

Thinking Like a Developer?

Comparing the Attention of Humans
with Neural Models of Code



Michael Pradel

Software Lab – University of Stuttgart

Joint work with Matteo Paltenghi

Executive Summary

Direct comparison:

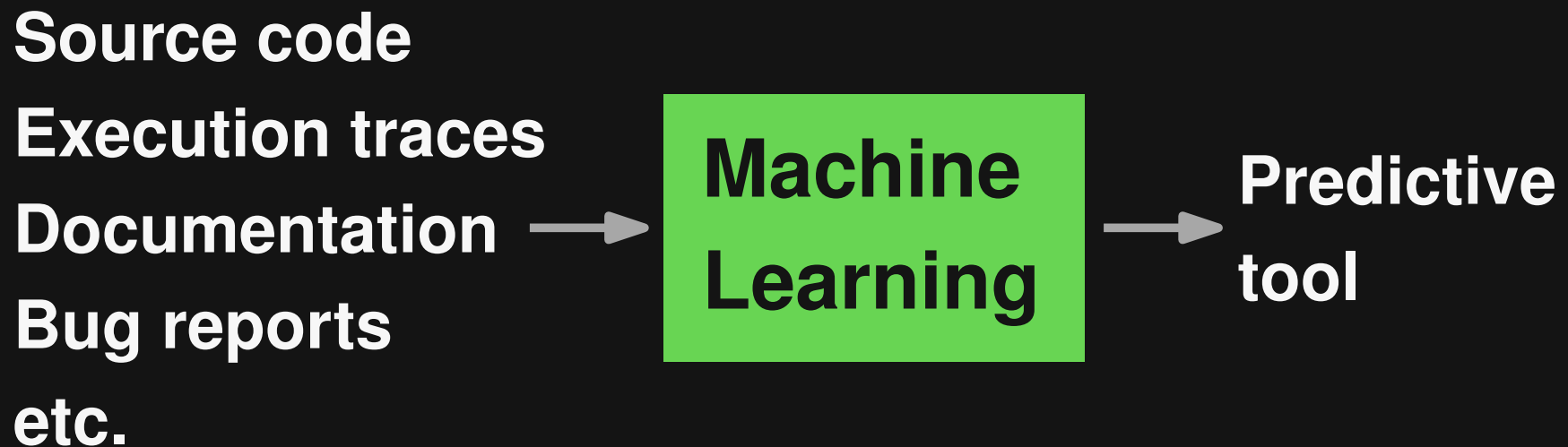
Developers vs. neural models of code

- Humans still (clearly) outperform models
- Partial agreement on what code to focus on
- Models ignore some tokens that developers deem important
- Human-model agreement correlates with prediction accuracy

Should try harder to mimic humans

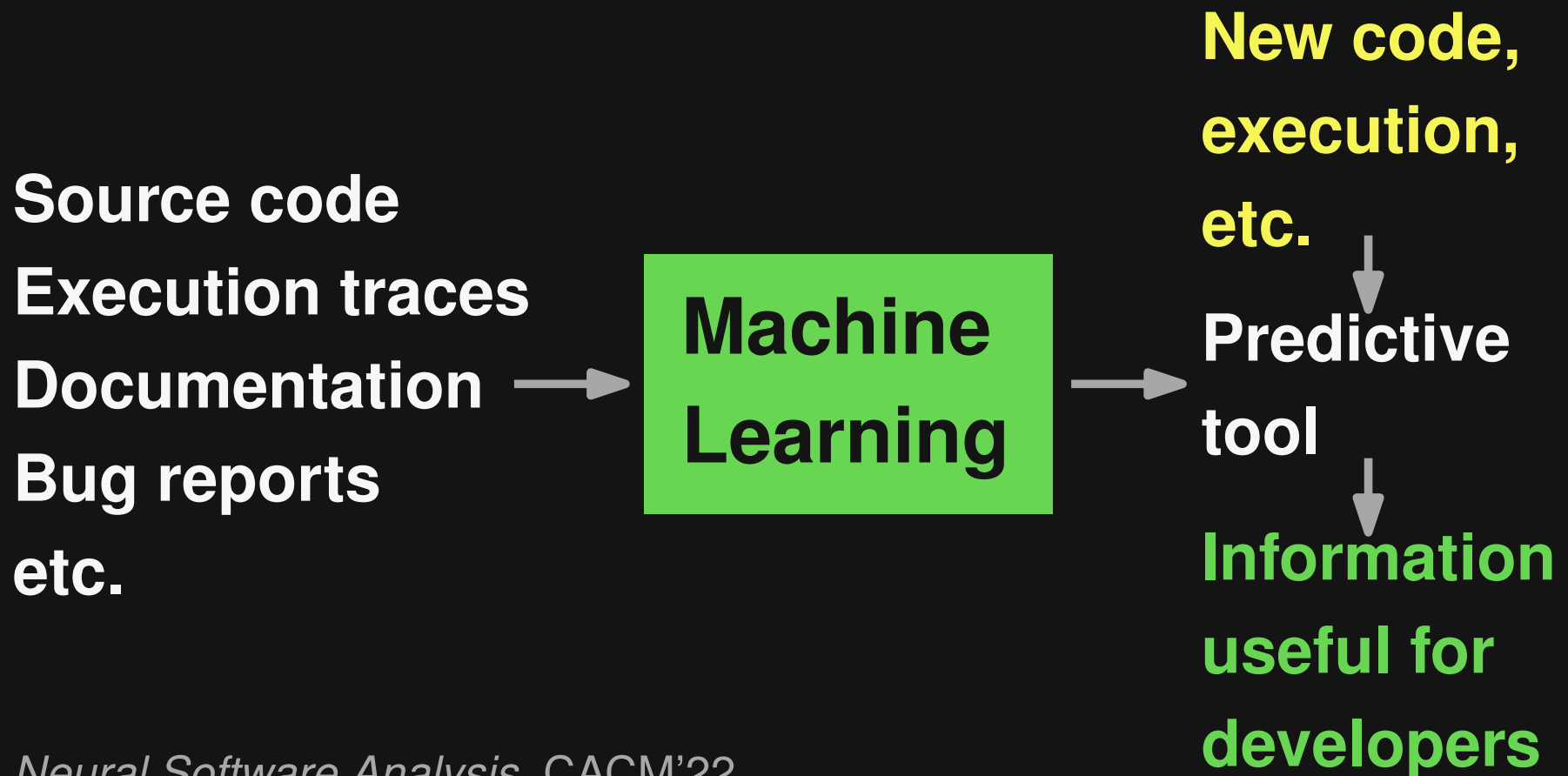
Neural Software Analysis

Learning developer tools from large software corpora



Neural Software Analysis

Learning developer tools from large software corpora



Common Tasks

Type prediction

Bug detection

Code summarization

Code completion

Program repair

Common Tasks

Type prediction

Bug detection

Code summarization

Code completion

Program repair

Humans could also do it.

→ Added value: Automation

Understanding Models of Code

- **Emphasis of most papers: Accuracy**

- **Mostly unclear:**

What do these models actually learn?

- Intellectually unsatisfying
- Risk of coincidental accuracy

Developers vs. Neural Models

Do **neural models** reason about code **similarly to human developers?**

- If yes: Good news
- If no: Should mimic developers more closely

Methodology

Idea: Compare Humans & Models



Developers

vs.


**Machine
Learning**

Neural models of code

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

Task 1: 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*

Task 2: Program Repair

```
public double sqrt(double x, double epsilon) {  
    double approx = x / 2d;  
    while (Math.abs(x - approx) > epsilon) {  
        approx = 0.5d * (approx + x / approx);  
    }  
    return approx;  
}
```

Input: Method with a buggy line



Output: Fixed line

```
while (Math.abs(x - approx * approx) > epsilon) {
```

Dataset: 16 bugs from QuixBugs (Java) *

* *QuixBugs: A Multi-Lingual Program Repair Benchmark Set Based on the Quixey Challenge, SPLASH'17 (Companion)*

Capturing Human Attention

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

Capturing Human Attention

Task 1: Code Summarization

Participant

Inspect the code and select the correct method name:

View guidelines. STATUS: Ready to answer.

1. testDeepConflictingReturnTypes
2. testAction
3. testInitializingDoesntTakeReadAction
4. testToStringDoesntExhaustIterator
5. disableSyncScrollSupport
6. calculateTimestamp
7. testCorrectProgressAndReadAction

2

Manager.getInstance().

CODE INSPECTION AREA

ANSWER SELECTION AREA

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

Capturing Human Attention

Task 2: Program Repair

The screenshot shows a web-based program repair interface. At the top, there is a 'Help' box with instructions: 'Hover to see the instructions again. Hover if you need more information on the algorithm.' To the right is a 'Submit' section with the text: 'Please press the submit button once you have fixed the bug, or indicate that you are not able to.' Below this text are two buttons: 'Submit' and 'Can't fix'. A 'Snippet Info' box is also visible. On the left, a 'Submit' button is shown with a red arrow pointing to the 'Submit' button in the top right. Below the top bar is a progress indicator showing '0%'. The main area is a 'Code Editor' containing Java code. A red arrow points to a specific line of code, labeled 'Buggy Line', which is highlighted in yellow. The code is as follows:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19 new ArrayList<Integer>()  
20  
21
```


Capturing Human Attention

- **27 participants:** Software engineers, graduate students
- **98 bug fixing records**
- **5–7 records for each of 16 bugs**
- **On average per record: 983 unblur events and 13 edit events**

Capturing Human Attention

Summarize fine-grained attention record into **attention map**:

```
public class SQRT {  
    public static double sqrt(double x, double epsilon)  
    {  
        double approx = x / 2d;  
        while (Math.abs(x - approx) > epsilon) {  
            approx = 0.5d * (approx + x / approx);  
        }  
        return approx;  
    }  
}
```

Model Attention

Task 1: Code summarization

- 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

Model Attention

Task 2: Program repair

- LSTM-based, sequence-to-sequence:

SequenceR

SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair, TSE'21

- Regular attention and copy attention

- AST-based transformer: Recoder

A Syntax-Guided Edit Decoder for Neural Program Repair, FSE'21

- Regular attention only

Results

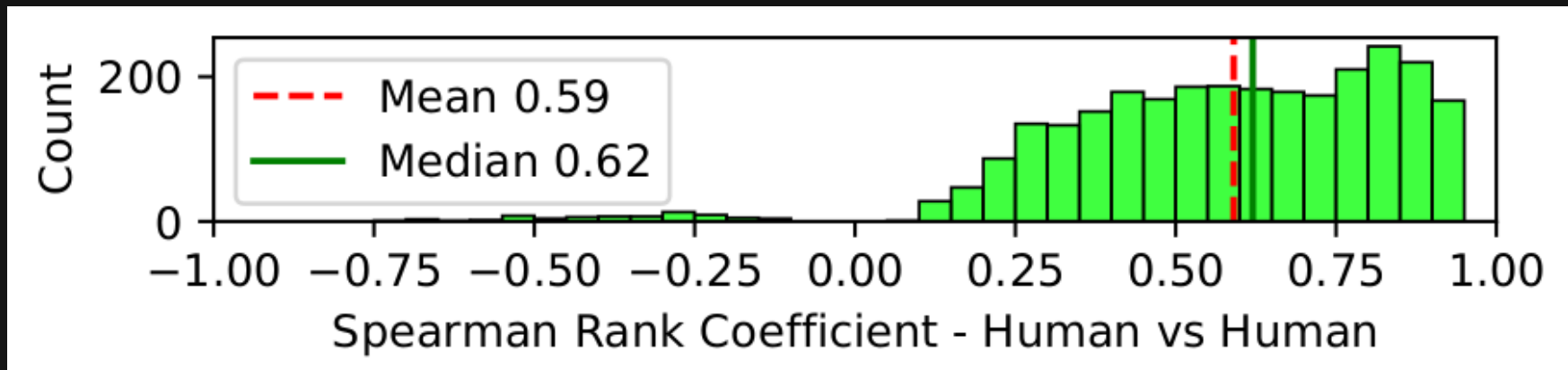
Human-Model Agreement

Do developers and models **focus on the same tokens?**

- Given for each code example
 - Human attention vector \vec{h}
 - Model attention vector \vec{m}
- **Measure agreement** between them
 - **Spearman rank correlation:** $\frac{\text{cov}(rg_{\vec{h}}, rg_{\vec{m}})}{\sigma_{rg_{\vec{h}}}, \sigma_{rg_{\vec{m}}}}$

Results: Summarization

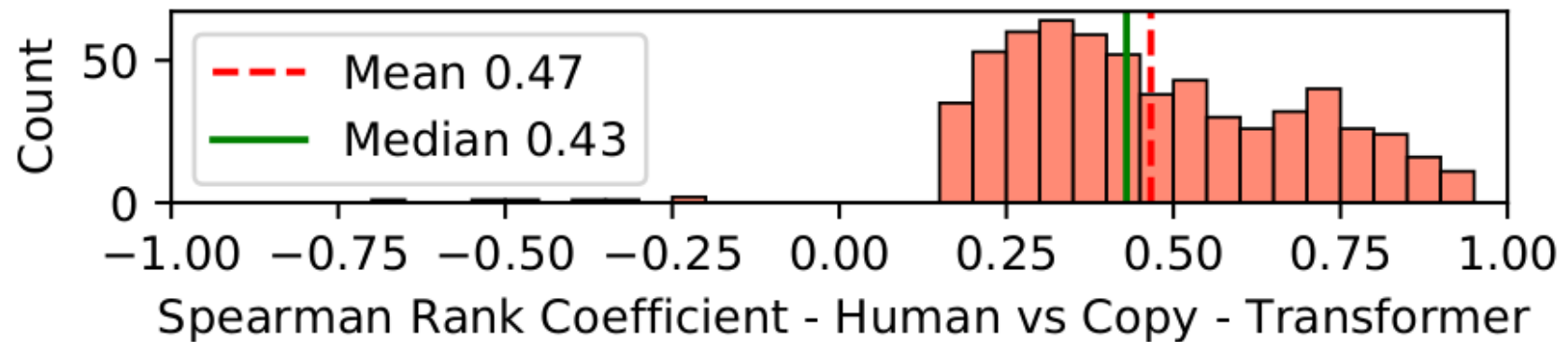
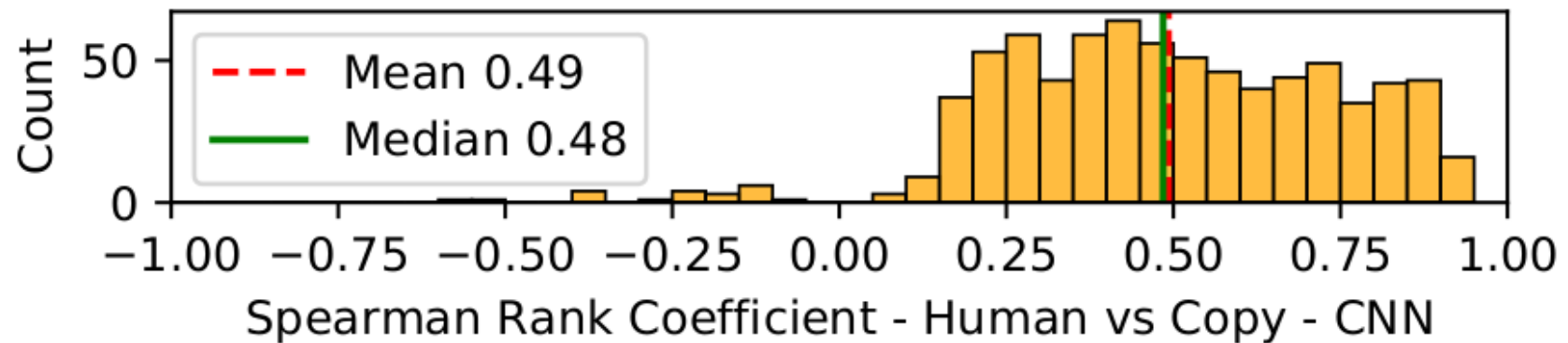
Human-human agreement:



Developers mostly agree on what code matters most

Results: Summarization

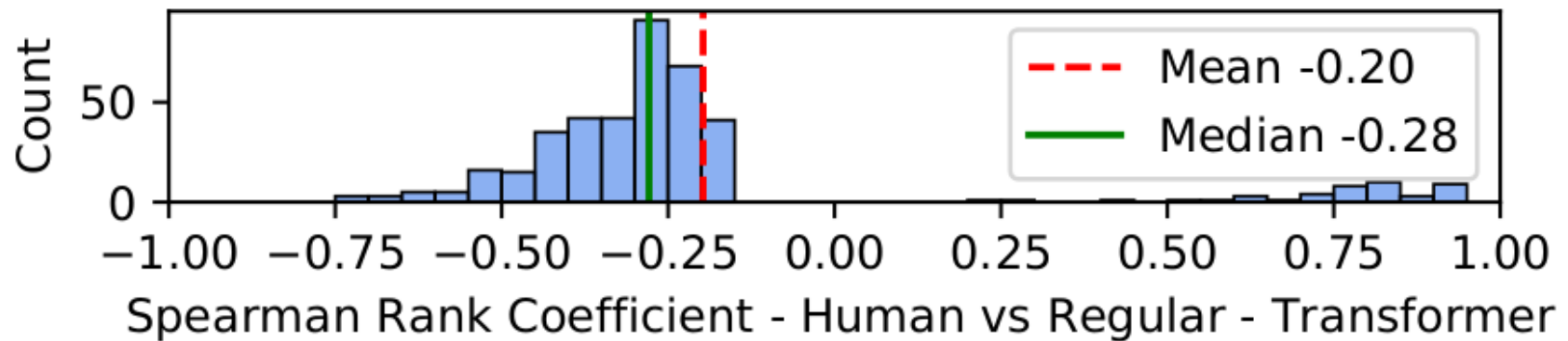
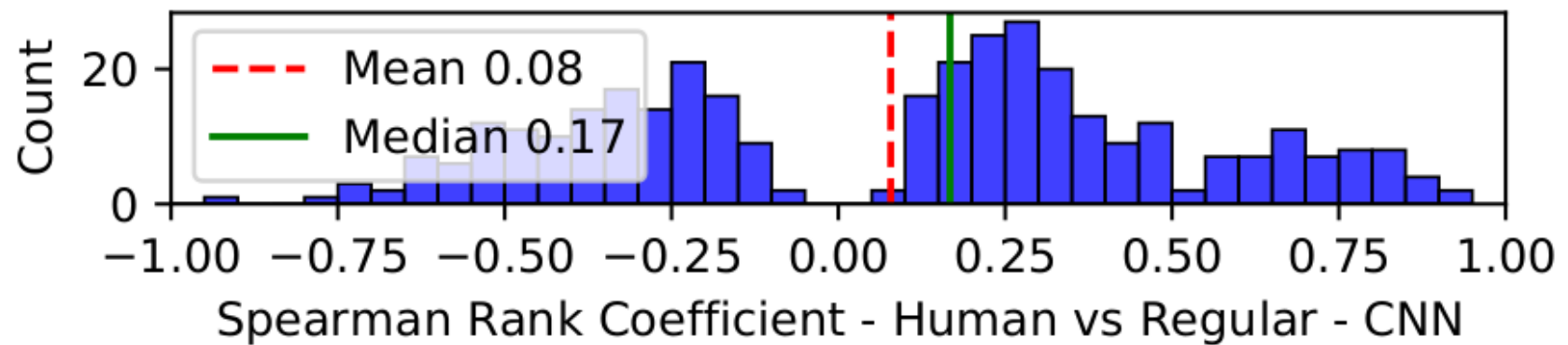
Human vs. copy attention:



Empirical justification for copy attention

Results: Summarization

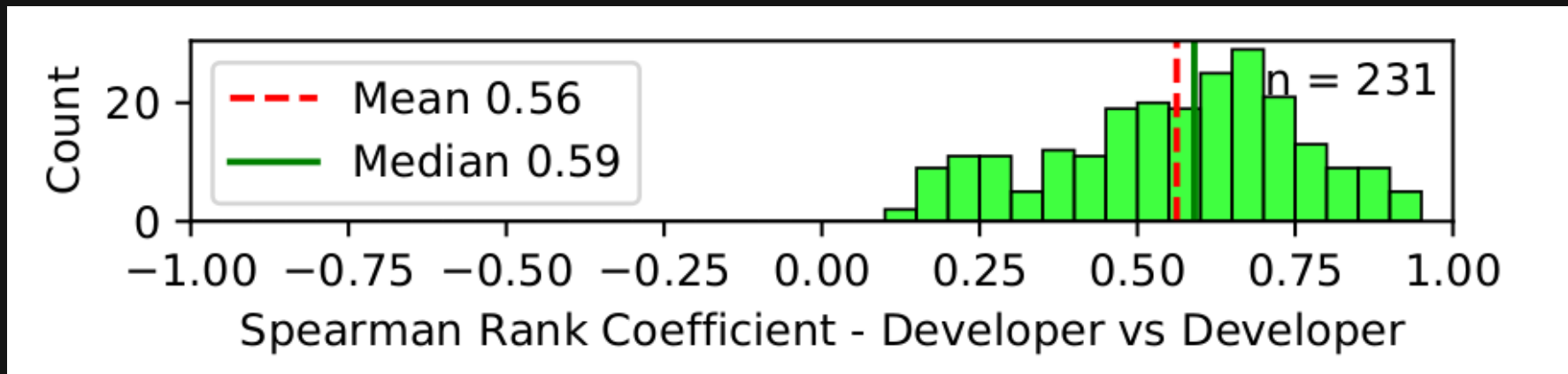
Humans vs. regular attention:



Lots of room for improvement!

Results: Program Repair

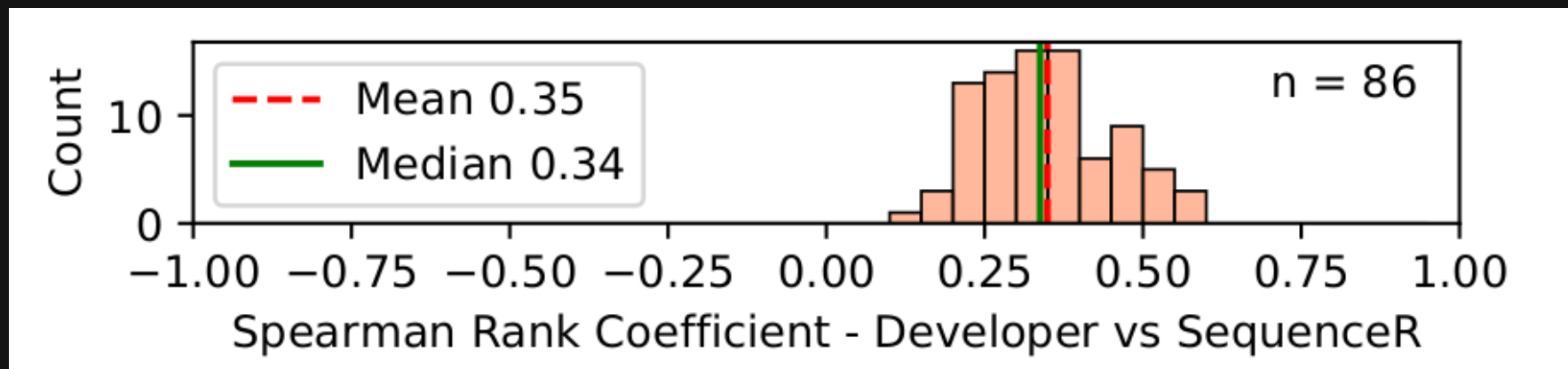
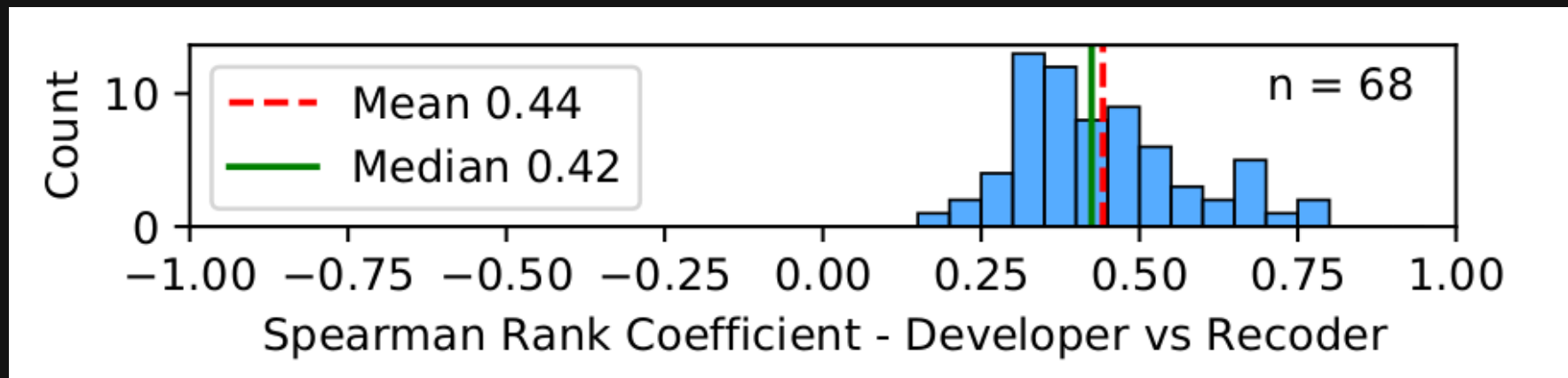
Human-human agreement:



Developers mostly agree on what code matters most

Results: Program Repair

Human-model agreement:



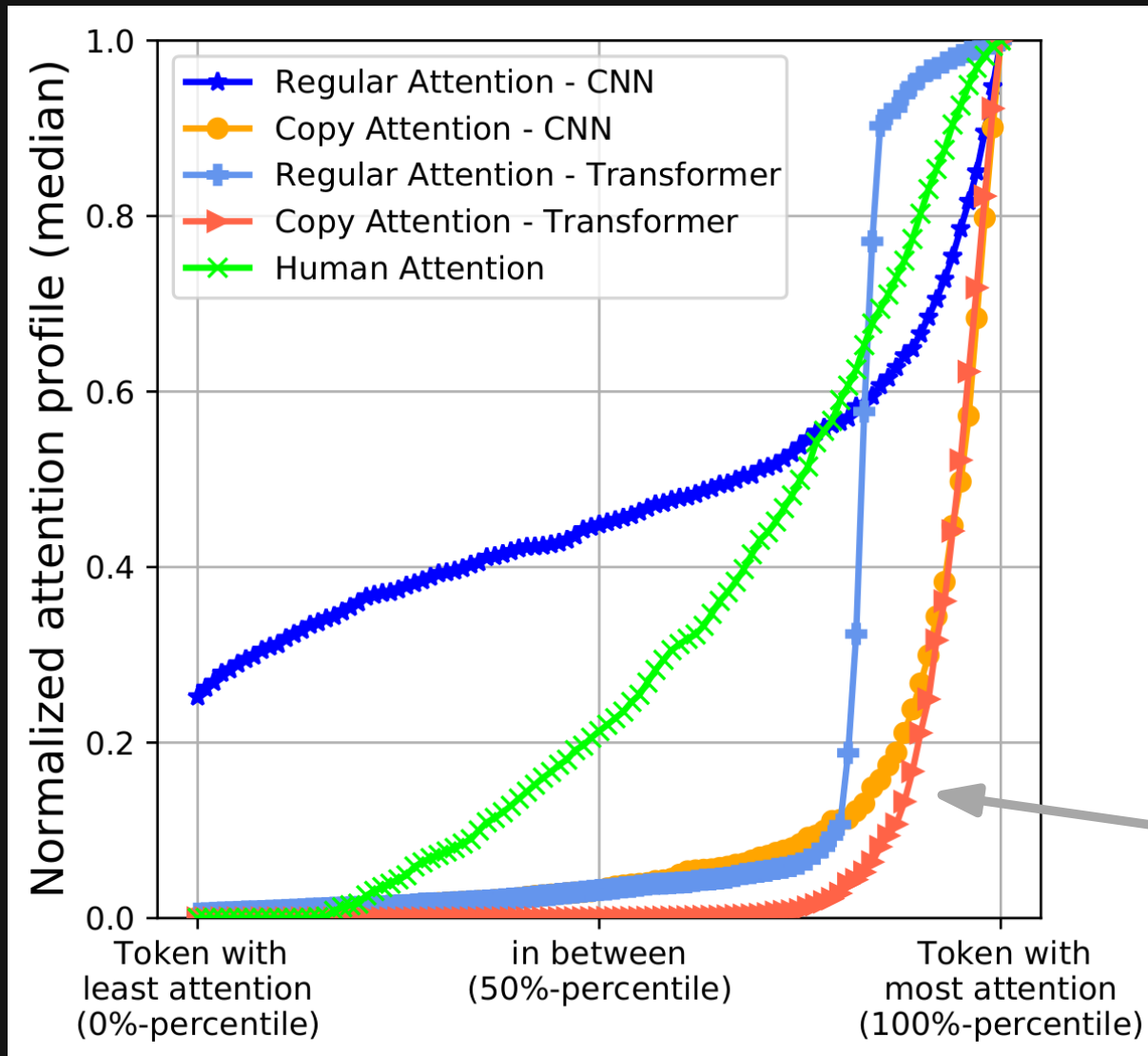
Some room for improvement

Divided vs. Selective Attention

How to **distribute attention** over the given code?

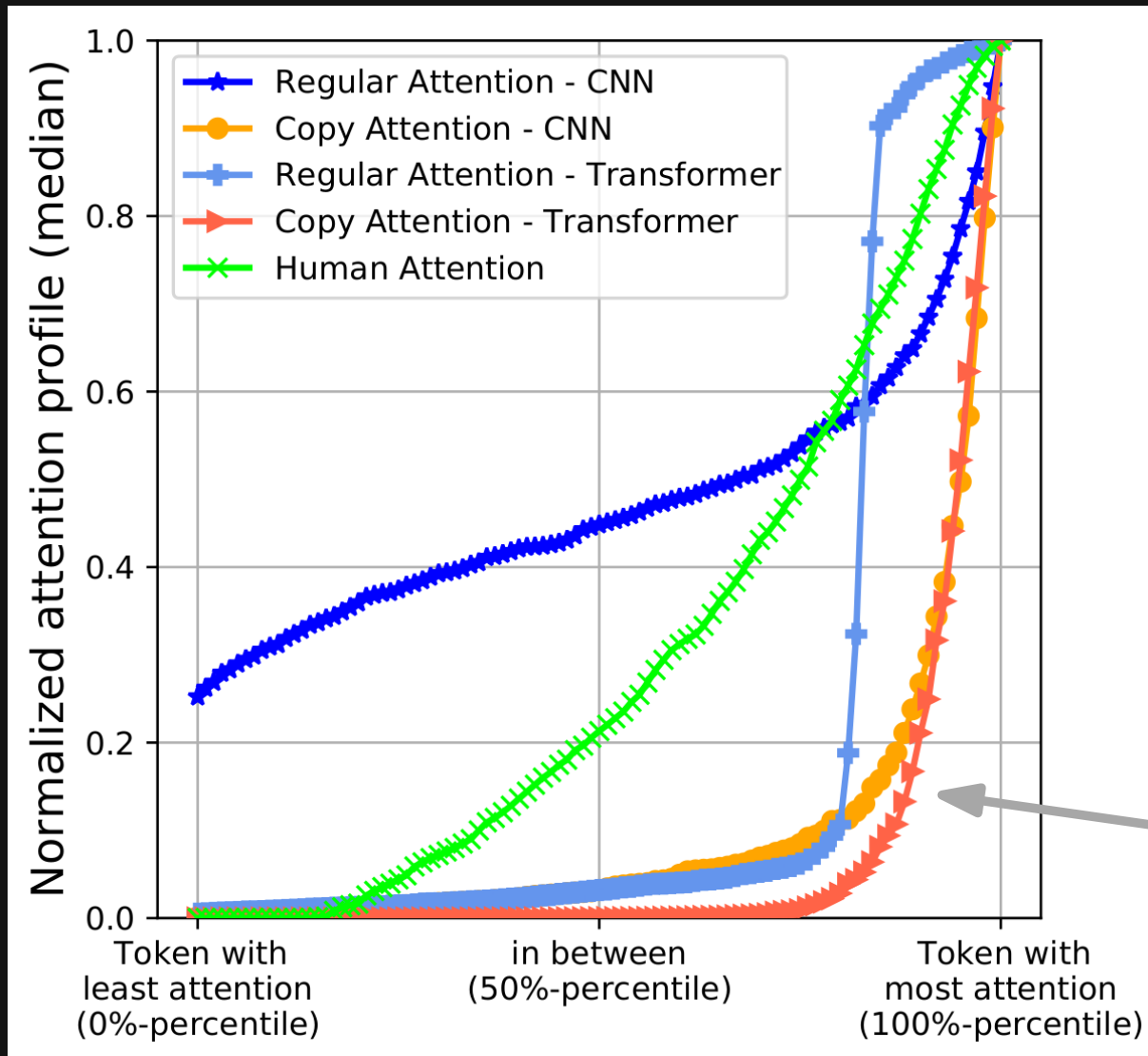
- One extreme: Equally distribute over all tokens
- Other extreme: Focus on a few tokens only

Results: Summarization



More dented
curve: Focus on
few tokens only

Results: Summarization



**No model
closely
matches
developers**

**Overspecial-
ization to a
few tokens**

More dented
curve: Focus on
few tokens only

Results: Program Repair

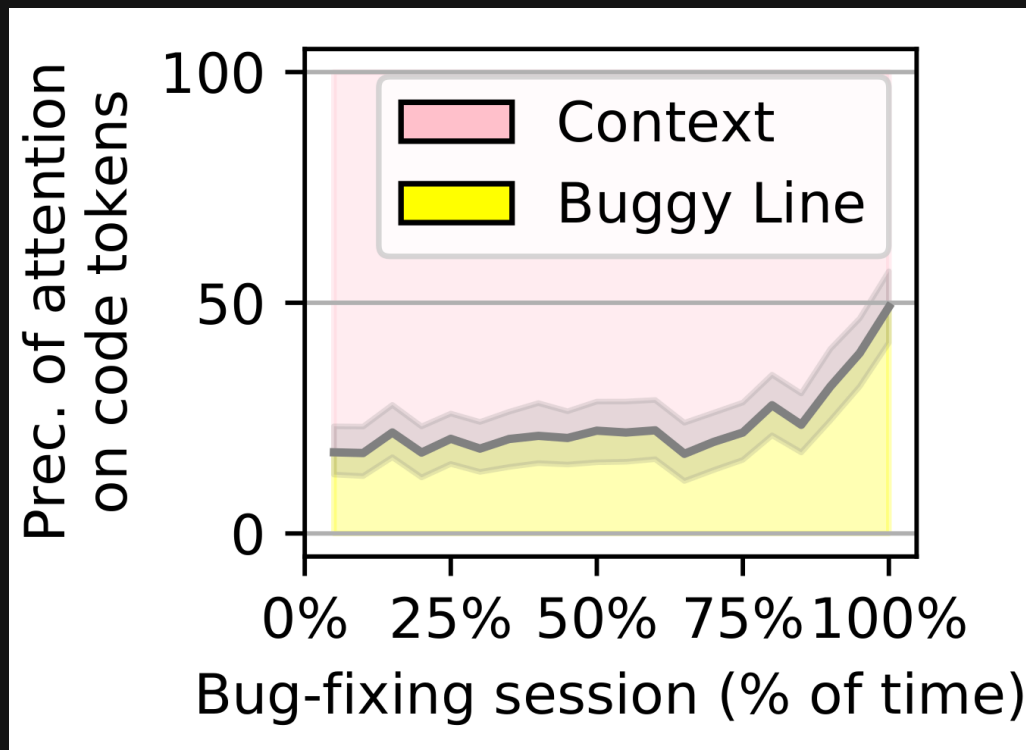
Focus on **buggy line vs. code context**:

| | Buggy line | Context |
|------------|------------|---------|
| Developers | 37% | 63% |
| SequenceR | 67% | 33% |
| Recoder | 13% | 87% |

Again, no model closely matches developers

Results: Program Repair

Human attention evolves over time:



Models could mimic human behavior:
First understand, then fix

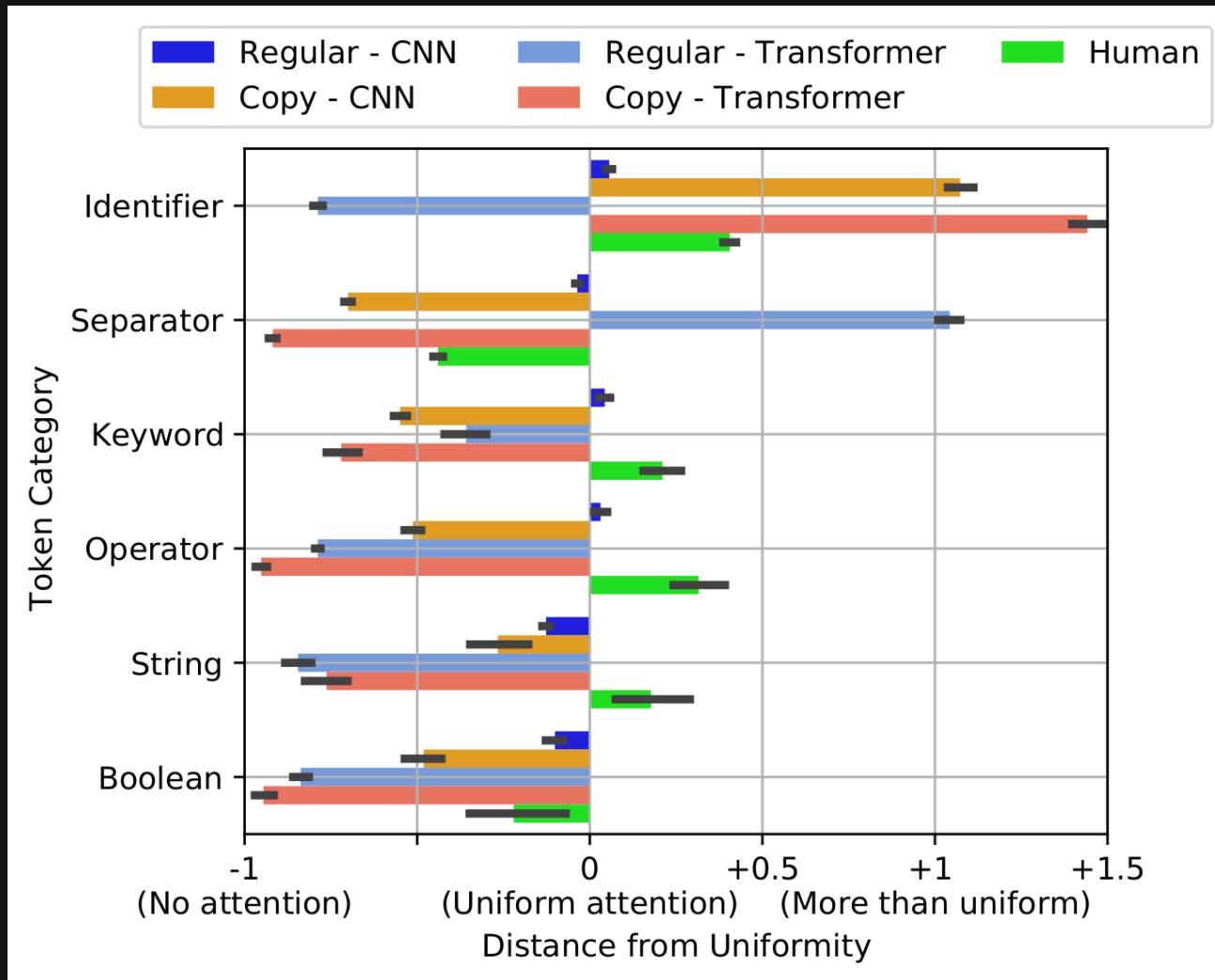
Tokens to Focus On

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

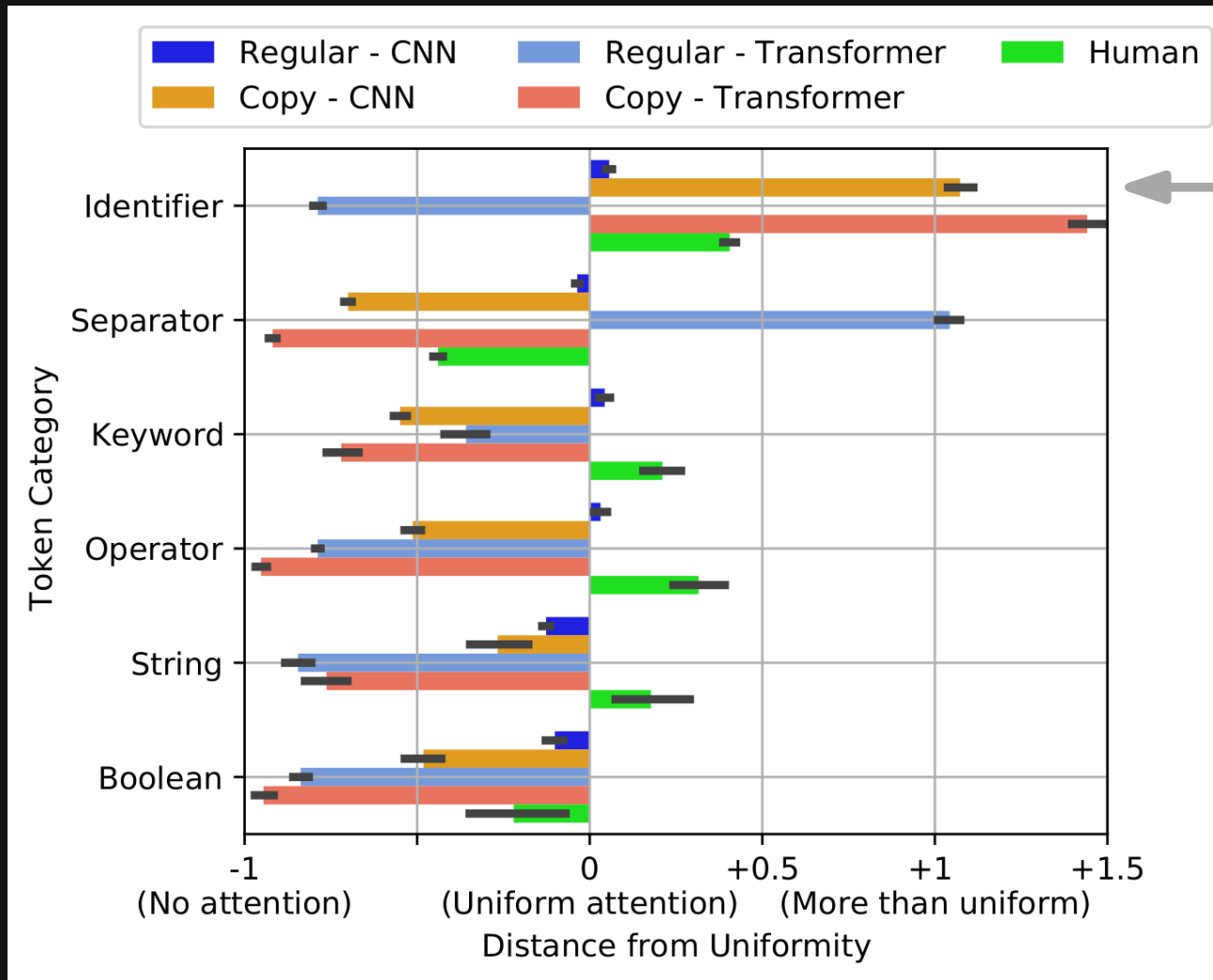
Results: Summarization

Distance from uniformity:



Results: Summarization

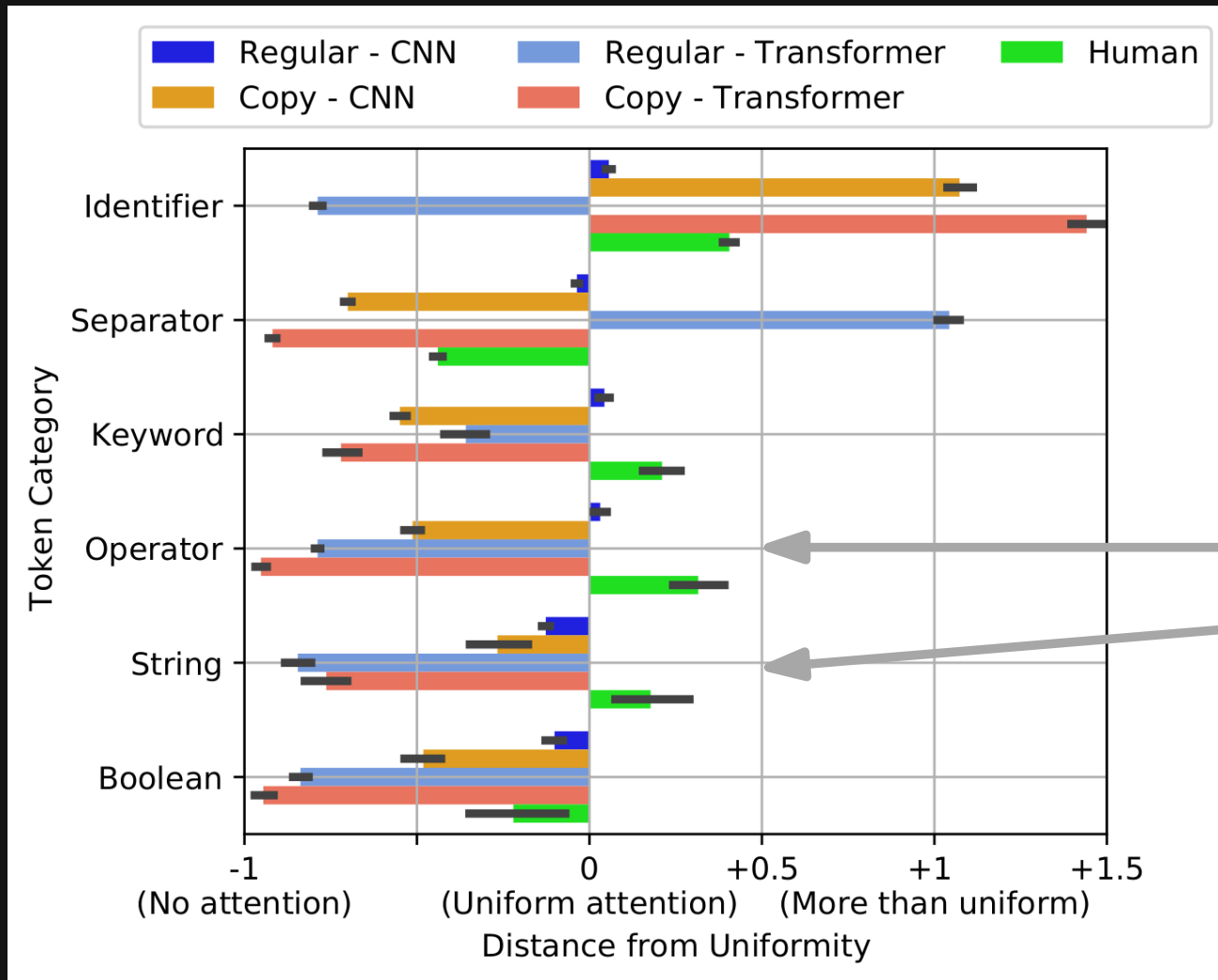
Distance from uniformity:



Identifiers
are deemed
important

Results: Summarization

Distance from uniformity:



**Models
mostly
ignore
some kinds
of tokens**

Results: Summarization

Example from Transformer 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.getResponseAsStream()).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.getResponseAsStream()).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

Results: Summarization

Example from Transformer model:

The diagram illustrates the difference between neural model attention and human attention. It features two versions of a Java code snippet. The top version, labeled 'Regular attention of neural model', shows blue highlights on various tokens across the code, with a black arrow pointing from the 'Requesting new token' log message to the 'getStatusLine()' method call. The bottom version, labeled 'Human attention', shows red highlights only on the relevant parts of the code (the log message, the status check, the exception, and the token retrieval), with a black arrow pointing from the log message to the 'result.getTextTrim()' line. A yellow box with black text is positioned between the two code blocks, stating: 'Model “wastes” attention on understanding syntax'.

```
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.getResponseAsStream()).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

Model “wastes” attention on understanding syntax

Human attention

Results: Summarization

Example from Transformer 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.getResponseAsStream()).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" : error.getText());
}
myToken = result.getTextTrim();
}
```

**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.getResponseAsStream()).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

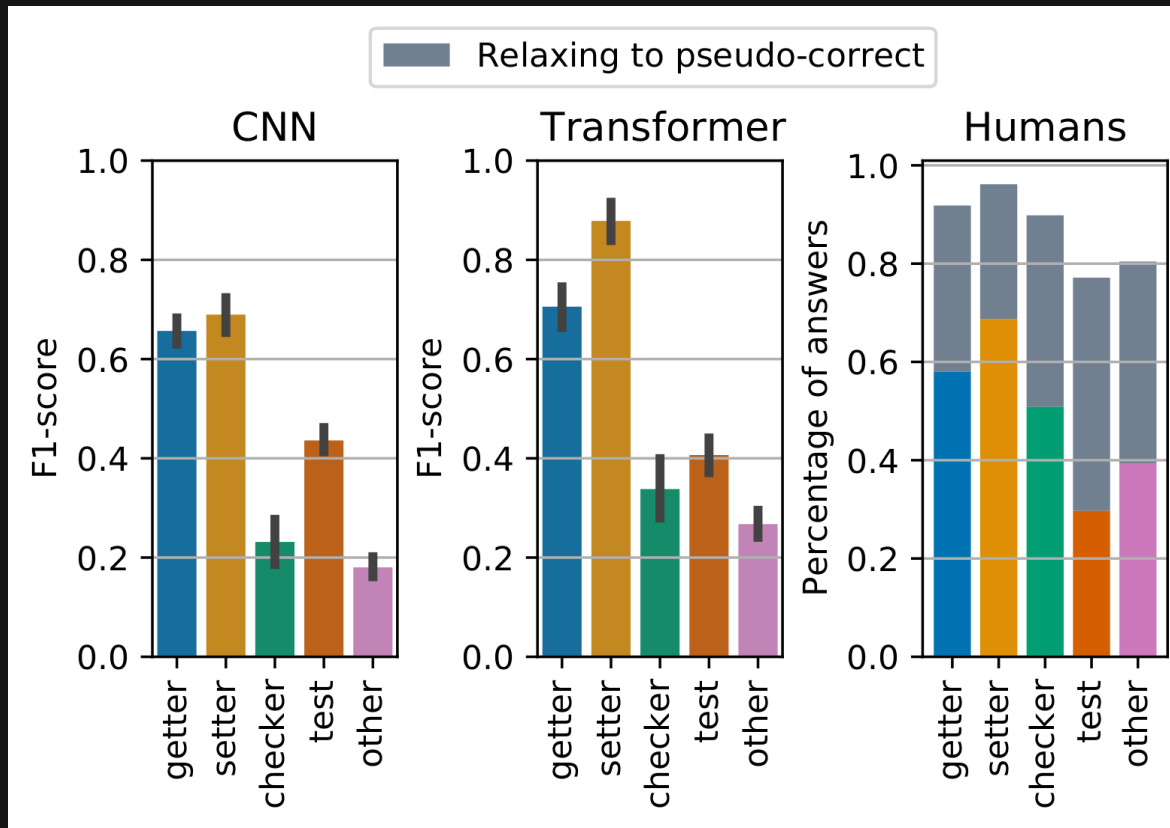
Effectiveness

Comparing developers and models w.r.t. their **effectiveness at solving the task**

- Strengths and weaknesses?
- Can current models compete with developers?

Results: Summarization

Comparing different kinds of methods:



Models underperform on non-trivial methods

Results: Program Repair

Success rate during program repair:

| | Plausible patch ratio | |
|------------------|-----------------------|-----------------------|
| | Top-5 | Top-100 |
| SequenceR | 2/80 (2.5%) | 17/1395 (1.2%) |
| Recoder | 2/80 (2.5%) | 10/908 (1.1%) |

Results: Program Repair

Success rate during program repair:

| | Plausible patch ratio | |
|------------|-----------------------|----------------|
| | Top-5 | Top-100 |
| SequenceR | 2/80 (2.5%) | 17/1395 (1.2%) |
| Recoder | 2/80 (2.5%) | 10/908 (1.1%) |
| | 5-7 developers/bug | |
| Developers | 68/98 (69.4%) | |

Models are far from human effectiveness

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 |



**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

Conclusions

- **Available for future research:**

- Interface for capturing human attention
- Datasets of human attention records

- **More details:**

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