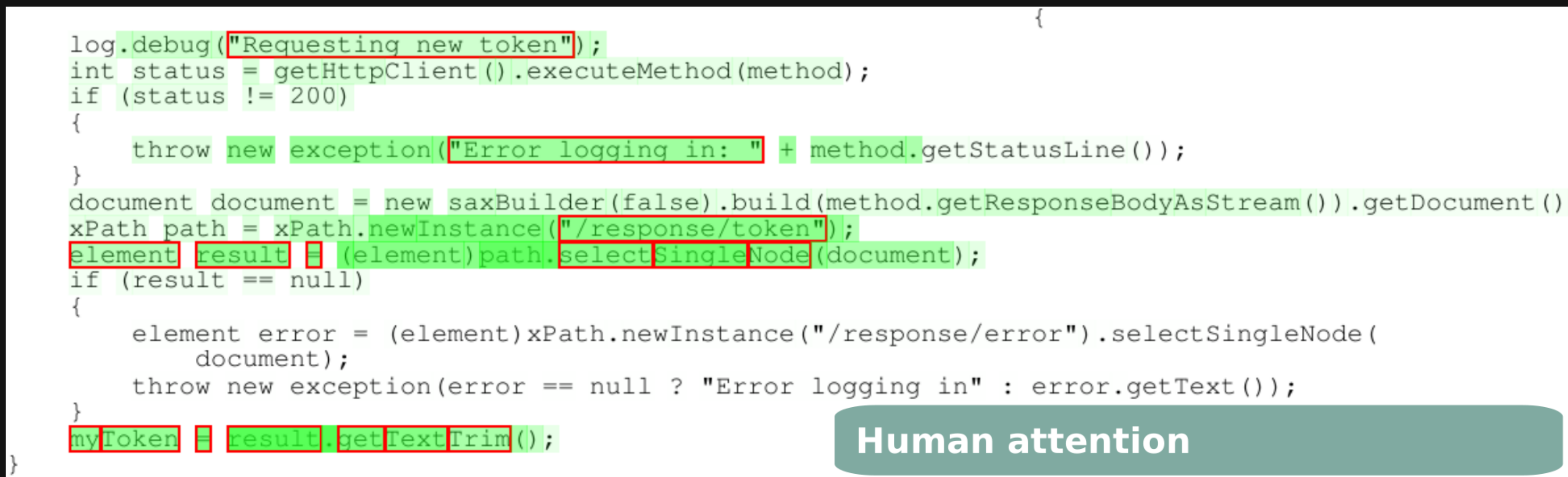
Machine Learning

Neural models of code

- **Same task**
- **Same code examples**
- **Measure attention and effectiveness**

Thinking Like a Developer? Comparing the Attention of Humans with Neural Models of Code, ASE'21

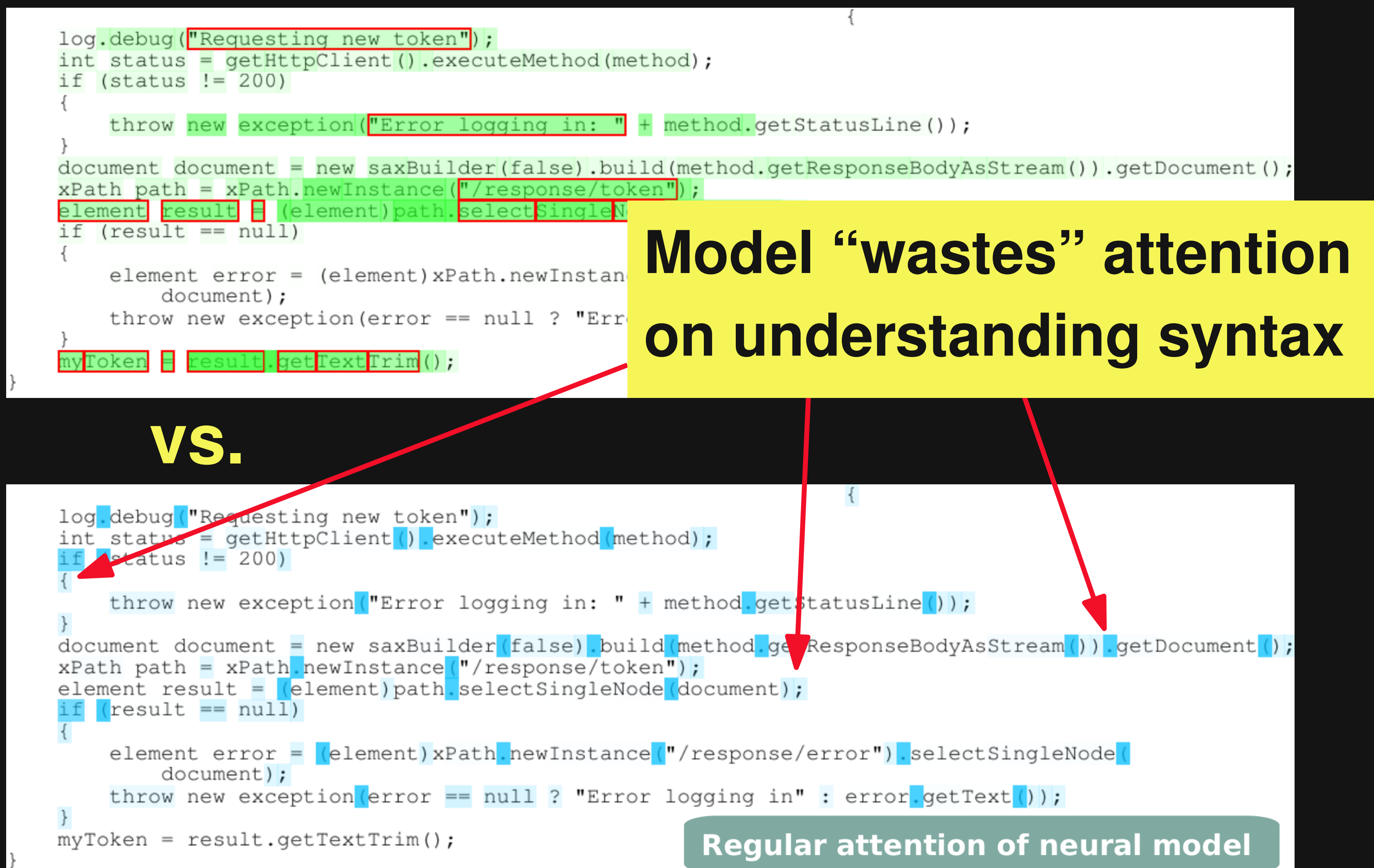
Human vs. Model Attention



vs.

Thinking Like a Developer? Comparing the Attention of Humans with Neural Models of Code, ASE'21

Human vs. Model Attention



Thinking Like a Developer? Comparing the Attention of Humans with Neural Models of Code, ASE’21

Human vs. Model Attention

```
                                                                                   {
    log.debug("Requesting new token");
    int status = getHttpClient().executeMethod(method);
    if (status != 200)
    {
        throw new exception("Error logging in: " + method.getStatusLine());
    }
    document document = new saxBuilder(false).build(method.getResponseBodyAsStream()).getDocument();
    xPath path = xPath.newInstance("/response/token");
    element result = (element)path.selectSingleNode(document);
    if (result == null)
    {
        element error = (element)xPath.newInstance("/response/error").selectSingleNode(
            document);
        throw new exception(error == null ? "Err
    }
    myToken = result.getTextTrim();
}
```

vs.

Model ignores tokens important to developers

```
    log.debug("Requesting new token");
    int status = getHttpClient().executeMethod(method);
    if (status != 200)
    {
        throw new exception("Error logging in: " + method.getStatusLine());
    }
    document document = new saxBuilder(false).build(method.getResponseBodyAsStream()).getDocument();
    xPath path = xPath.newInstance("/response/token");
    element result = (element)path.selectSingleNode(document);
    if (result == null)
    {
        element error = (element)xPath.newInstance("/response/error").selectSingleNode(
            document);
        throw new exception(error == null ? "Error logging in" : error.getText());
    }
    myToken = result.getTextTrim();
}
```

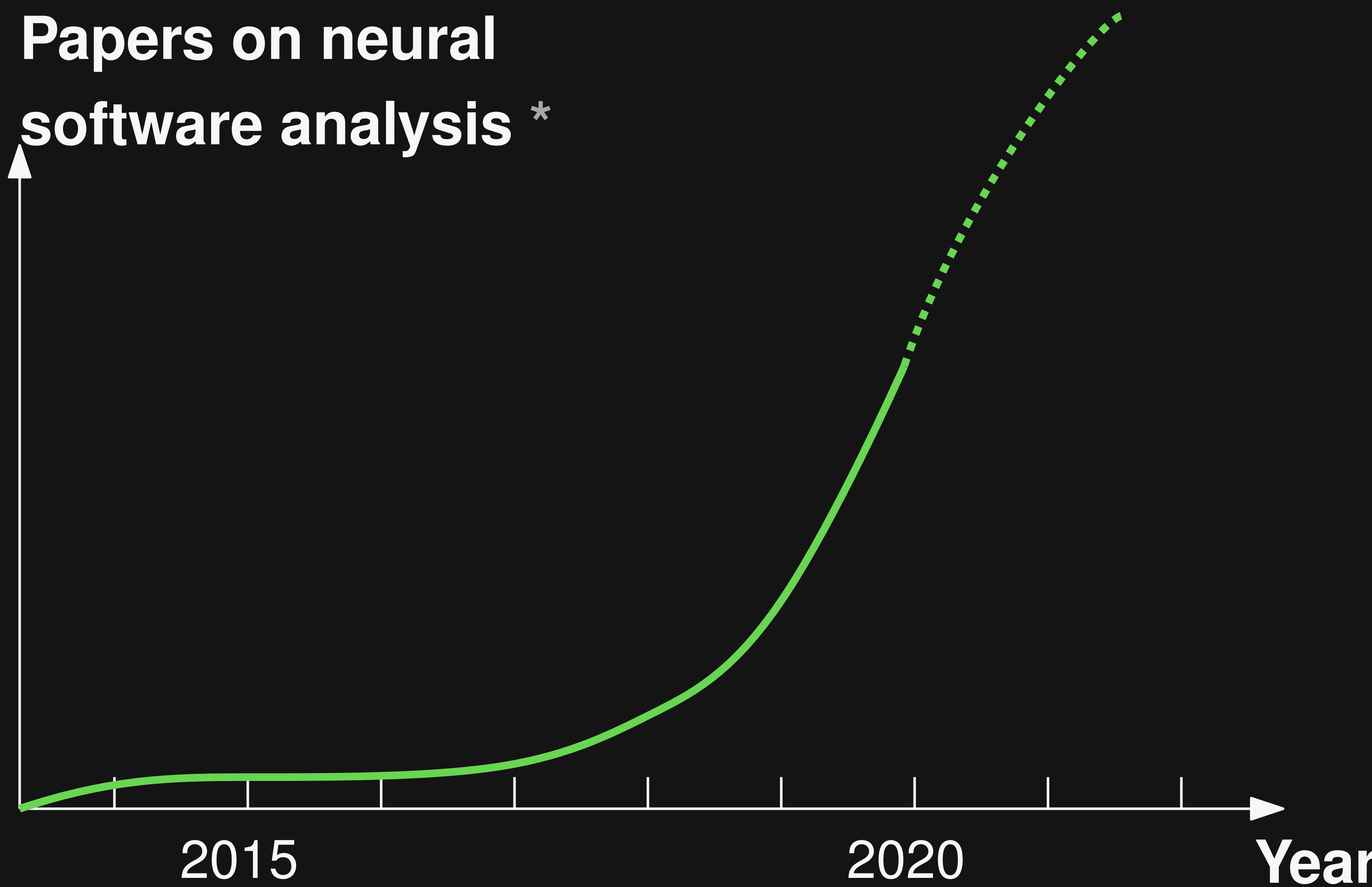
Regular attention of neural model

Thinking Like a Developer? Comparing the Attention of Humans with Neural Models of Code, ASE'21

Findings & Implications

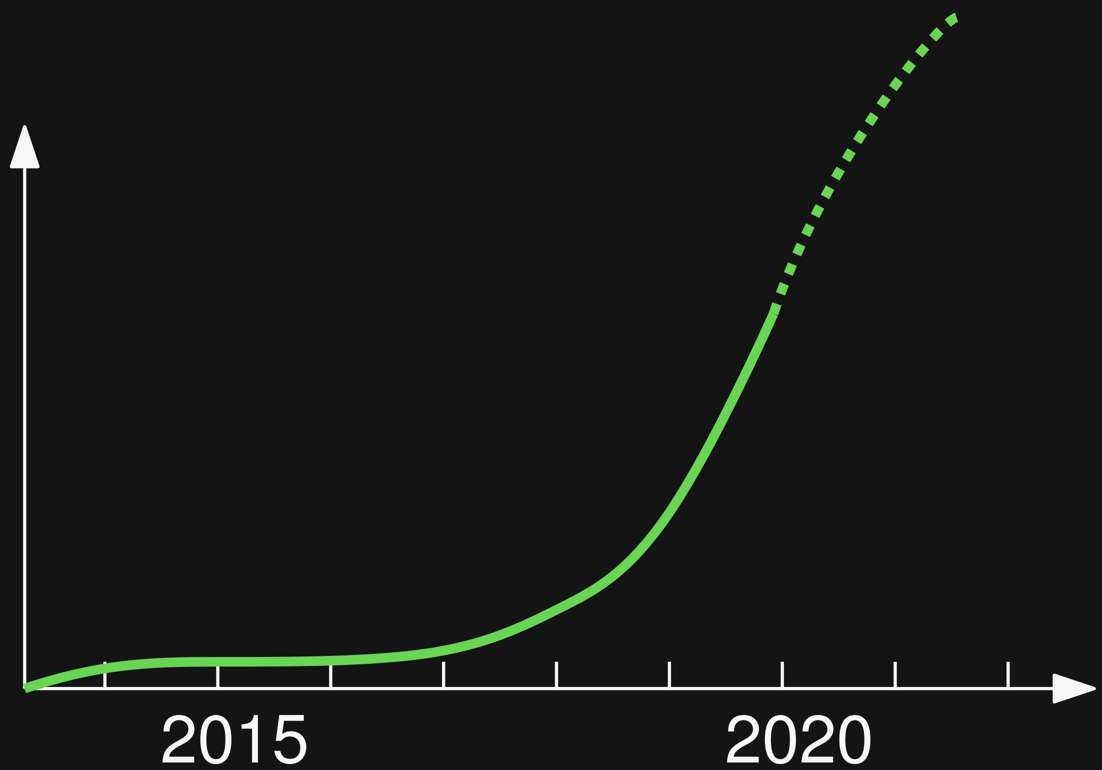
- **Findings**
  - Only partial agreement on what code matters
  - Higher agreement correlates with higher model accuracy
- **Implications**
  - Direct human-model comparison: Helps understand why models (do not) work
  - Should create models that mimic humans

The Road Ahead



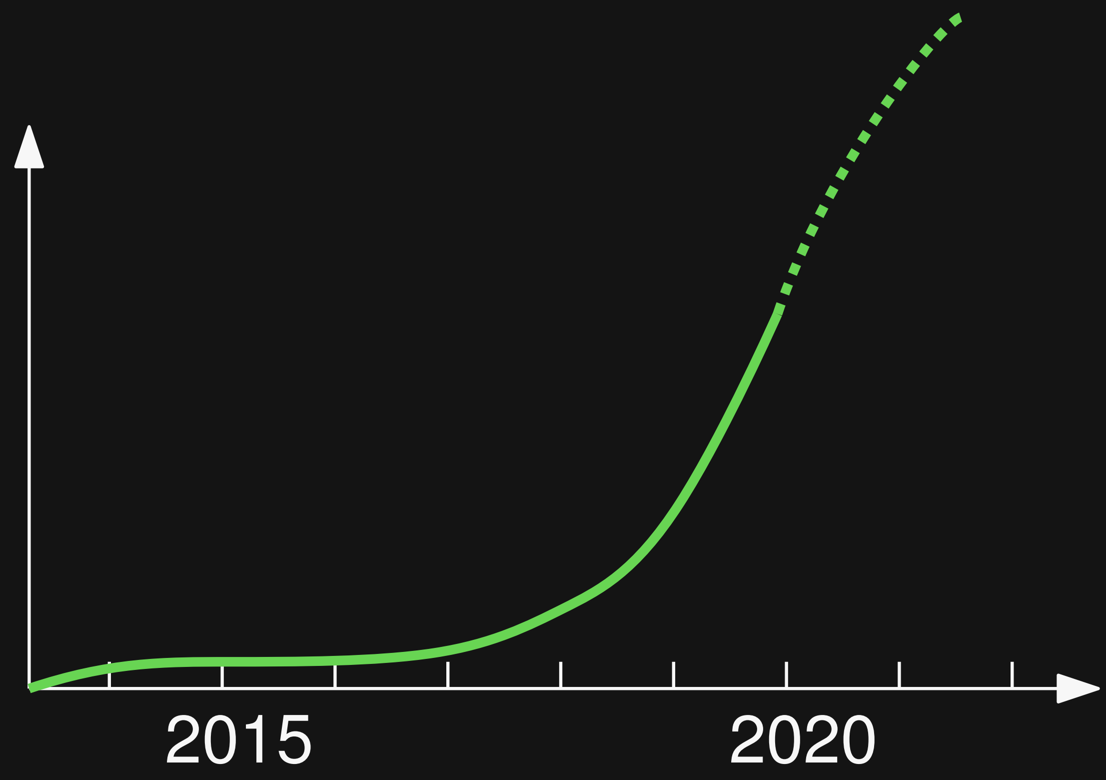
* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

The Road Ahead

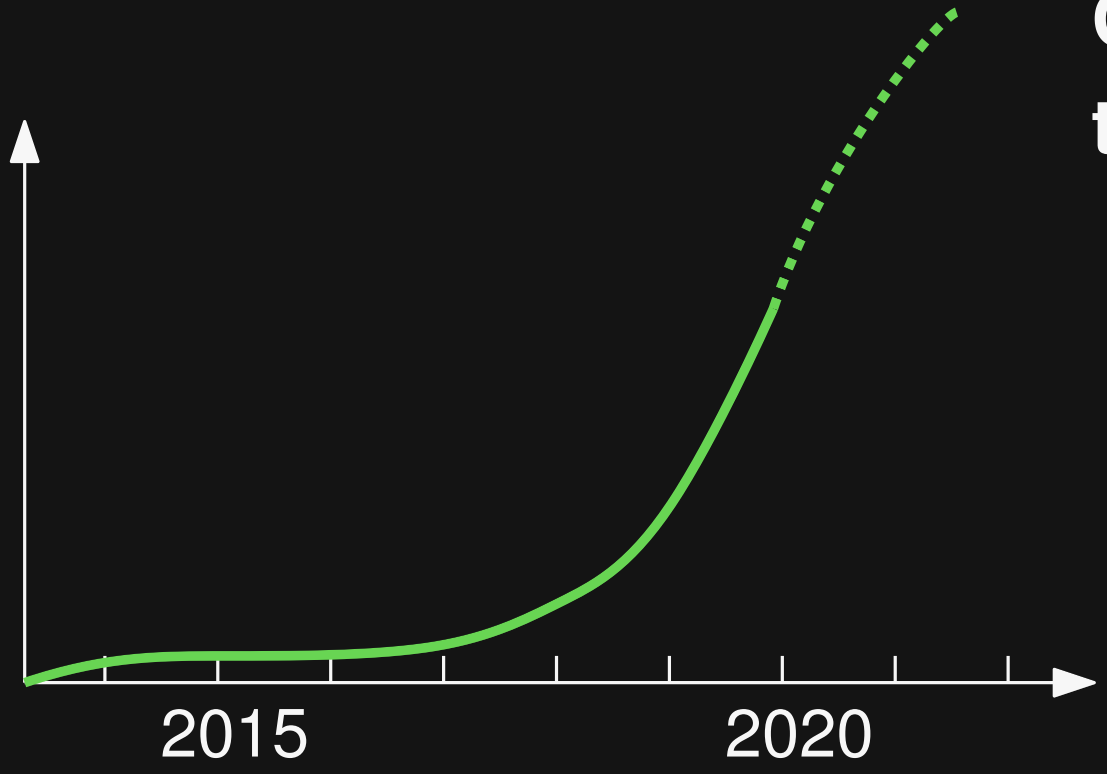


The Road Ahead

General-purpose language models



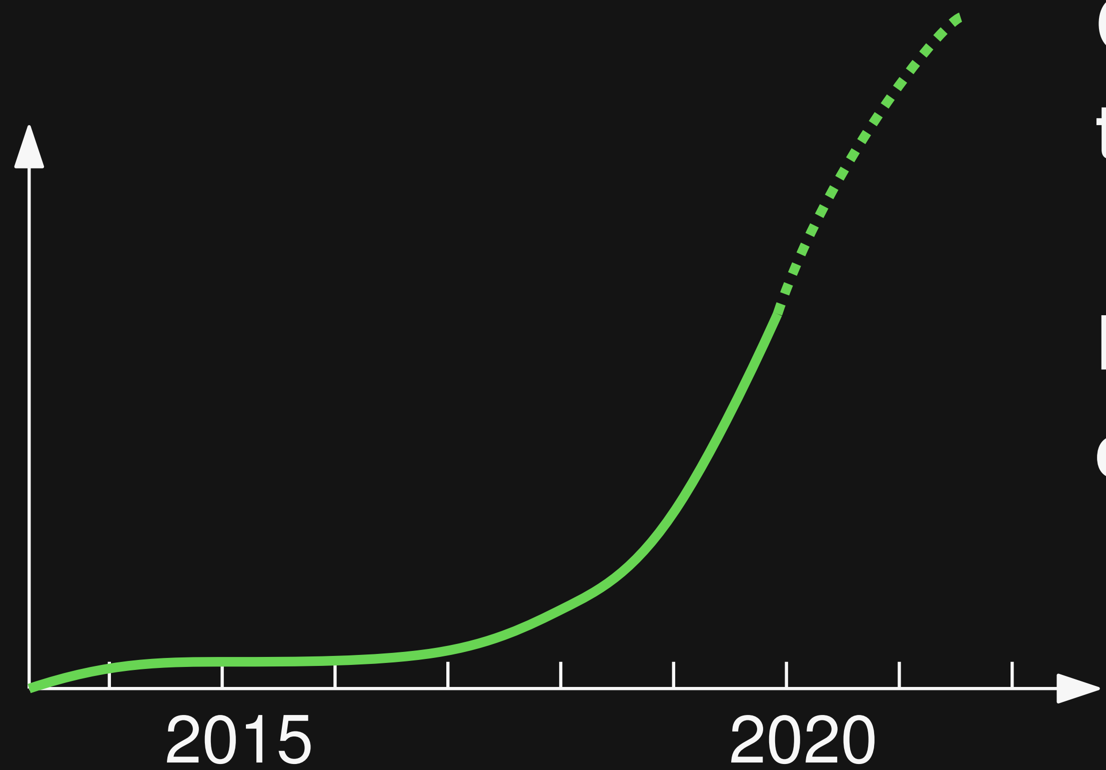
The Road Ahead



General-purpose language models

Combining neural & traditional analysis

The Road Ahead



General-purpose language models

Combining neural & traditional analysis

Reasoning about executions