Machine Learning for Programming (ML4P)

Course page

http://software-lab.org/teaching/winter2022/ml4p/

Prof. Dr. Michael Pradel

Winter 2022/23

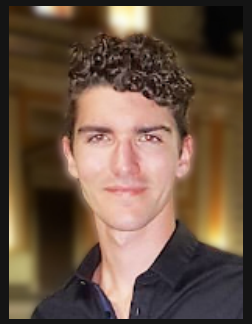
Software Lab, University of Stuttgart

About Me: Michael Pradel

- **Since 9/2019: Full Professor at University of Stuttgart**
- **Before Stuttgart**
  - Studies at TU Dresden, ECP (Paris), and EPFL (Lausanne)
  - PhD at ETH Zurich, Switzerland
  - Postdoctoral researcher at UC Berkeley, USA
  - Assistant Professor at TU Darmstadt
  - Sabbatical at Facebook, Menlo Park, USA

About the Software Lab





- **My research group since 2014**
- **Focus: Tools and techniques for building reliable, efficient, and secure software**
  - Program testing and analysis
  - Machine learning, security
- **Thesis and job opportunities**

Plan for Today

1. Organization

2. Topic of this seminar

Why Have a Seminar?

- **Learn fundamentals of doing research**
  - Read and digest papers
  - Present complex ideas to others
  - Scientific writing
- **Learn about machine learning and program analysis**
  - Exciting and “hot” research area with highly relevant practical applications
  - Maybe your future thesis topic

Organization

- **Today: Kick-off meeting**
- **During the semester**
  - Meetings with mentor
  - Talks by students
- **Your tasks:**
  - Term paper
  - Talk
  - Active participation

Organization

- **Today: Kick-off meeting**
- **During the semester**
  - Meetings with mentor
  - Talks by students
- **Your tasks:** **Grading:**
  - Term paper ← 40%
  - Talk ← 40%
  - Active participation ← 20%

Talk

- **15 minutes + questions**
- **English**
- **Present a recent research paper**

- **Your mentor will help you prepare the presentation**
  - Ask questions about the paper
  - Send slides one week before the talk
  - Incorporate feedback given by the mentor

Talk: Some Advice

Content:

- No need to explain all technical details
- But: Must contain some ”meat”

Presentation:

- Examples are your secret weapon
- Stick to the time limit
- Practice, practice, practice

Pro tip: View video *How to give a good research talk* by Simon Peyton Jones

Talk: Rules

- **Prepare your own slides**
  - No copy & paste from existing slides, even if available
- **You may use examples from the paper**
  - Using your own examples is encouraged

Term Paper

- 6 pages
- English
- LaTeX template on course web site
- Summarize the paper in your own words
- Must be self-containing

Term Paper: Some Advice

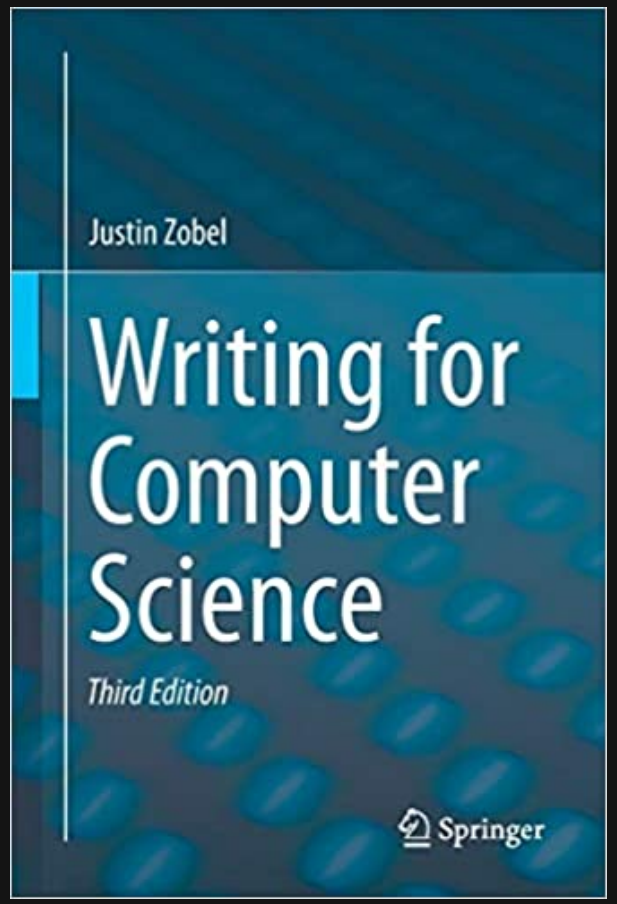
- **Don’t waste space on basics**
- **Examples are your secret weapon (yes, again)**
- **Use a neutral perspective**
  - “the analysis” or “the authors”, not “we”
- **Bad English distracts from good content**
- **Revise, revise, revise**

General Writing Advice

Great book with many useful tips:

“Writing for Computer Science”

by Justin Zobel



Term Paper: Rules

- **No verbatim copying or paraphrasing of existing text**
  - Exception: Clearly marked, short quotes
- **You may copy figures (e.g., result graphs)**
- **You must use exclusively your own example(s)**

Your Choice

	Paper-focused	Talk-focused
Term paper	Early deadline for first draft. Two rounds of feedback	One round of feedback
Talk	Give talk once	Give talk, get feedback, give talk again
Grading	Same for both. Only final versions count	

Your Choice

	Paper-focused	Talk-focused
Term paper	Early deadline for first draft. Two rounds of feedback	One round of feedback
Talk	Give talk once	Give talk, get feedback, give talk again
Grading	Same for both. Only final versions count	

Advice: Choose to focus on the skill you'd like to improve the most

Dates

- **From Nov 10, 2022** (Thu, 2pm-3:30pm): Talks
- **Nov 18, 2022**: First draft of term paper (only paper-focused students)
- **Jan 13, 2023**: Second draft of term paper
- **Feb 10, 2023**: Final term paper

Meetings

- **All meetings are**
  - in the classroom
  - without recording
- **Participation is not mandatory**
  - But: Active participation contributes to the grade
- **First round of talks:**
  Starting on Nov 10, 2022

Registering for the “Exam”

- **As with all other courses: Students must register for the exam**
  - Prerequisite for obtaining a grade
- **“Exam” here means participating in the course**
  - No written exam at end of semester

Topics To Choose From

- **19 recently published research papers:**
  http://software-lab.org/teaching/winter2022/ml4p/
- **Submit your preferences until next Monday (Oct 24, end of day)**
  - You pick three topics, we assign one
  - Choose between paper-focused and talk-focused
  - Indicate your preferences in a mail to katharina.plett@iste.uni-stuttgart.de

Plan for Today

1. Organization ✓

2. Topic of this seminar

Topic of This Seminar

Machine Learning for Programming

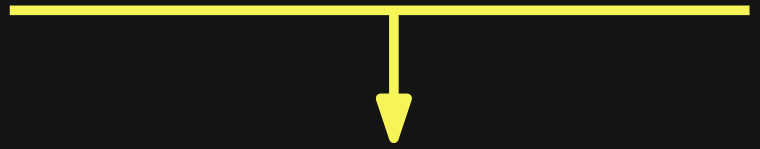
Topic of This Seminar

Machine Learning for Programming

- Tools for improving software reliability and security
- E.g., program analyses to detect bugs, to complete partial code, or to de-obfuscate code

Topic of This Seminar

Machine Learning for Programming



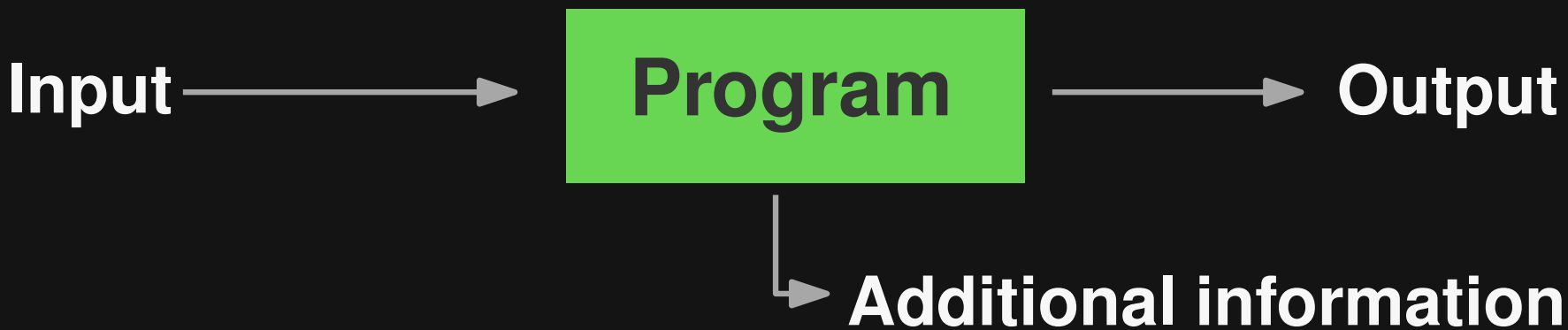
- Source code as data
- Large code corpora to learn from
- Train models that predict program properties

What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities

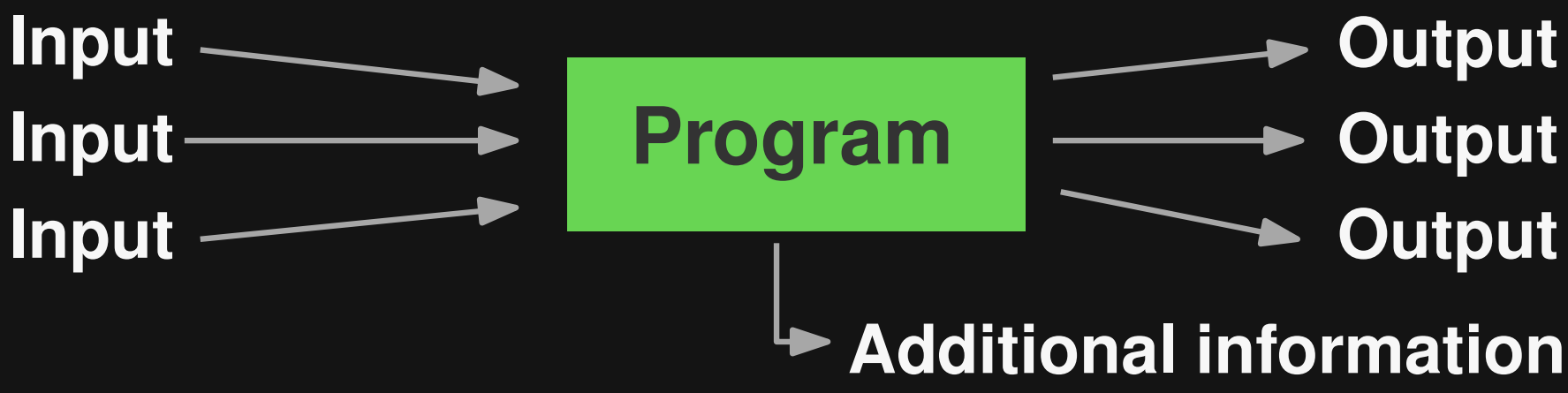
What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities



What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities



Why Do We Need It?

Basis for various tools that make developers productive

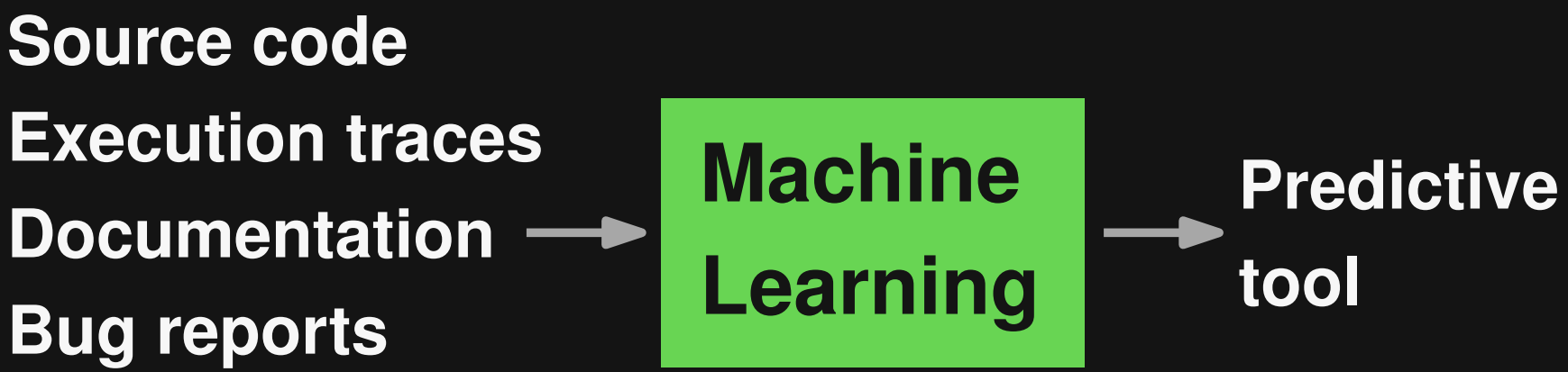
- Compilers
- Bug finding tools
- Performance profilers
- Code completion
- Automated testing
- Code summarization/documentation

Traditional Approaches

- **Analysis has built-in knowledge about the problem to solve**
- **Significant human effort to create a program analysis**
  - Conceptual challenges
  - Implementation effort
- **Analyze a single program at a time**

Neural Software Analysis

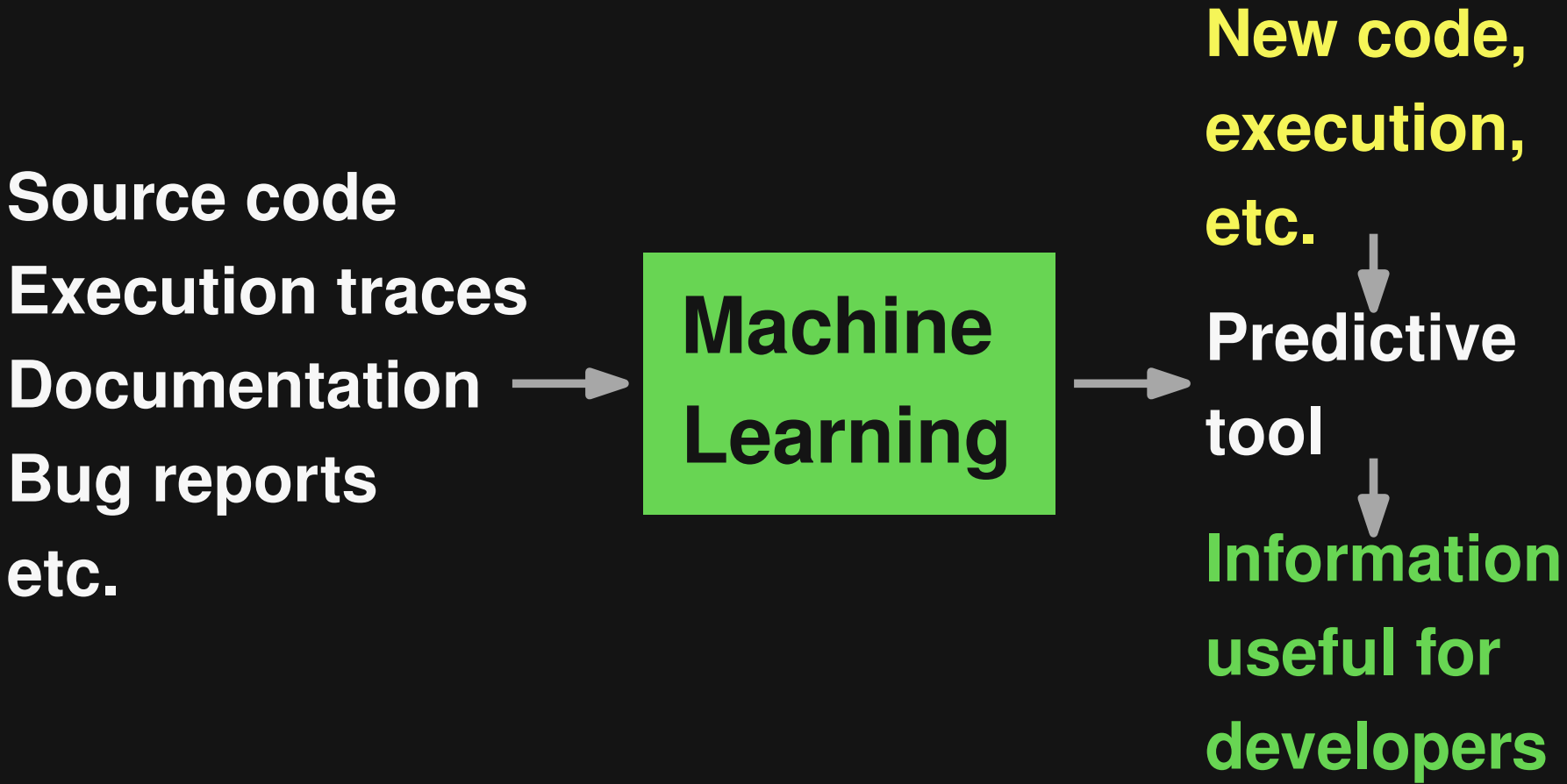
Insight: Lots of data about software development to learn from



Neural Software Analysis, Pradel & Chandra, CACM'22

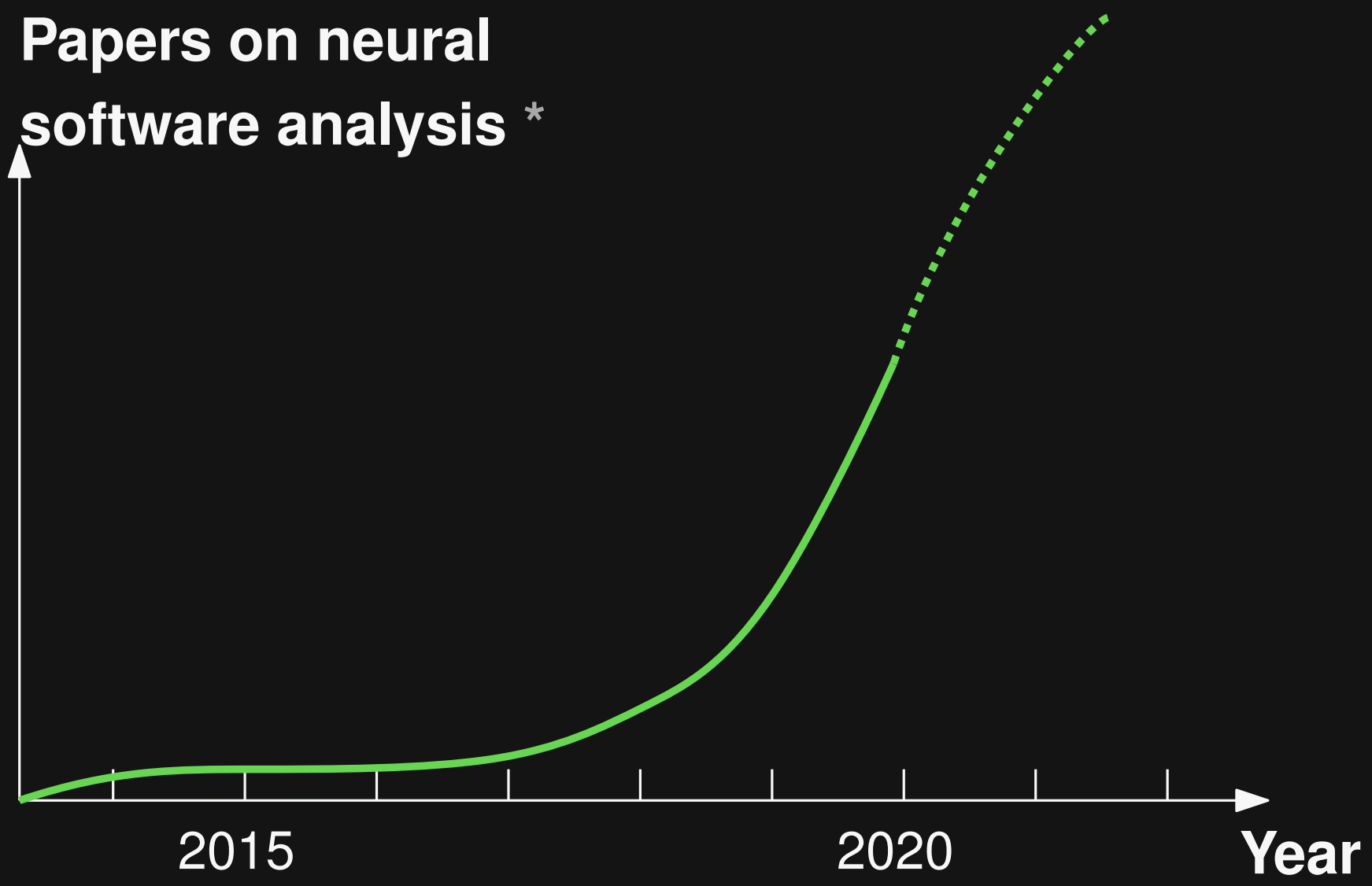
Neural Software Analysis

Insight: Lots of data about software development to learn from



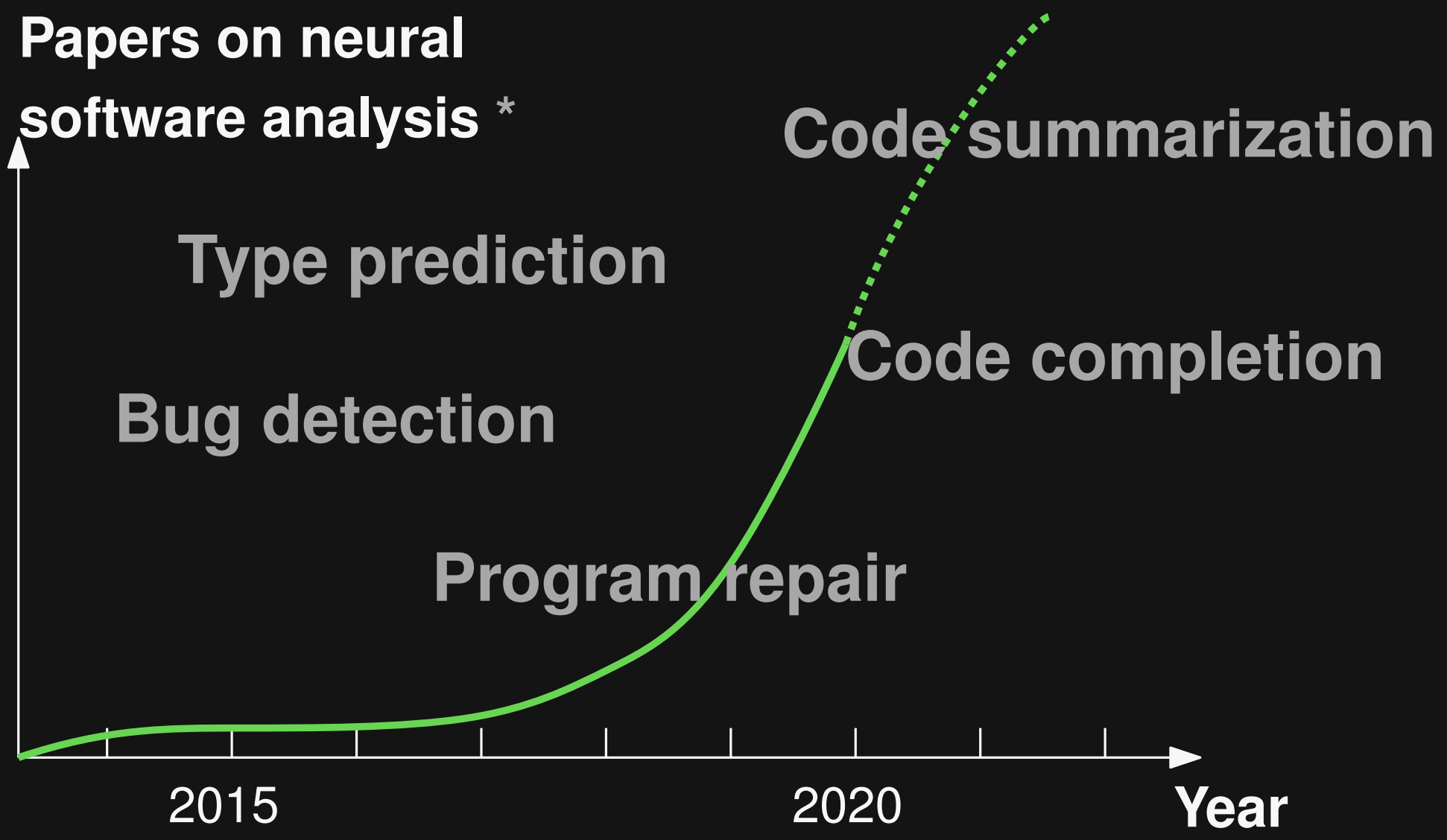
Neural Software Analysis, Pradel & Chandra, CACM'22

Join the Hype!



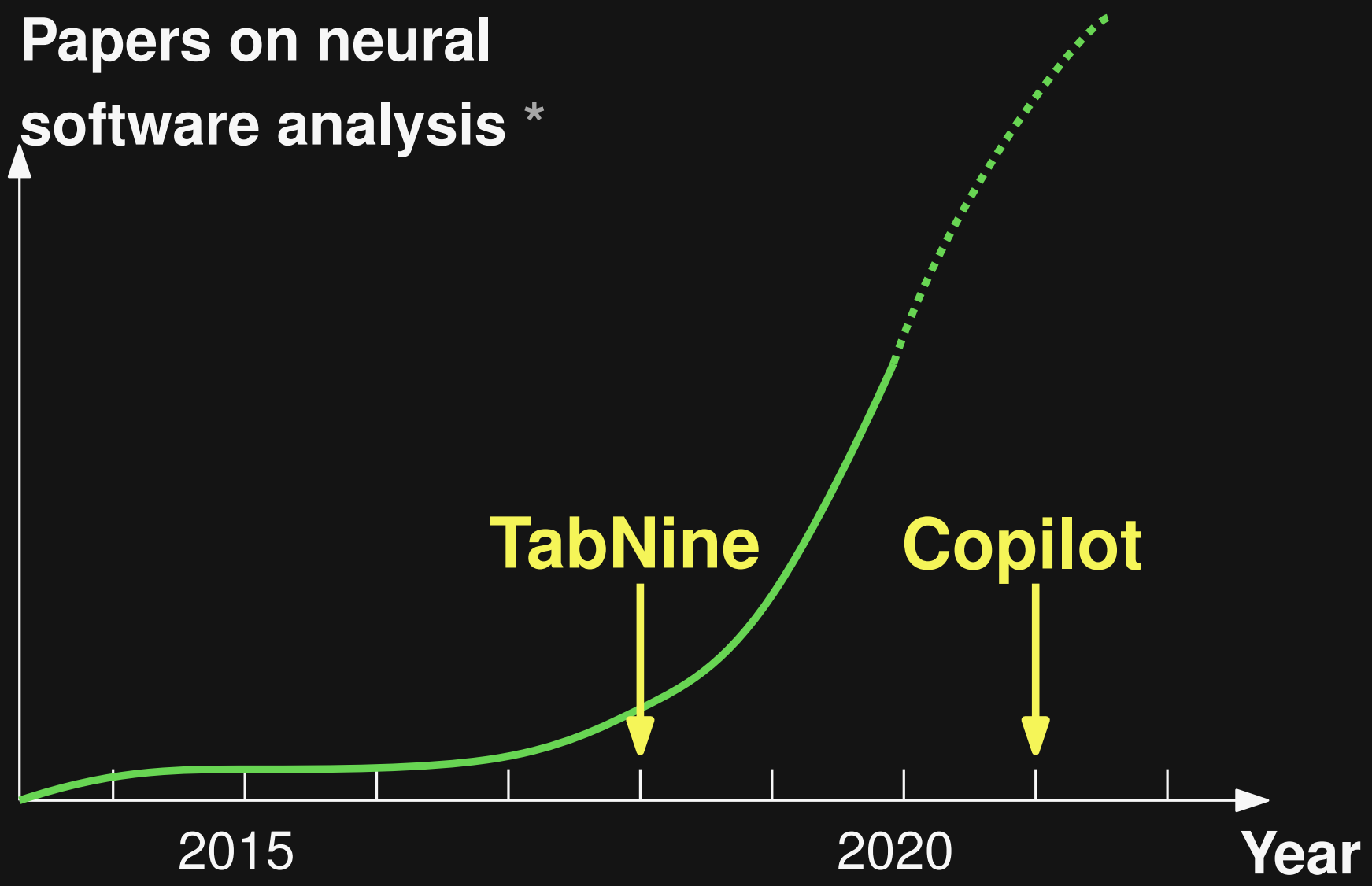
* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

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

Neural Software Analysis

The Good, the Bad, and the Ugly

Neural Software Analysis

The Good, the Bad, and the Ugly

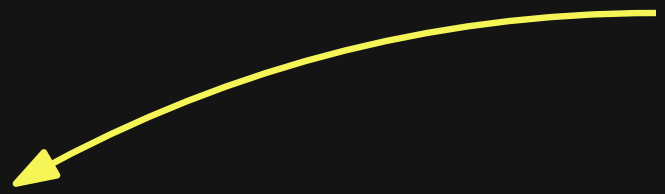
Bug detection with Nalin

Type prediction with TypeWriter

Motivation

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

Motivation

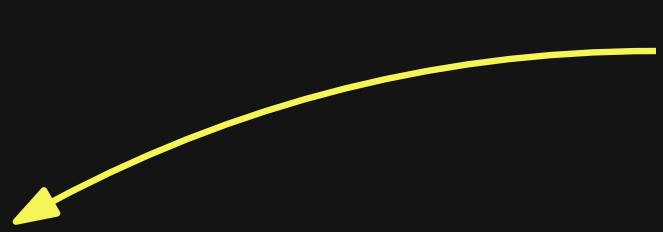


Incorrect value:

`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

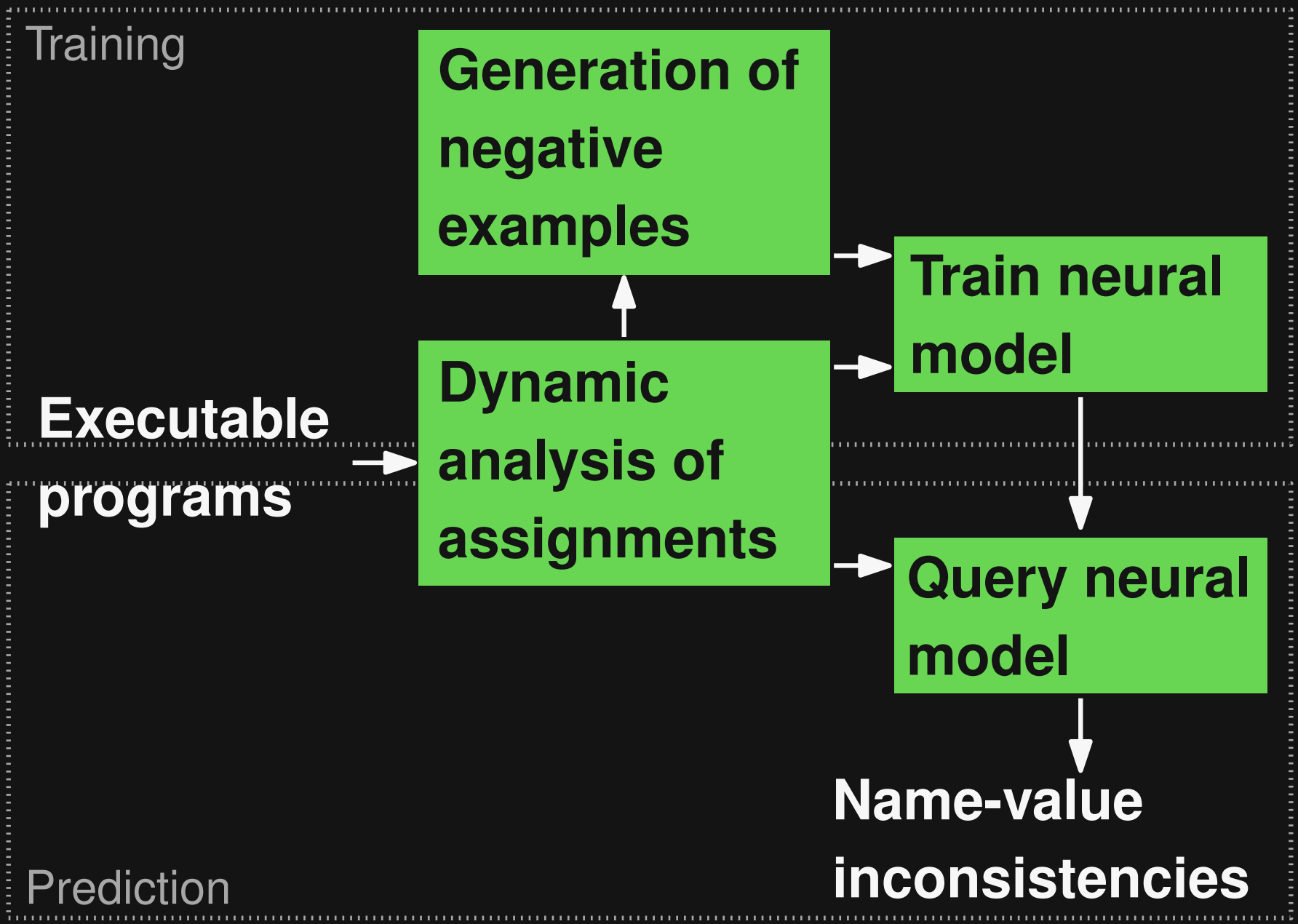
Challenge 1:
Understand the
meaning of names

Challenge 2:
Understand the
meaning of values

Finding name-value inconsistencies

Challenge 3:
Precisely pinpoint
unusual pairs

Overview of Nalin



Analyzing Assignments

Dynamic analysis

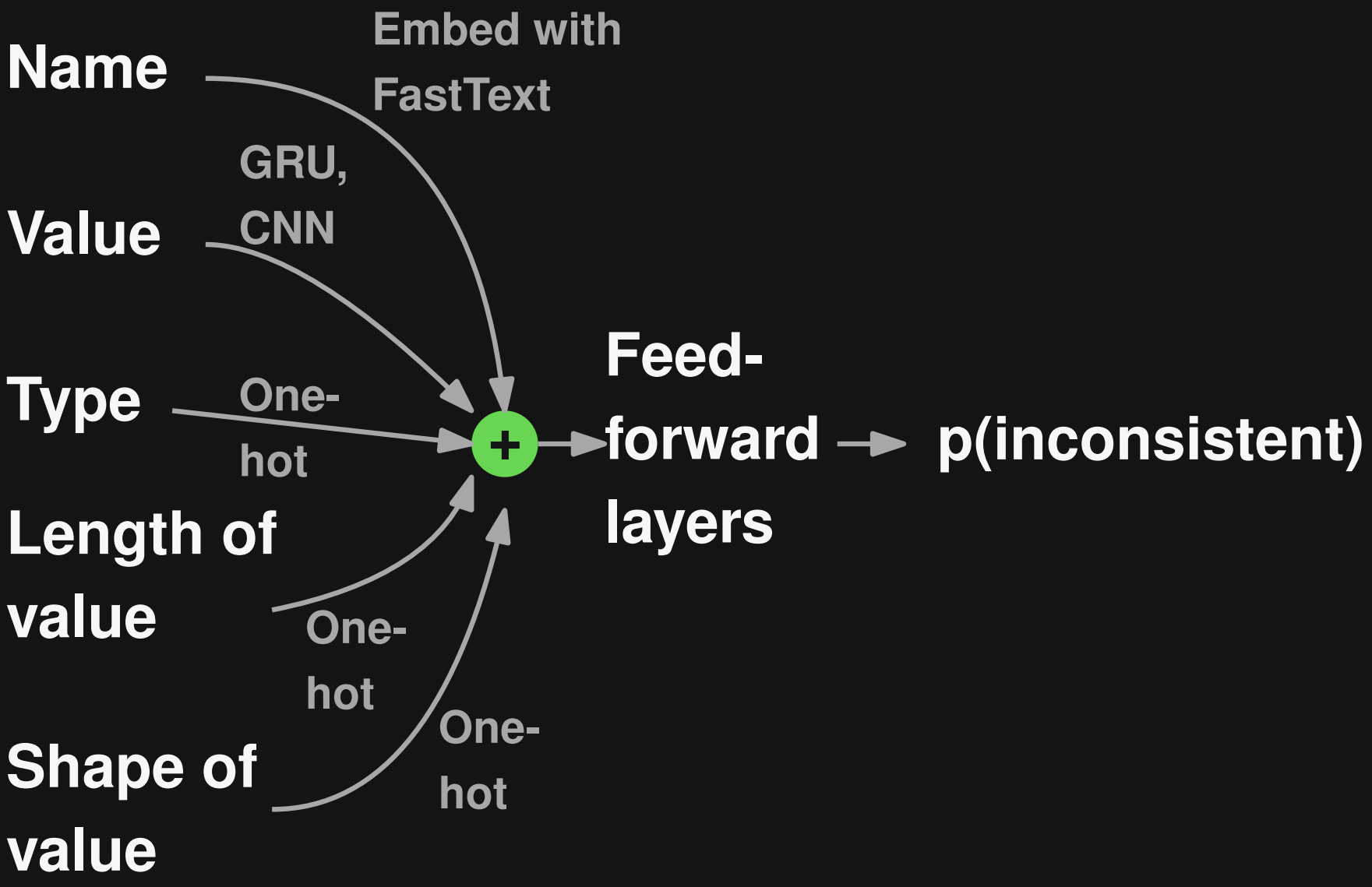
- Extract for each assignment
  - Name of left-hand side
  - String representation of value
  - Type of value
  - Length of value
  - Shape of value

Analyzing Assignments

Example:

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

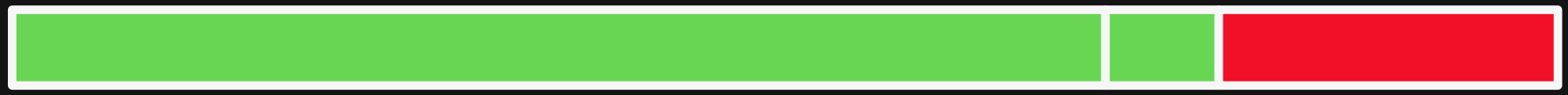
Evaluation

- **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



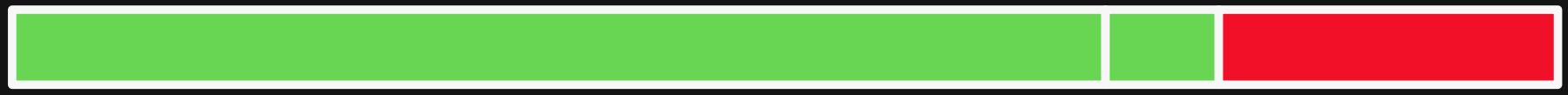
21 misleading names

2 incorrect values

7 false positives

Kinds of Inconsistencies

30 inspected warnings



21 misleading names

2 incorrect values

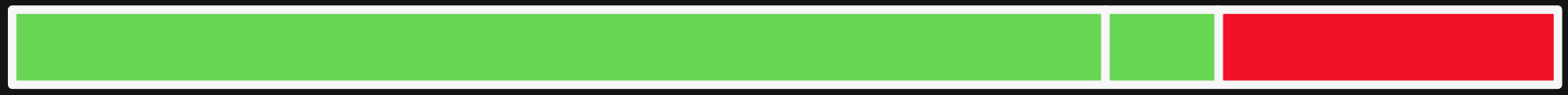
7 false positives

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

Unusual combination

Kinds of Inconsistencies

30 inspected warnings



21 misleading names

2 incorrect values

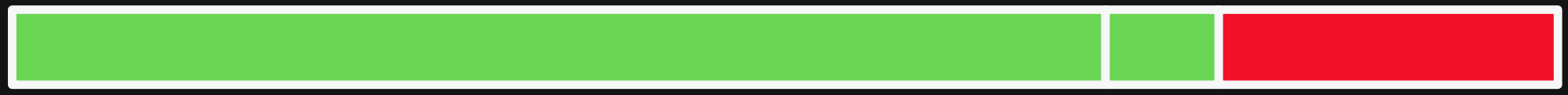
7 false positives

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

Value: `"Corporate"`

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?

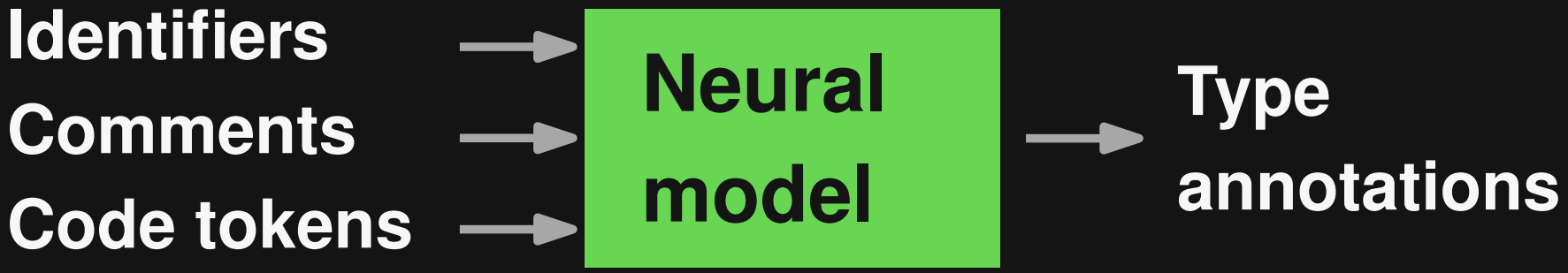
Yes, but: Lots of code without type annotations

How to Add Type Annotations?

- **Option 1: Static type inference**
  - Guarantees type correctness, but very limited
- **Option 2: Dynamic type inference**
  - Depends on inputs and misses types
- **Option 3: Probabilistic type prediction**
  - Models learned from existing type annotations

Probabilistic Type Prediction

Neural model to predict types



Prior models, e.g.:

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

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

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"]
```

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 (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"]
```

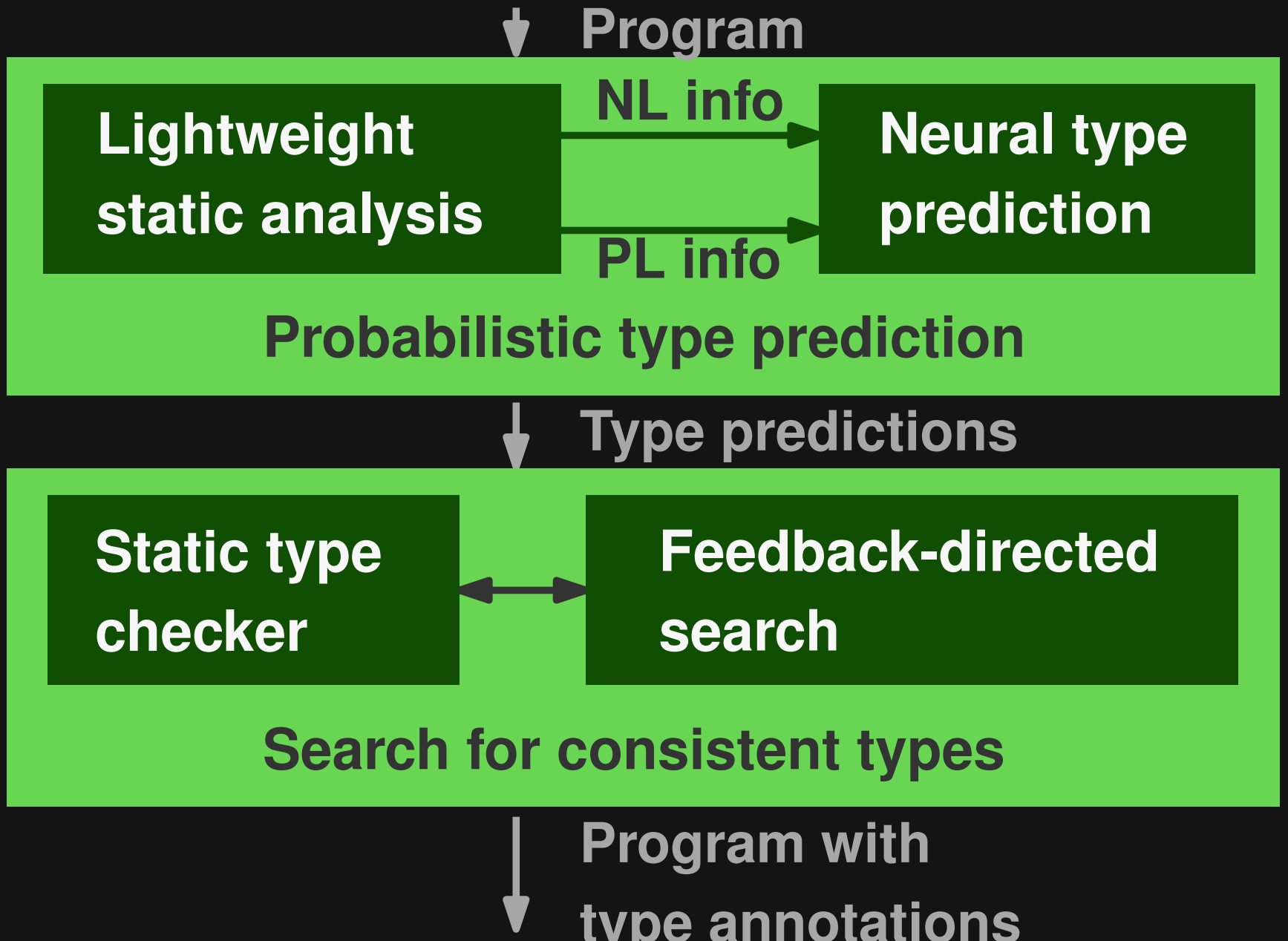
Correct predictions

Predictions:
1) int
2) str
3) bool

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

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

Overview of TypeWriter



Extracting NL Information

```
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

```
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"]
```

Function-level comments

Extracting PL Information

```
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"]
```

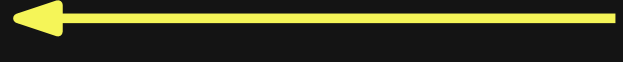
Extracting PL Information

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"]
```

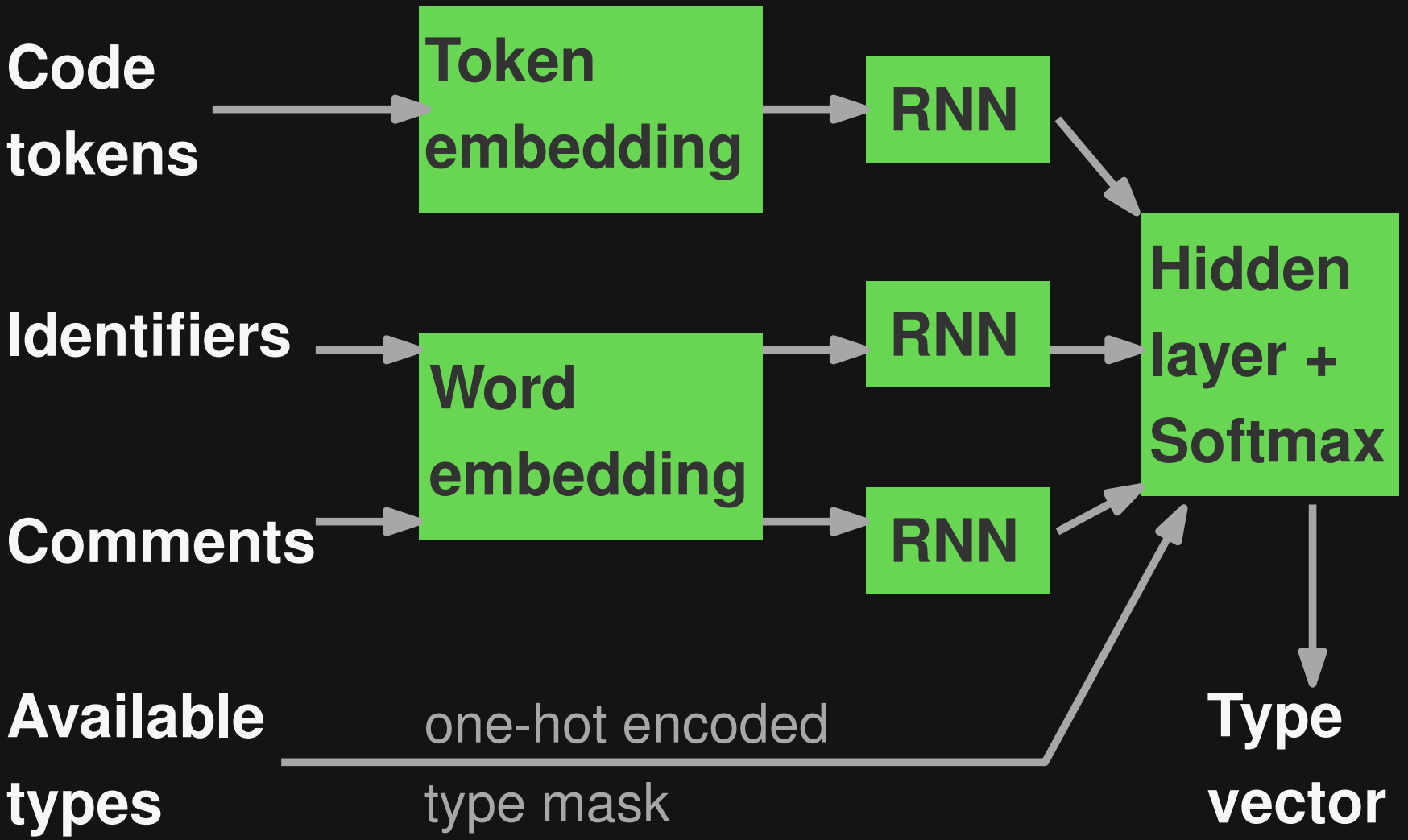
Extracting PL Information

```
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



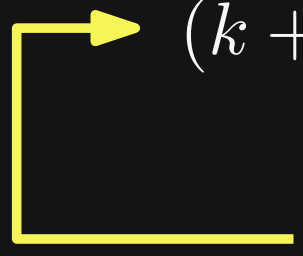
TypeWriter: Neural Type Prediction with Search-based Validation, FSE'20

Searching for Consistent Types

- **Top-k predictions for each missing type**
  - Filter predictions using gradual type checker
  - E.g., 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!

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

```
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 (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

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

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

Evaluation

- **Experimental setup**
  - Facebook's Python code
  - 5.8 millions lines of open-source code
- **Results**
  - Neural model: 80% F1-score (top-5)
  - Search: Correctly annotates 75% of all missing types 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 = ♡

Neural Software Analysis

The Good, the Bad, and the Ugly

Neural Software Analysis

The Good, the Bad, and the Ugly

Let's address all program analysis problems through neural software analysis!

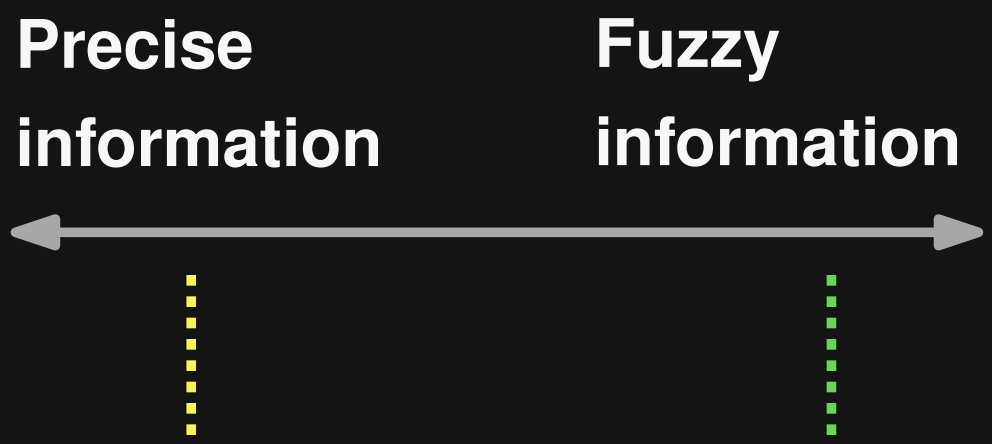
~~Let's address all program analysis problems through neural software analysis!~~

When to (not) use neural software analysis?

- Fuzziness of available information
- Well-defined correctness criterion
- Data to learn from

Fuzziness of Available Information

Fuzziness of Available Information

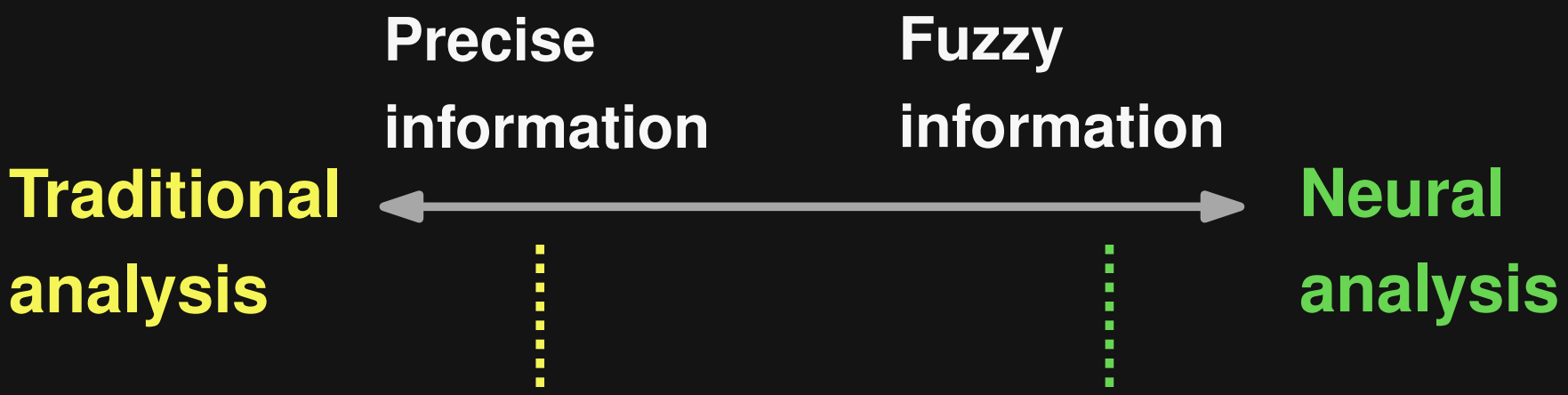


If code has property A, then B holds.

If code is similar to pattern A, then B is likely to hold.

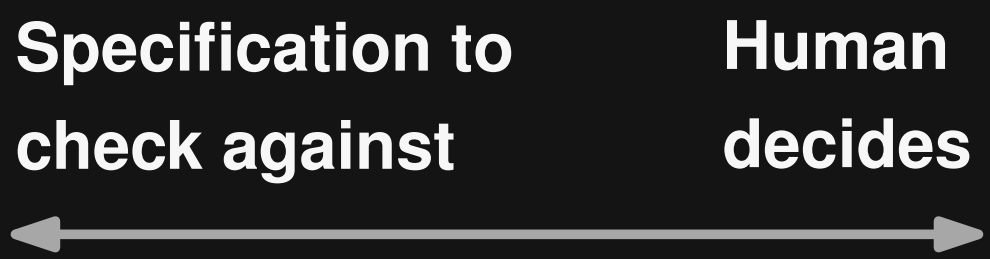
Neural Software Analysis, Pradel & Chandra, CACM'22

Fuzziness of Available Information



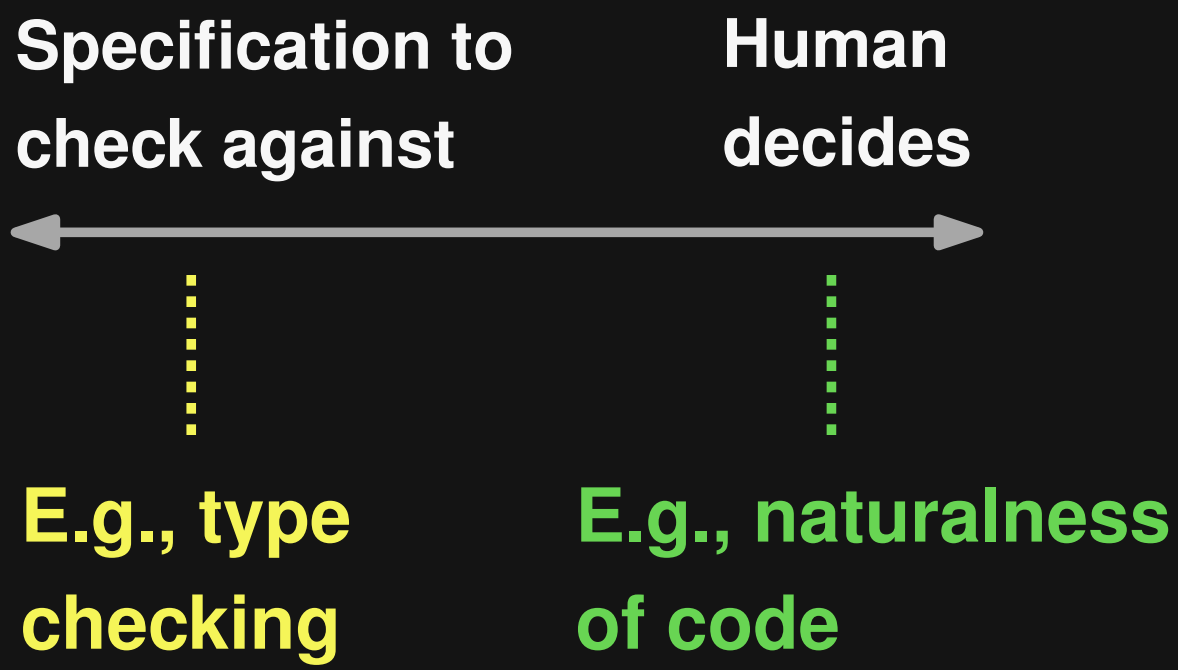
Neural Software Analysis, Pradel & Chandra, CACM'22

Well-defined Correctness Criterion



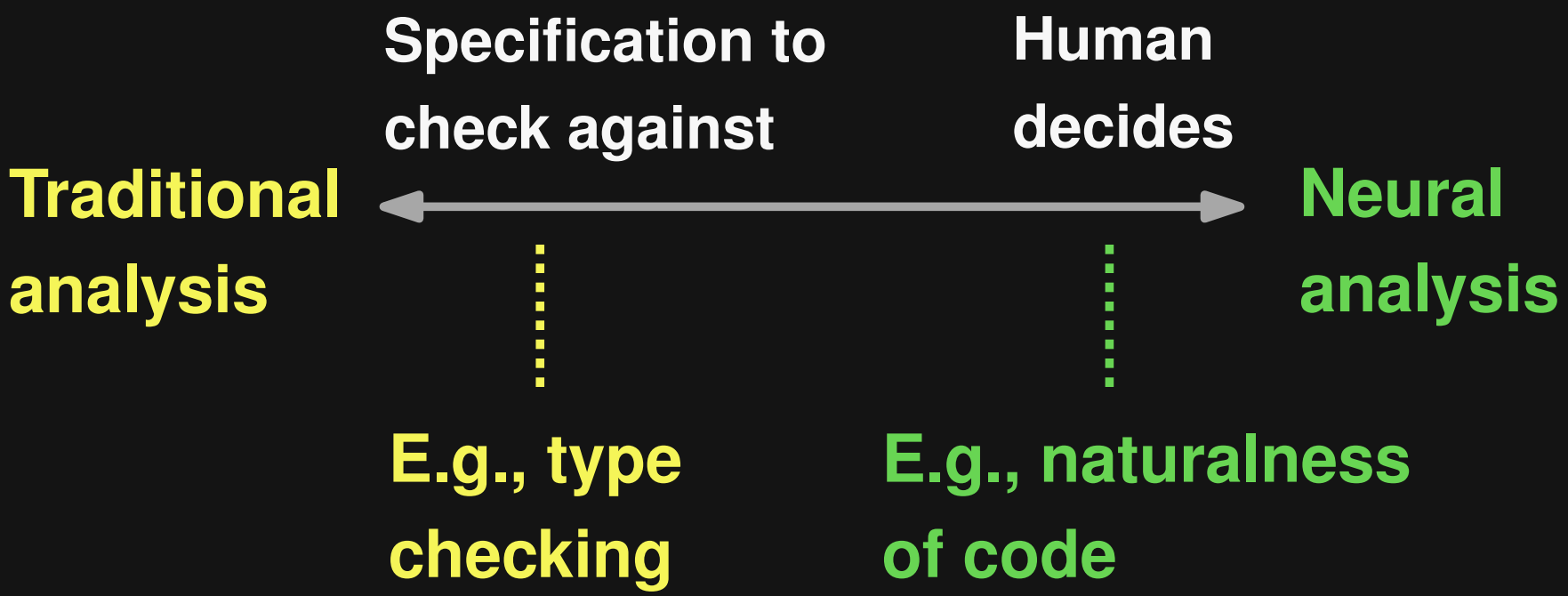
Neural Software Analysis, Pradel & Chandra, CACM'22

Well-defined Correctness Criterion



Neural Software Analysis, Pradel & Chandra, CACM'22

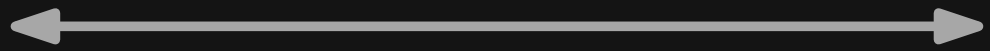
Well-defined Correctness Criterion



Neural Software Analysis, Pradel & Chandra, CACM'22

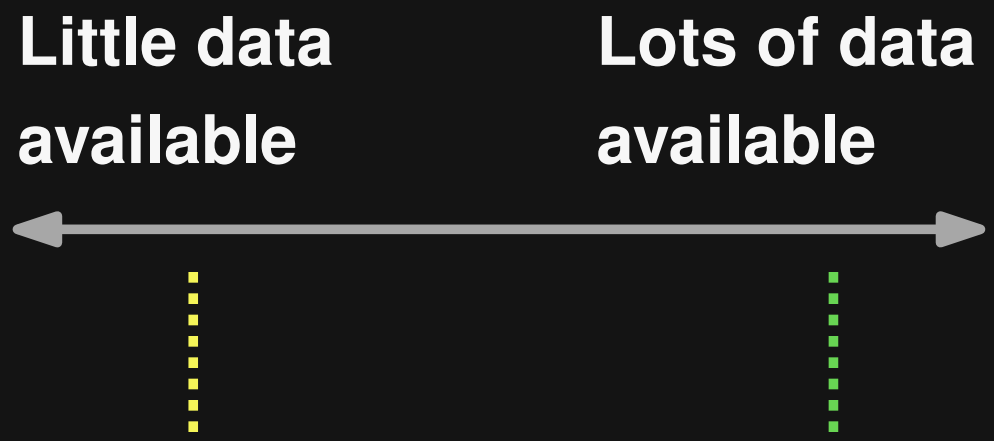
Data to Learn From

Little data available ⟷ **Lots of data available**



Neural Software Analysis, Pradel & Chandra, CACM'22

Data to Learn From



E.g., anything requiring human interaction

E.g., code completion

Neural Software Analysis, Pradel & Chandra, CACM'22

Data to Learn From

Neural Software Analysis, Pradel & Chandra, CACM'22

Data to Learn From

Want: Realistic, low-noise dataset

Neural Software Analysis, Pradel & Chandra, CACM'22

Data to Learn From

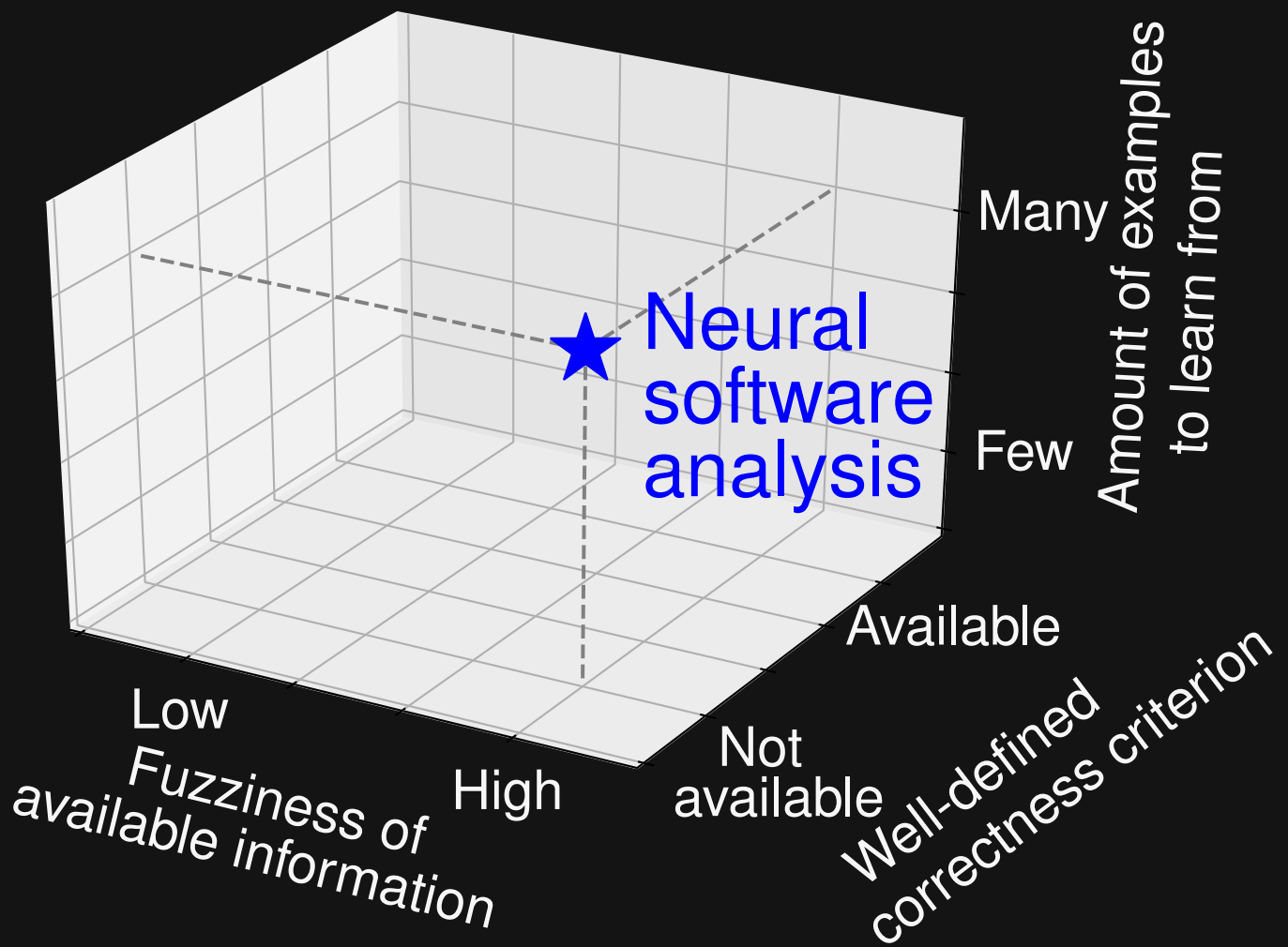
Want: Realistic, low-noise dataset

... for a task that developers care about

Neural Software Analysis, Pradel & Chandra, CACM'22

Neural Software Analysis

When to (not) use it?



Neural Software Analysis, Pradel & Chandra, CACM'22

Neural Software Analysis

The Good, the Bad, and the Ugly

Neural Software Analysis

The Good, the Bad, and the Ugly

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

Task: Code Summarization

```
{
  if (!prepared(state)) {
    return state.setStatus(MovementStatus.PREPPING);
  } else if (state.getStatus() == MovementStatus.PREPPING) {
    state.setStatus(MovementStatus.WAITING);
  }
  if (state.getStatus() == MovementStatus.WAITING) {
    state.setStatus(MovementStatus.RUNNING);
  }
  return state;
}
```

Input: Method body → **Output: Method name**

`updateState`

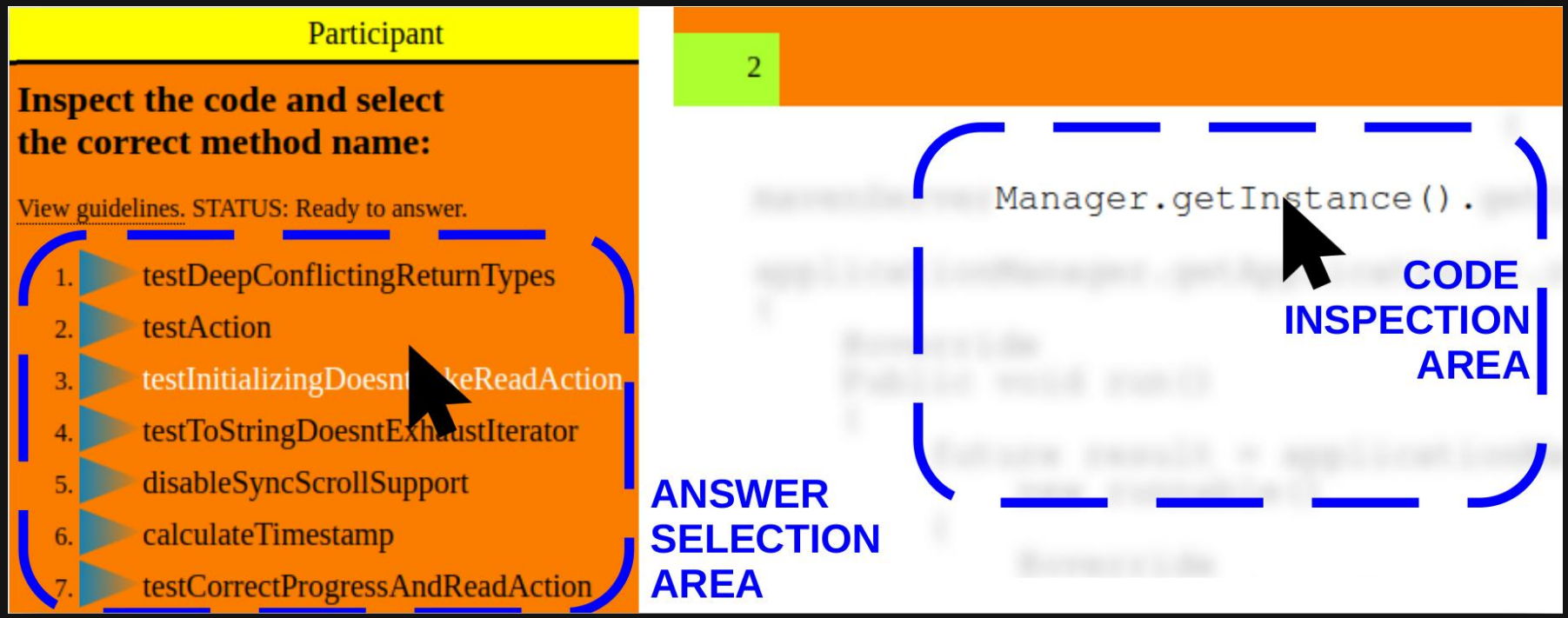
Dataset: 250 methods from 10 Java projects *

* *A Convolutional Attention Network for Extreme Summarization of Source Code*, ICML'16

Capturing Human Attention

- **Goal: Track human attention while performing the task**
- **Approach: Unbluring-based web interface**
  - Initially, all code blurred
  - Moving mouse/cursor temporarily unblurs tokens

Capturing Human Attention



Capturing Human Attention

- **91 participants**: Undergrads, graduate students, crowd workers
- **1,508 human attention records**
- 5+ records for each of 250 methods
- On average per record:
  1,271 mouse-token events

Model Attention

Two code summarization models

- Convolutional sequence-to-sequence (CNN)
  A Convolutional Attention Network for Extreme Summarization of Source Code, ICML'16
- Transformer-based, sequence-to-sequence model
  A Transformer-based Approach for Source Code Summarization, ACL'20
- Both models:

  Regular attention and copy attention

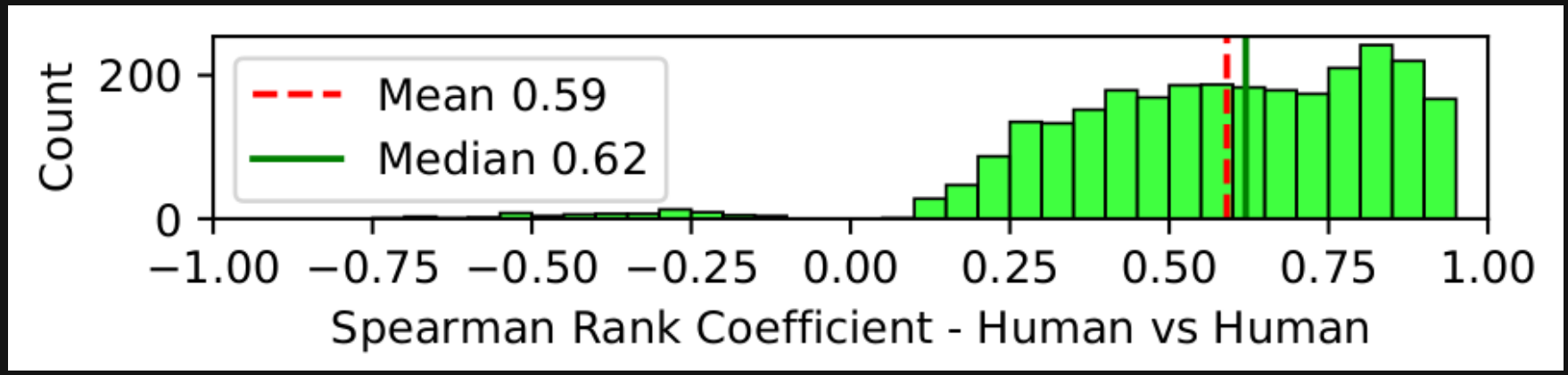
Human-Model Agreement

Do developers and models focus on the same tokens?

- Measure agreement between attention vector via Spearman rank correlation

Results: Human-Model Agreement

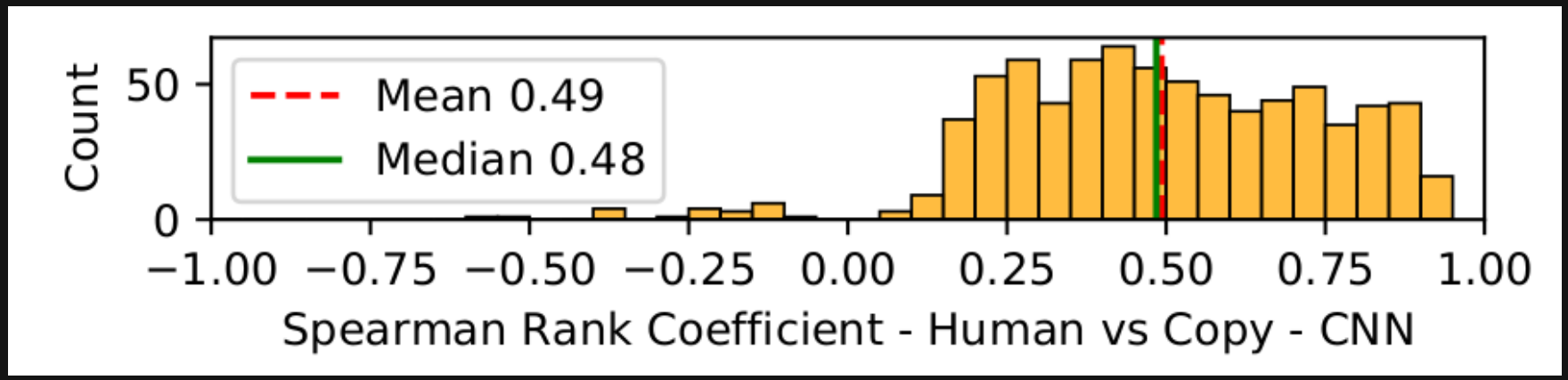
Human-human agreement:

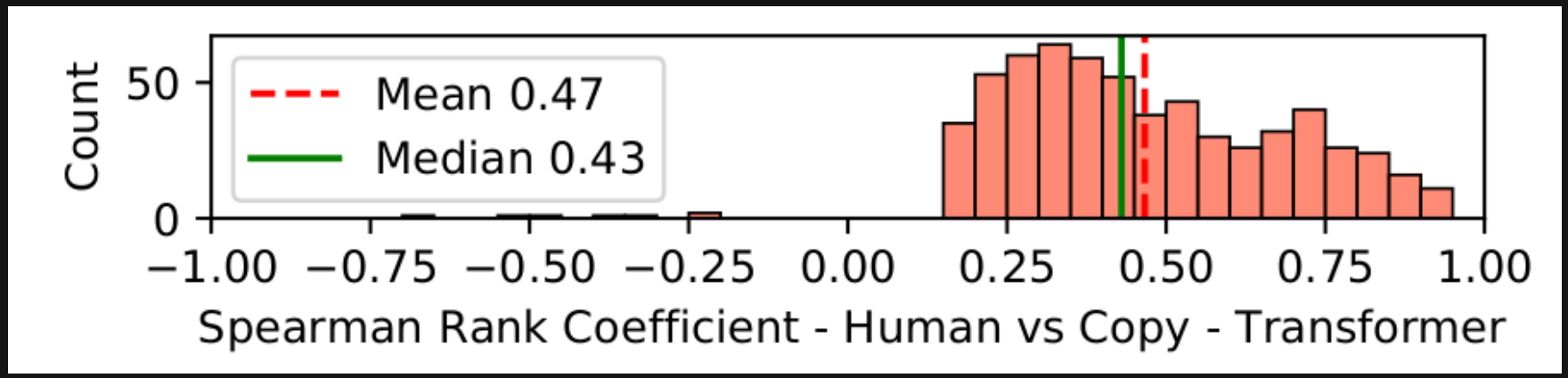


Developers mostly agree on what code matters most

Results: Human-Model Agreement

Human vs. copy attention:

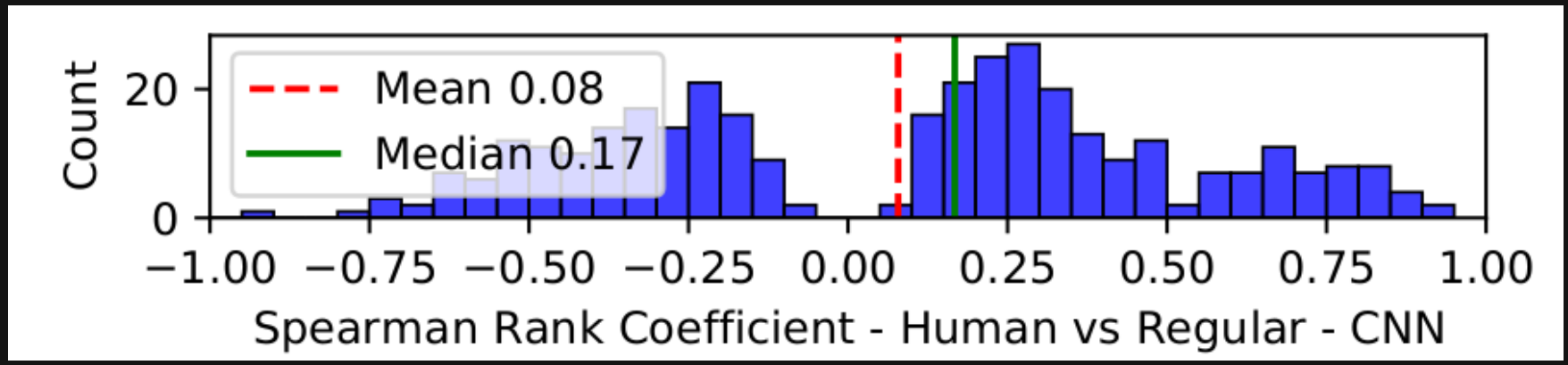


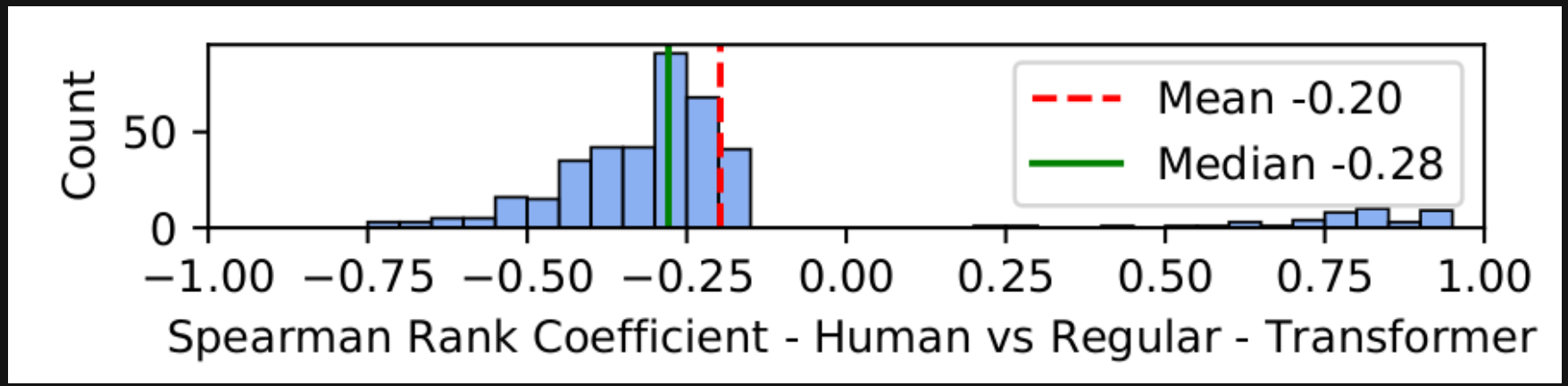


Empirical justification for copy attention

Results: Human-Model Agreement

Humans vs. regular attention:





Lots of room for improvement!

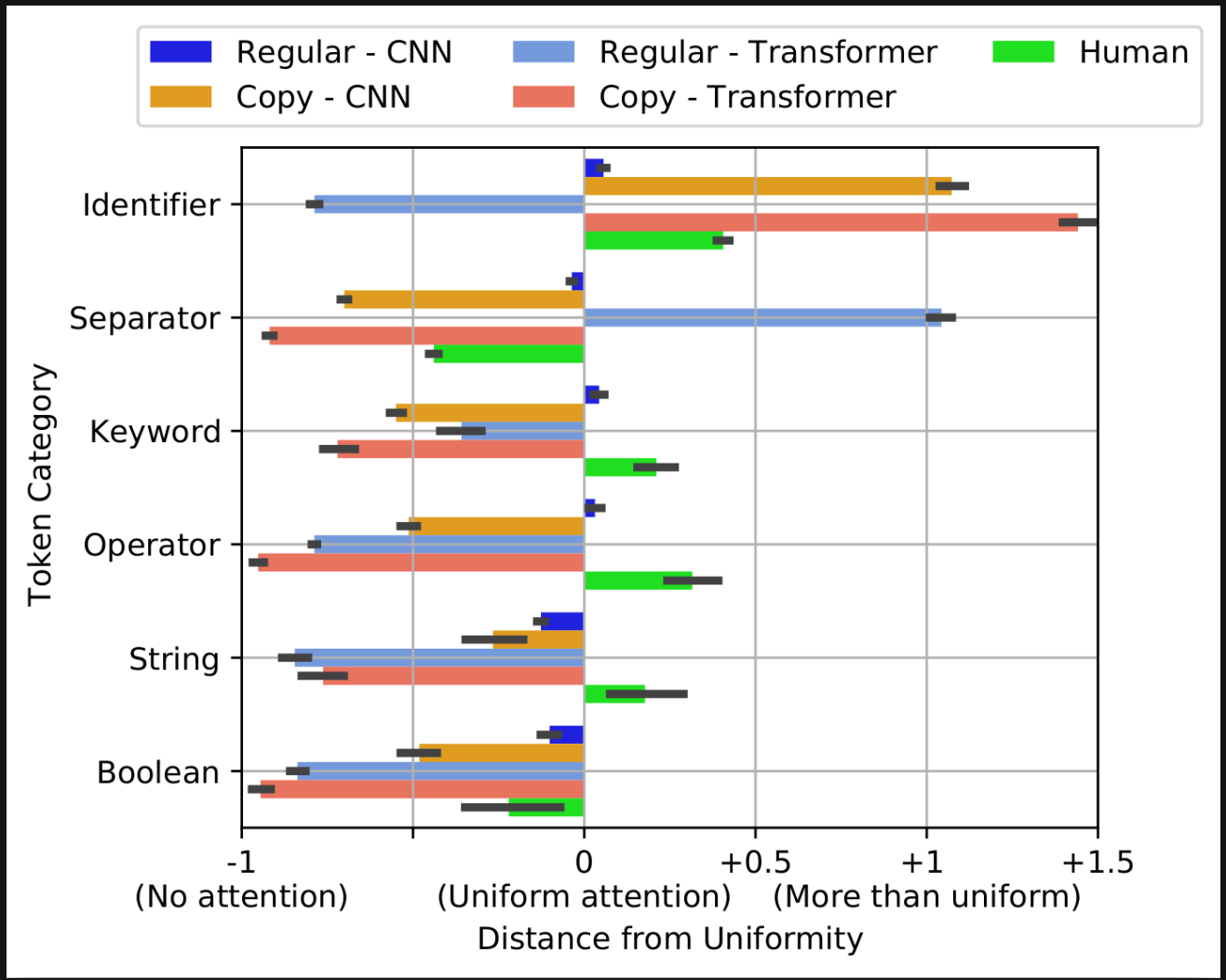
Tokens to Focus On

What kind of tokens to focus on?

- Different kinds: Identifiers, separators, etc.
- For each kind, compute distance from uniformity
  - $= 0$ means uniform attention
  - -1 means no attention at all
  - > 0 means more than uniform attention

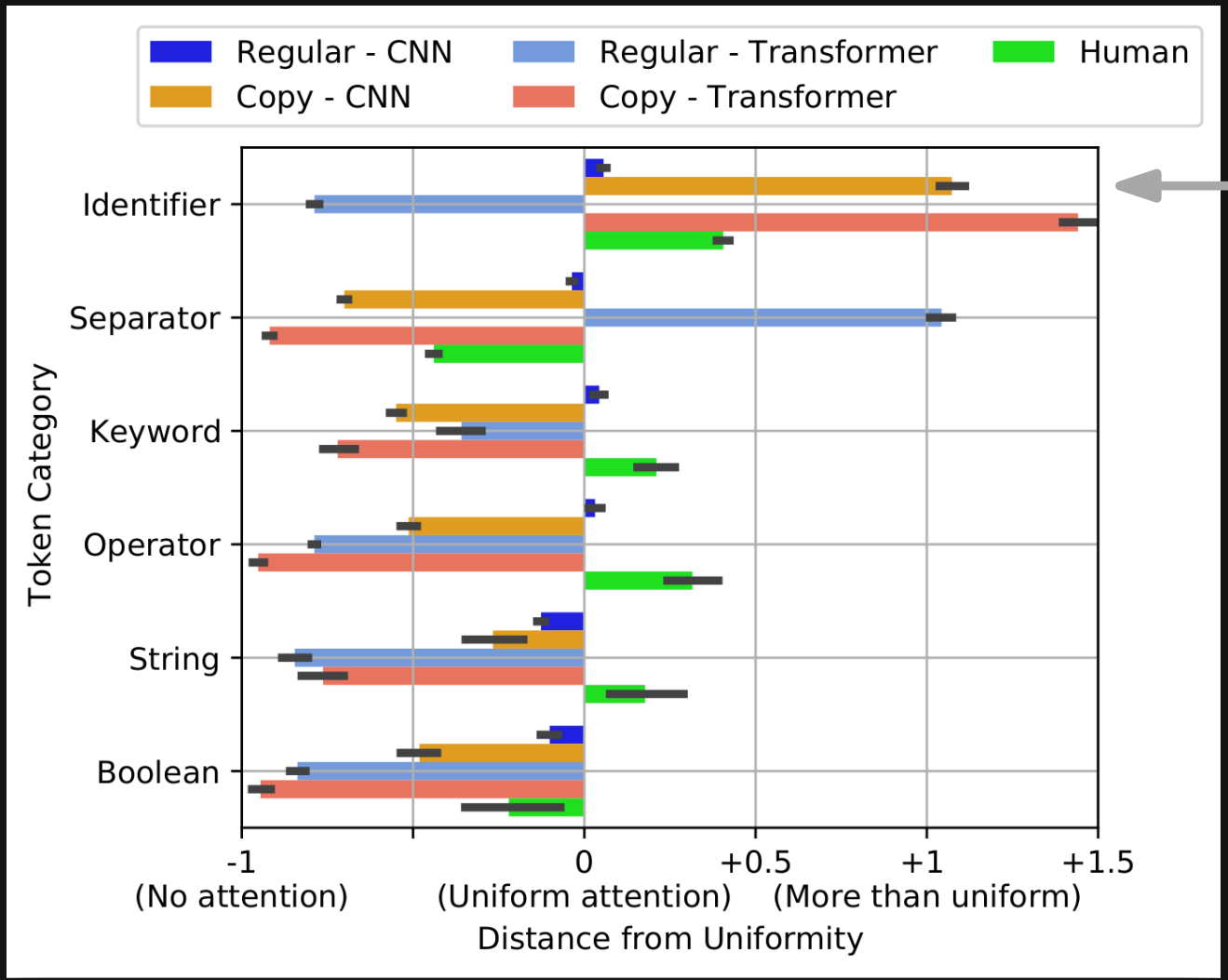
Results: Tokens to Focus On

Distance from uniformity:



Results: Tokens to Focus On

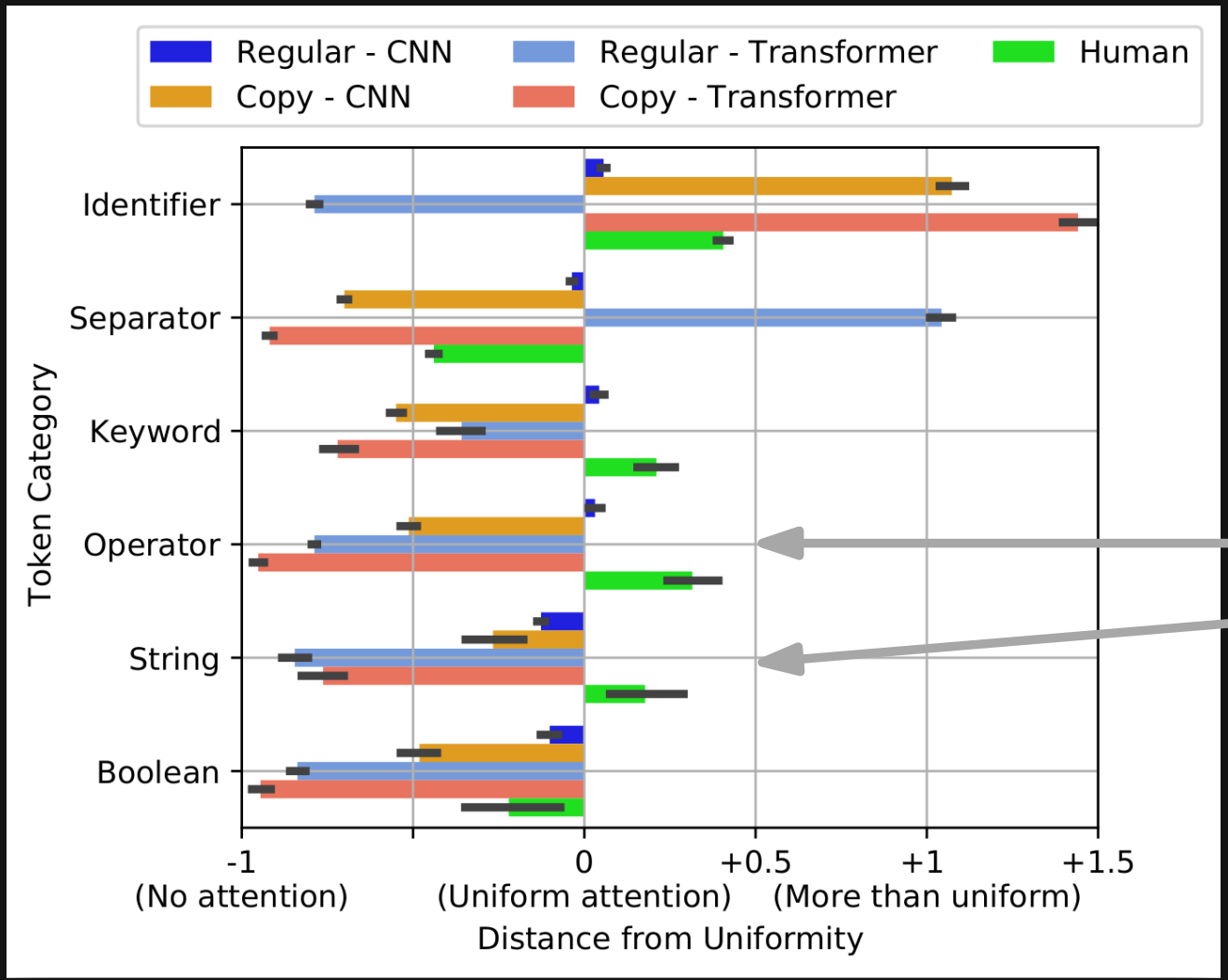
Distance from uniformity:



Identifiers are deemed important

Results: Tokens to Focus On

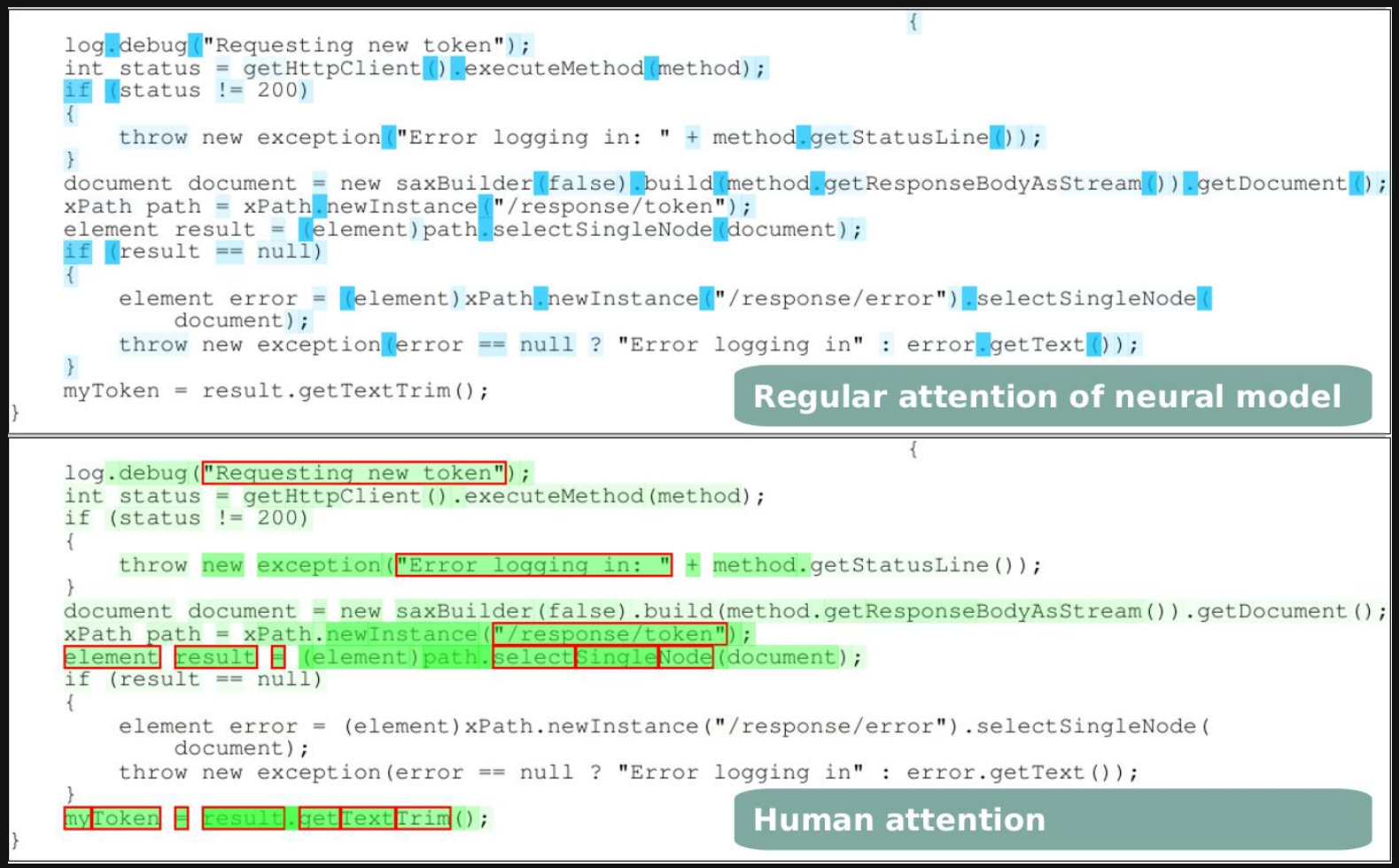
Distance from uniformity:



Models mostly ignore some kinds of tokens

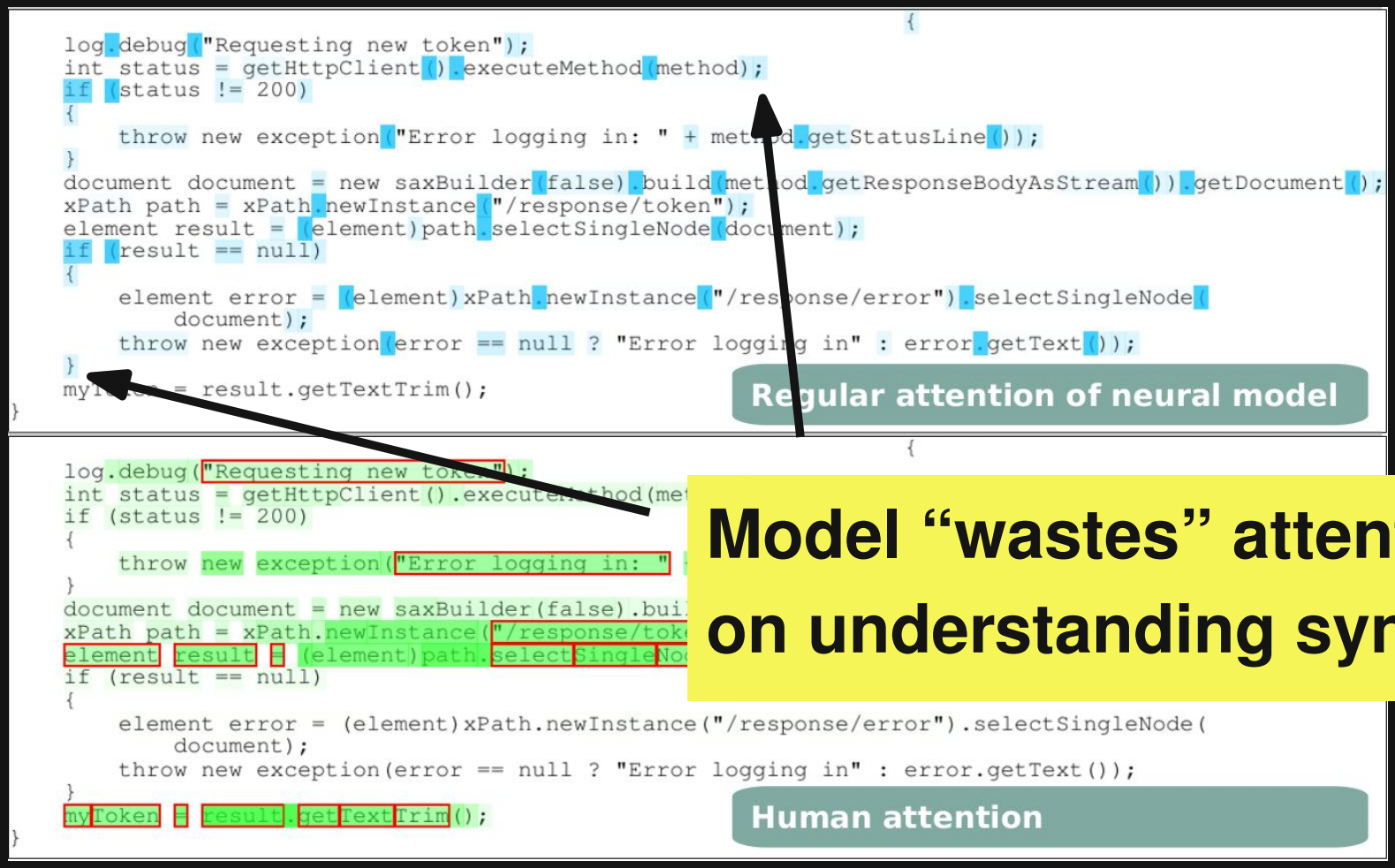
Results: Tokens to Focus On

Example from Transformer model:



Results: Tokens to Focus On

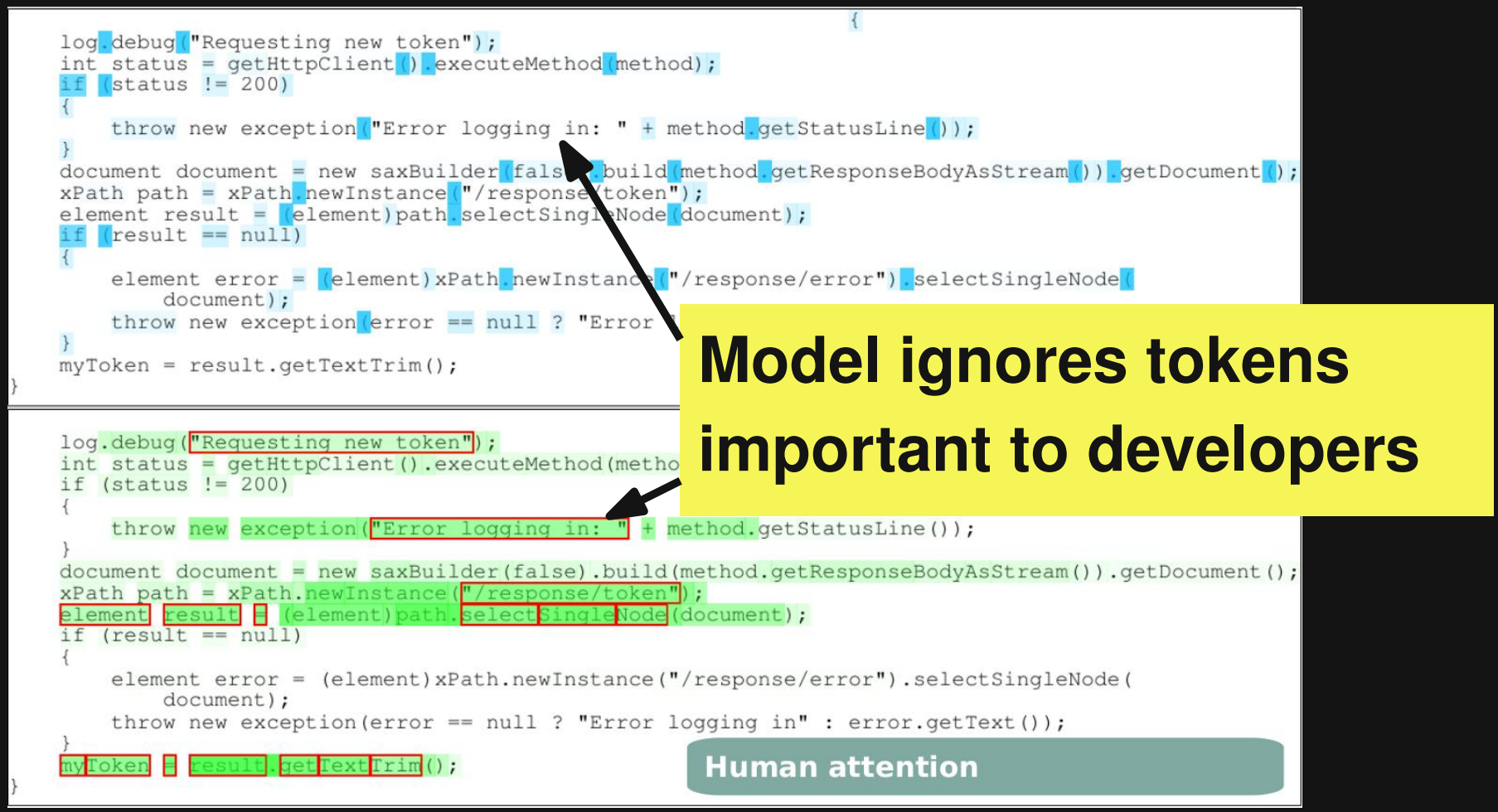
Example from Transformer model:



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

Results: Tokens to Focus On

Example from Transformer model:



Effectiveness vs. Agreement

Are models more effective when they agree more with developers?

Results: Summarization

Human-model agreement for all vs. accurate predictions:

	Spearman rank correl.	
	All methods	Methods with F1 $\geq$ 0.5
CNN (regular)	0.08	0.24
CNN (copy)	0.49	0.55
Transformer (reg.)	-0.20	0.02
Transformer (copy)	0.47	0.55

Results: Summarization

Human-model agreement for all vs. accurate predictions:

	Spearman rank correl.	
	All methods	Methods with F1 $\geq$ 0.5
CNN (regular)	0.08	0.24
CNN (copy)	0.49	0.55
Transformer (reg.)	-0.20	0.02
Transformer (copy)	0.47	0.55

Results: Summarization

Human-model agreement for all vs. accurate predictions:

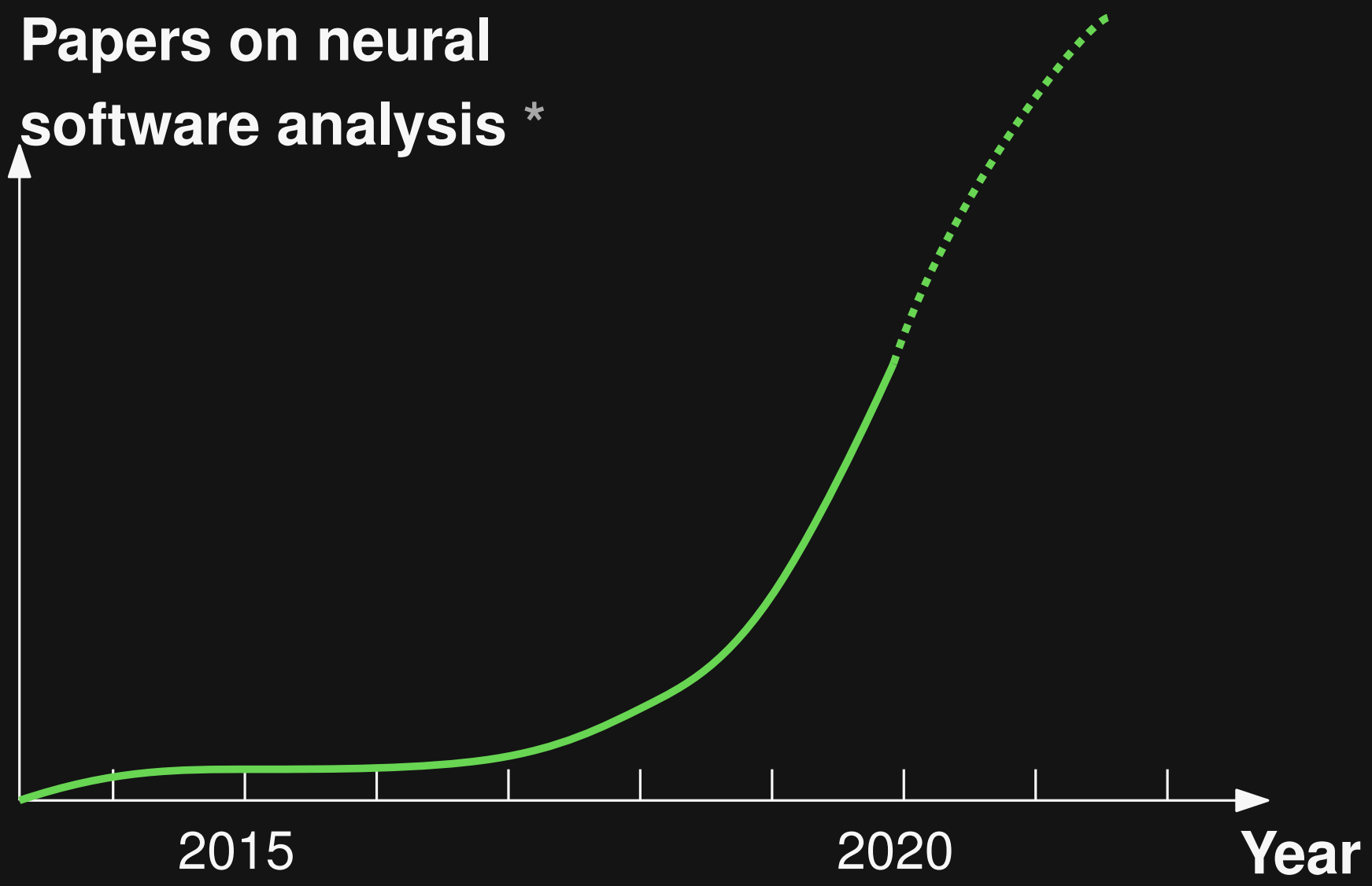
	Spearman rank correl.	
	All methods	Methods with F1 $\geq$ 0.5
CNN (regular)	0.08	0.24
CNN (copy)	0.49	0.55
Transformer (reg.)	-0.20	0.02
Transformer (copy)	0.47	0.55

More human-like predictions are more accurate

Implications

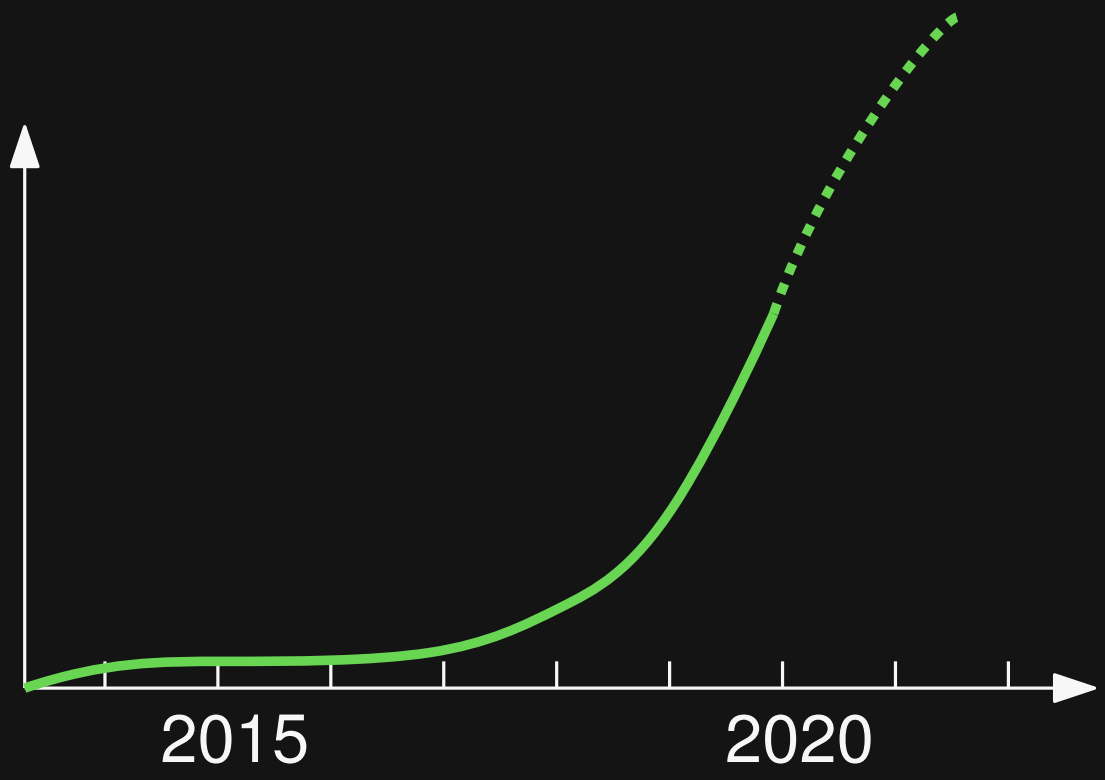
- **Direct human-model comparison**
  - Helps understand why models (do not) work
- **Should create models that mimic humans**
  - Use human attention during training
  - Design models that address current weaknesses
    - E.g., understanding string literals

The Road Ahead



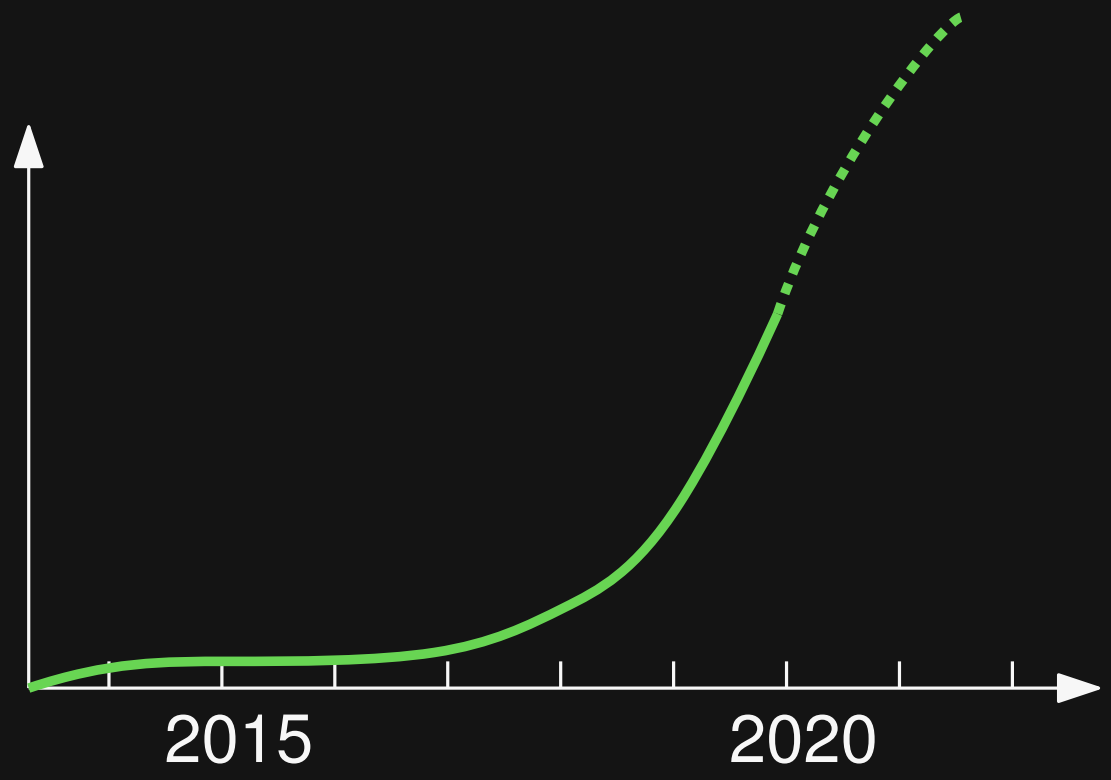
* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

The Road Ahead



The Road Ahead

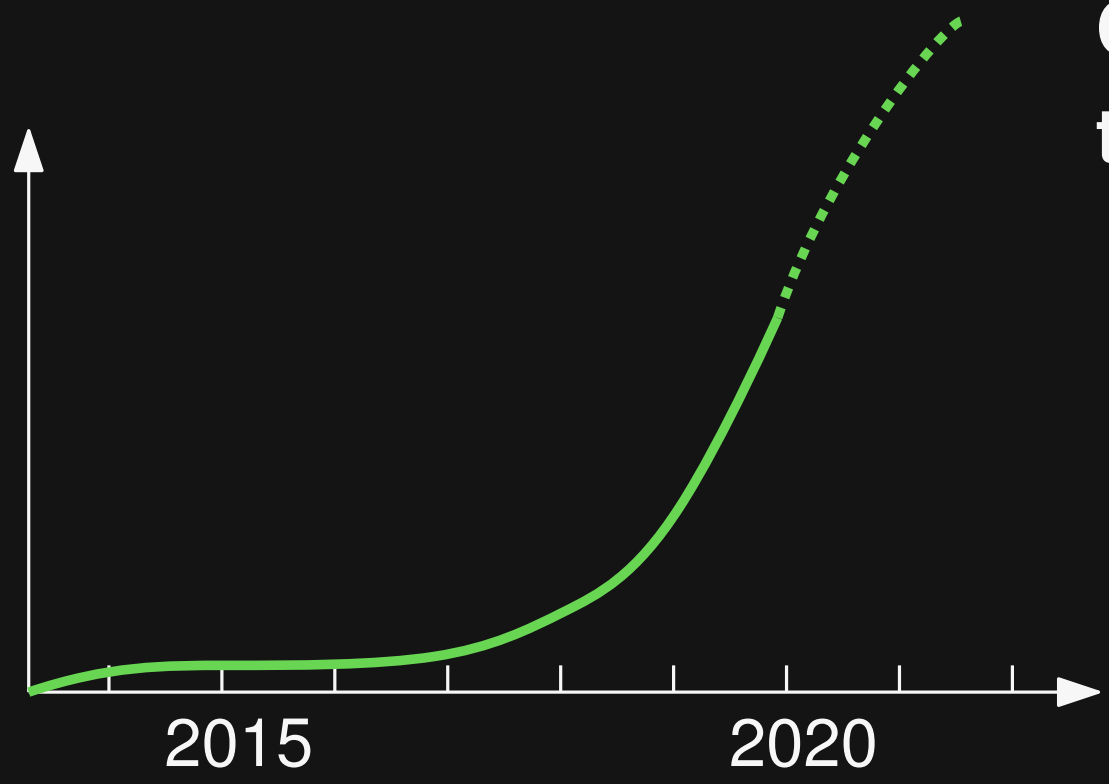
General-purpose language models



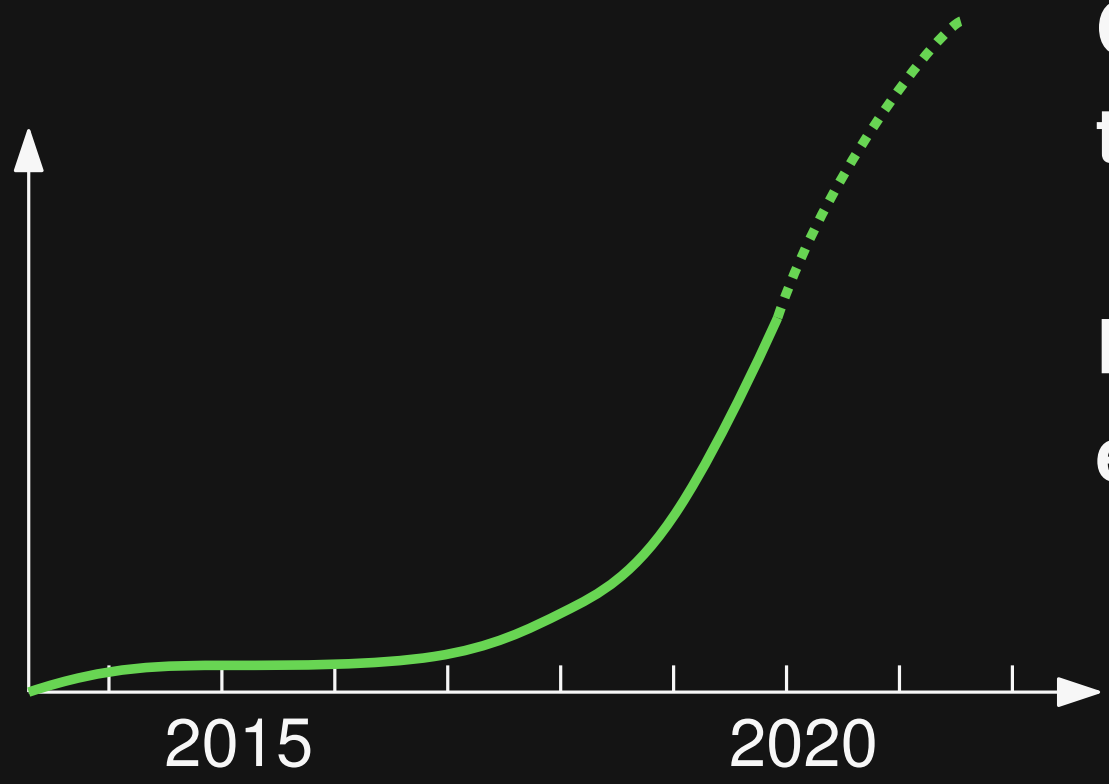
The Road Ahead

General-purpose language models

Combining neural & traditional analysis



The Road Ahead



General-purpose language models

Combining neural & traditional analysis

Reasoning about executions

Neural Software Analysis

The Good, the Bad, and the Ugly

Bug detection with Nalin

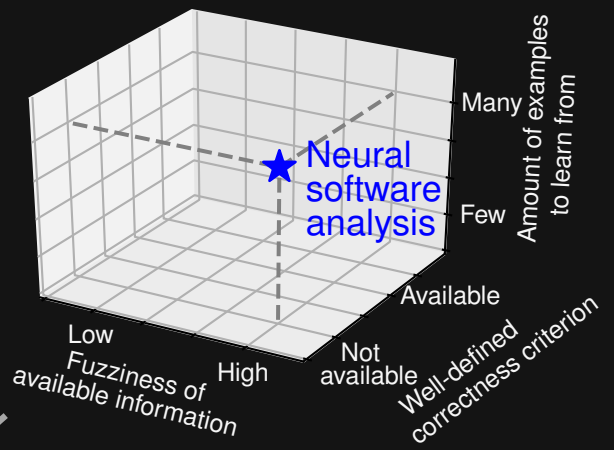
Type prediction with TypeWriter

Neural Software Analysis

The Good, the Bad, and the Ugly

Bug detection with Nalin

Type prediction with TypeWriter

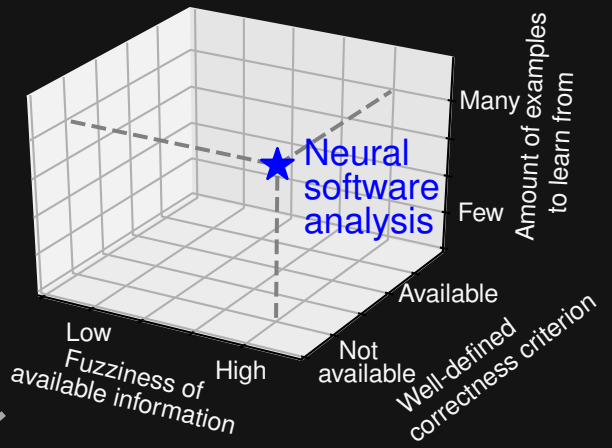


Neural Software Analysis

The Good, the Bad, and the Ugly

Bug detection with Nalin

Type prediction with TypeWriter



Neural Software Analysis

The Good, the Bad, and the Ugly

```
{
    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

```
{
    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();
}
```

Human attention