Thinking Like a Developer? Comparing the Attention of Humans with Neural Models of Code

Matteo Paltenghi
Department of Computer Science
University of Stuttgart
Stuttgart, Germany
mattepalte@live.it

Michael Pradel
Department of Computer Science
University of Stuttgart
Stuttgart, Germany
michael@binaervarianz.de

***Abstract*—Neural models of code are successfully tackling various prediction tasks, complementing and sometimes even outperforming traditional program analyses. While most work focuses on end-to-end evaluations of such models, it often remains unclear what the models actually learn, and to what extent their reasoning about code matches that of skilled humans. A poor understanding of the model reasoning risks deploying models that are right for the wrong reason, and taking decisions based on spurious correlations in the training dataset. This paper investigates to what extent the attention weights of effective neural models match the reasoning of skilled humans. To this end, we present a methodology for recording human attention and use it to gather 1,508 human attention maps from 91 participants, which is the largest such dataset we are aware of. Computing human-model correlations shows that the copy attention of neural models often matches the way humans reason about code (Spearman rank coefficients of 0.49 and 0.47), which gives an empirical justification for the intuition behind copy attention. In contrast, the regular attention of models is mostly uncorrelated with human attention. We find that models and humans sometimes focus on different kinds of tokens, e.g., strings are important to humans but mostly ignored by models. The results also show that human-model agreement positively correlates with accurate predictions by a model, which calls for neural models that even more closely mimic human reasoning. Beyond the insights from our study, we envision the release of our dataset of human attention maps to help understand future neural models of code and to foster work on human-inspired models.**

I. Introduction

Neural models that analyze source code [46] have become extremely effective on various tasks, such as code summarization [5], [6], [31], bug detection [28], [48], bug injection [44], bug repair [16], and type inference [3], [26], [40], [47], [65]. In essence, these models learn implicit rules, patterns, and heuristics from a large number of code examples, and then apply them to previously unseen code. An implicit assumption is that the models reason about code in a way similar to human software developers. However, it currently remains unclear to what extent the computations performed by trained neural models actually resemble how humans reason about code.

Understanding the relation between neural and human reasoning is an important step toward better understanding why neural models of code work, or sometimes do not work. Current models are mostly black boxes and it remains difficult to understand why a model succeeds or fails. Gaining a better understanding of these models is crucial to validate that a model is right for the right reasons, instead of, e.g., picking up coincidental but meaningless correlations in a dataset. Moreover, it will ultimately help build more effective models by identifying current weaknesses and confirming why particular techniques are effective.

A popular way to increase the transparency of neural networks is an attention mechanism [11], [60]. It assigns a weight to each part of the input, showing which parts of an input a model is most interested in. Recent work in natural language processing studies the effectiveness of attention weights as an explanation technique by comparing them with alternative explanation approaches [32], [66]. As a first attempt to understand attention weights in models of code, Bui et al. [13] artificially remove individual statements, observe how it affects a model's predictions, and then compare the importance of an input segment to attention weights. A question that remains open is how attention weights in models of code relate to how humans reason about source code.

This paper presents the first systematic study to compare neural models of code with human reasoning. We compare the attention weights of neural models with the attention that humans pay when reasoning about source code examples. Our work focuses on the method summarization task, which is interesting, as it requires a thorough understanding of a potentially complex piece of source code, and a popular task for neural models of code. For this task, we study two neural models [1], [5], which offer two kinds of attention weights: *regular attention*, showing what tokens the model focuses on, and *copy attention*, showing what tokens the model considers to copy verbatim into the output. We compare these attention weights to humans working on the same method summarization task while participating in our study.

A key challenge for our work is capturing the attention of humans while they reason about source code. We address this challenge through a novel methodology for recording human attention paid while solving a code-related task. Intuitively, the idea is to approximate the human attention with the time a human looks at a particular code element. To measure which parts of the code the participants of our study pay attention to, we show blurred source code to the participants, who must (temporarily) unblur individual tokens of the source code to

understand it. We gather a total of 1,508 human attention maps from 91 participants, including five examples for each of 250 Java methods sampled from ten real-world projects.

Based on the dataset of human attention maps, we thoroughly study to what extent human attention matches the attention weights of learned models. We envision that a better understanding of the human-machine relationships could inspire either future explainability methods or human-inspired models of code. Our study addresses the following research questions.

RQ1: Are the intrinsic attention weights of neural models correlated with human attention? Answering this question quantifies how closely neural models resemble the human reasoning. We find that neural models and humans agree on copy attention but not on regular attention, which experimentally confirms the benefit of copy attention in models of code.

RQ2: How does the distribution of attention across tokens differ between models and humans? Answering this question could highlight differences between models and humans that help understand their strengths and weaknesses. We find that none of the studied models closely mimics how humans distribute their attention. Similar to copy attention, humans sometimes do not explore all tokens, but ignore tokens not relevant for the prediction task.

RQ3: Do neural models and humans attend to the same kinds of tokens, e.g., identifiers, separators, operators, and keywords? If humans and models attend to different kinds of tokens, then future neural models may want to take inspiration from the humans to better mimic their way of understanding source code. We find that neural models pay more attention to basic syntactic tokens, whereas humans pay more attention to strings, operators, and keywords.

RQ4: Do learned models and humans struggle with the same kinds of examples? Answering this question could reveal complementary strengths, and it may motivate work toward addressing the current weaknesses of neural models. We find that models and humans largely agree on what methods are hard to summarize, showing the need for more effective models, especially on longer methods. We also find that models are most effective on getter and setter methods, whereas humans understand a wider range of methods.

RQ5: How does the agreement between models and humans relate to model effectiveness? This question is relevant to check if creating models that more closely resemble human reasoning is likely to yield more effective models. We find that if a model pays attention to the same tokens as humans, then the chance of making a correct prediction increases. This result motivates research on human-inspired neural models.

In summary, this paper makes the following contributions:

- A novel methodology for recording human attention during code-related tasks.
- An in-depth study of similarities, differences, and correlations between the attention paid by models and humans, and its impact on model effectiveness.
- A dataset of 1,508 human attention maps, which we make available for future work.

Our replication package, tooling, and all data are permanently available at https://github.com/MattePalte/thinking-like-a-developer.

II. Background

a) Attention-Based Models: Neural network models have recently been proposed to help developers on various tasks [46]. Many of them use attention layers [1], [4]–[7], [31]. An attention mechanism lets the network learn a weighted sum of the input units to compose a weighted context vector for downstream modules [11]. Besides improving a model's predictions, attention weights also give some transparency to models by showing which parts of the inputs are most relevant to a model. Attention is used to attend to different parts of source code, e.g., source code tokens [5], paths in the abstract syntax tree [6], or nodes of a graph representation of source code [4], [17]. The attention weights of these models tell us which tokens, paths, or graph nodes explain a specific prediction. The meaning of attention weights is still under active discussion [10], [13], [32], [66], but to the best of our knowledge, no one has ever studied attention weights by comparing them to the attention of programmers while performing a task on source code.

b) Copy Attention: Many models of code use copy attention to learn to copy relevant tokens from the input code verbatim to the predicted output. This idea was introduced with the pointer network architecture [61] for solving combinatorial problems characterized by an output vocabulary of arbitrary size. A pointer network uses attention as a pointer to select a part of the input sequence as the output. The output of the attention mechanism is a probability distribution over the input. Copy attention has become popular to address the out-of-vocabulary problem in code summarization tasks, such as method naming. Allamanis et al. [5] notice that about 35% of the output tokens appear in the method body. Indeed, copy attention improves the effectiveness of models [1], [5], especially when using a dedicated attention layer [1], [21], [43]. A central assumption for these models is that the output tokens are either in the fixed-size vocabulary, which is learned during training, or in the input sequence.

III. Methodology

When studying the similarities and differences between attention-based neural models and humans, we face two challenges. (1) Creating an environment to capture the human attention on code-related tasks. (2) Comparing the captured human attention to the learned attention weights in neural models. We present our methodology for addressing these two challenges in Sections III-B and III-C, respectively, and we start by describing the prediction task our work focuses on.

A. Code Summarization Task

Among the many tasks that neural models of code are applied to [46], we select code summarization. One reason for this choice are the various neural models targeting this task [1], [5]–[7], [31], [63], which could be impacted by our

findings. Another reason is that the task requires a model to capture and summarize relatively large pieces of code, i.e., there are several code elements to attend to.

Two variants of the code summarization task exist: predicting the method name [5]–[7] and predicting a method-level comment [1], [19], [29]. Since both variants are popular in previous work, we select the method name prediction task, as it simplifies creating a corresponding human task. Nonetheless, our methodology could also be used to study the comment prediction task in future work. Thus the task for the model is to predict the original method name given the method body as an input. Specifically, the models we study predict the name as a sequence of tokens[1], e.g., "get", "client", "data" for a method called `getClientData`. We ask the human participants of our study to perform a variant of this tasks: inspect a method body and then select the correct method name among a set of seven alternatives. In addition to the correct solution, the alternatives consist of three names similar to the correct name and three randomly selected other names. The similar names are intended to stimulate the participant's reasoning process, and we select them from the nearest neighbors of the correct name in a tf-idf-based bag-of-words encoding of method names. The set of alternatives in Figure 1 includes `testInitializing DoesntTakeReadAction`, which is the real name of the method, three close-by alternatives (`testToStringDoesntExhaustIterator`, `testCorrectProgressAndReadAction`, `testAction`), and three random options (`disableSyncScrollSupport`, `testDeepConflictingReturnTypes`, `calculateTimestamp`). The tasks given to the models and the humans are closely related since both the model and the humans must inspect the method body and condense it into a summary of the functionality of the code.

As a dataset of methods, we sample 236 methods from an existing Java corpus [5]. The corpus contains code from several application domains and is a popular benchmark for neural models [6], [64].

B. Capturing Human Attention

The first major challenge is how to capture the attention of a human working on the code summarization task. We address this challenge through a novel, gamification-inspired attention annotation interface, called Human Reasoning Recorder (HRR). The participants working with the HRR are not aware that the ultimate goal is to capture how much attention they pay to specific code elements, but instead are simply asked to select a suitable name for a method. The interface is divided into two main areas (Figure 1): an *answer selection area*, which shows the names to choose from, and a *code inspection area*, which shows the code of the method body.

The key idea to capture which parts of the code a participant attends to is a deblurring mechanism. Initially, all code in the method body is blurred, and the participant must deblur tokens

[1]The term "token" in this paper means subtokens that result from tokenizing code as specified by the programming language and then further splitting identifiers based on camel-case and other conventions.

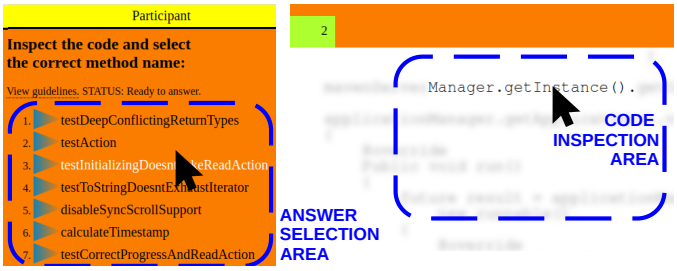

Fig. 1. Interface of the Human Reasoning Recorder.

to understand the code. Participants can deblur a token in two ways. On the one hand, moving the mouse over a token reveals the token and its neighbors for the time the mouse pointer is on them. Based on our initial pilot study and the fact that the average number of pieces of information that a human can hold in short-term memory is seven [42], we set the neighborhood to three tokens before and three tokens after the pointed-to token, ignoring neighbors in other lines.

On the other hand, participants can also click on tokens to make them permanently visible, even when the mouse moves away, and they can blur tokens again with another click. This mechanism allows participants to pinpoint the most important parts of the already explored code. To force participants to move the mouse away from the code while reading the candidate method names, the answer selection area is also blurred by default and becomes visible only when the mouse is in this section.

The HRR tracks which tokens a participant deblurs and for how long each token is visible. To this end, the interface continuously logs all mouse movements and clicks. Overall, the HRR captures the following information each time a participant summarizes a method:

Definition 1 (Human attention record)**.** The human attention record is a tuple $hr = (id, uid, evts, s, r)$, where id is a unique identifier for a method in the dataset, uid is the unique user identifier, $evts$ is a sequence of mouse-token interaction events, s is the selected method name, and r is a rating given by the user about the difficulty of naming the method.

The sequence $evts$ contains events for a specific token becoming visible and invisible, either through hovering or clicking. The rating r is obtained by asking the participant after each method to rate how difficult choosing the name was, on a scale from 1 ("easy") to 5 ("hard").

Each participant in our study gets assigned a set of 20 methods that are randomly sampled from the test set of the dataset [5]. Following a common practice in human studies [18], we consider the first three questions as a warm-up, i.e., every participant produces 17 human attention records. The sampling is done such that each method is seen by m different participants, where we set $m = 5$ following related studies [57], [62].

We recruit participants among undergraduate-level and graduate-level university students in computer science, and

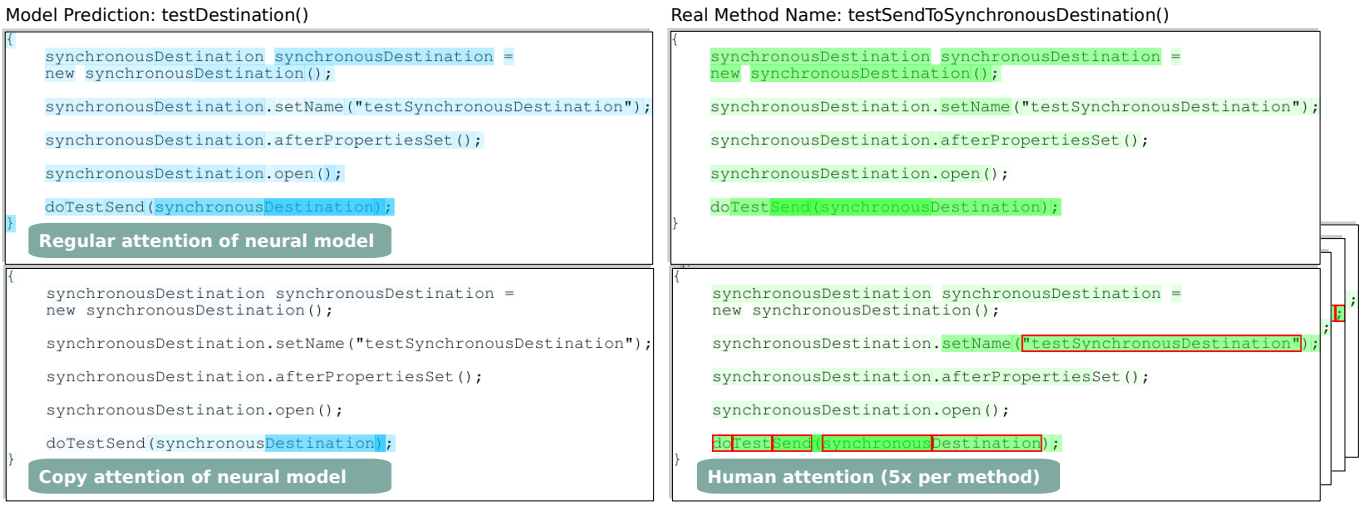

Fig. 2. Visualization of token-level attention weights derived from CNN model (left) and humans (right). Red boxes indicate which token was clicked by the participant.

via Amazon Mechanical Turk (AMT). The AMT participants must have at least 90% approval rate and 100 approved tasks to participate in the study [34] and are rewarded with one US dollar upon successful completion [62]. To filter out participants who misunderstand the task or fail to successfully complete it for other reasons, we measure the correctness of the selected method name on two levels: *exactly correct* when the participant selects the original method name, and *pseudo-correct* when the participant selects either the original name or one of the three similar names.

For each combination of a number of correct and a number of pseudo-correct answers we compute the probability that a participant performing random guessing obtains the same or a better result. Based on this probability, we accept only participants that provide results with a probability of less than 5% to come from a random guesser. Specifically, we consider the multinomial distribution, as we are dealing with an extension of the binomial experiment, with three possible outcomes: A for the participant selecting the exactly correct answer, B for the participant selecting a pseudo-correct answer, and C for the participant selecting a wrong name. For a random guesser, these outcomes have probabilities: $P(A) = \frac{1}{7}$, $P(B) = \frac{3}{7}$, $P(C) = \frac{3}{7}$. Given the number of exactly correct, pseudo-correct, and wrong answers of a single participant as x, y and z respectively, we compute the probability for a randomly acting participant to obtain the same or a better result:

$$
\begin{aligned}
P_{rnd\_guesser}(C = x, N = y, W = z) = \\
P(C = x, N = y, W = z) + \\
\sum_{i=1}^{y} P(C = x + i, N = y - i, W = z) + \\
\sum_{i=1}^{z} P(C = x, N = y + i, W = z - i) + \\
\sum_{i=1}^{z} P(C = x + i, N = y, W = z - i)
\end{aligned}
$$

where the vector of random variables $X = (C, N, W)$ is multinomially distributed with index $n = 17$ and parameters $\pi = (P(A), P(B), P(C)) = (\frac{1}{7}, \frac{3}{7}, \frac{3}{7})$, i.e., $X \sim Mult(n, \pi)$. After filtering, the final dataset contains 1,508 human attention records, which contain at least five human annotations for 250 methods. The records are gathered from those 91 out of 166 participants that pass our filtering. 26 of the accepted participants are computer science students and 65 are recruited via AMT. On average, a human attention record contains about 1,271 mouse-token interaction events.

C. Comparing Attention: Neural Models vs. Humans

The second major challenge is comparing the captured human attention to learned attention weights in neural models. Figure 2 illustrates the problem with an example method, where the model attention is shown on the left, and the human attention is shown on the right. The following describes the neural models we study and how we compare them against the human attention records.

1) Neural Models: We study two attention-based neural models representative of two widely-used architectures: a convolutional attention model [5], called *CNN model*, and a transformer-based model [1], called *transformer model*. Despite the recent popularity of transformers, we choose to study also a CNN model because it is one of the first attention-based models for this task and because it allows us to draw more general conclusions. The CNN model also has the advantage of taking arbitrarily sized inputs, whereas the transformer model imposes a fixed input length.

Both models have a sequence-to-sequence architecture that reasons about the method body as a sequence of tokens and then predicts the tokens of the method name.

We distinguish between two kinds of attention:

- *Regular attention*, which is implemented as convolutional attention for the CNN model and as multi-head self-attention for the transformer model. The regular attention shows which parts of the code the models pay most attention to when reasoning about the meaning of the method.
- *Copy attention*, a mechanism to tackle the out-of-vocabulary problem by optionally copying some tokens from the method body to the output. The copy attention shows which parts of the method body the model considers as candidates for verbatim copying.

For both the CNN and the transformer, we train project-specific models, as they outperform a single cross-project model [5]. We leave the model architecture and all hyper-parameters in their default configurations, except for one adaptation of the transformer model. The original model is designed to summarize a method into a Javadoc sentence. We adapt the model to the method naming task by first pre-training the model on its original dataset [1] and by then fine-tuning the project-specific models on the method naming dataset [5]. Moreover, since the transformer model attends only to the first 150 tokens of a method, all results for this model consider those tokens only.

To measure the effectiveness of the models, we compute the F1-score of the top-most predicted name [5], [6]. It consists of comparing the set of predicted tokens and the tokens in the original name, and then computing the harmonic mean between precision $= \frac{\text{\# correctly predicted tokens}}{\text{\# predicted tokens}}$ and recall $= \frac{\text{\# correctly predicted tokens}}{\text{\# tokens in the original name}}$. The average F1-score of the CNN model and the transformer model are 0.40 and 0.46, respectively, which is in line with the originally reported results.

2) Measuring Human-Model (Dis)Agreement: We summarize the attention spent by humans and models into vectors and then compute correlations between the two. As a proxy of the human attention given to a token, we consider the total time that the token was visible, similarly to fixation time in eye tracking studies [22], [55]:

Definition 2 (Human attention)**.** Let $hr = (id, uid, evts, s, r)$ be a human attention record for a method body with n tokens. The human attention vector is $\vec{h} = (h_1, h_2, ..., h_n)$, where h_i is the total time that the token at position i has been visible to the participant according to $evts$.

For each of the two kinds of attention, we summarize the model's attention as follows:

Definition 3 (Model attention)**.** Suppose a method body with n tokens and a sequence of k tokens predicted by the model as the method name. The machine attention vector is $\vec{m} = (m_1, m_2, ..., m_n)$, where $m_i = mean(a_i^1, ..., a_i^k)$ with a_i^j indicating the attention weight assigned by the model to the token at position i of the method body during the prediction of the jth output token.

Computing the mean handles the fact that the models produce one vector of attention weights for every predicted token in the method name. For example, for a predicted method name `getClientData`, for each token in the method body, we average the attention weights assigned to that token during the prediction of the three tokens "get", "client", and "data".

Given two attention vectors $\vec{h}$ and $\vec{m}$, we compute to what extent they agree on the importance of tokens by computing Spearman's rank coefficient [58]. To this end, we convert the attention vectors to a ranking of tokens, rg_h and rg_m, and then compute Spearman's rank correlation coefficient as the Pearson correlation coefficient between the rank variables: $Spearman = \frac{cov(rg_h, rg_m)}{\sigma_{rg_h}, \sigma_{rg_m}}$ where cov and σ are the standard deviation and the covariance, respectively. The coefficient ranges between -1 and 1, where 1 means that both attention vectors perfectly agree, -1 means that the attention weights yield exactly the inverse ranking, and zero means that both are unrelated. For all reported correlation coefficients, we include only pairs of attention vectors with high-confidence results (p-value ≤ 0.05). A valid alternative measure to Spearman's coefficient is the Kendall Tau [36]; it yields similar results as those reported here and is omitted for space reasons.

IV. Results

Our dataset contains 1,508 human attention records from which we extract the same number of token-level attention vectors, as explained in Definition 2. It comprises 250 methods from ten different repositories, where each method is annotated by at least five of the 91 human annotators. On average our participants took 57 seconds to name a single method, producing 1,271 mouse-token events each time.

A. RQ1: Correlation Between Model and Human Attention

To quantify the overall degree of agreement between the attention paid by neural models of code and humans, we compute the correlation between both. Figure 3 shows the distribution of Spearman rank correlation coefficients for the two kinds of model attention for each of the two models we study. Each plot shows how many of the studied Java methods fall into a specific correlation range. A correlation of 0.0 would mean that a model and the humans are completely unrelated, and a correlation of 1.0 would mean that both agree perfectly on the relatively order of all attention weights. The first and second plot show the results for regular and copy attention of the CNN model, respectively, and likewise with the third and fourth plot for the transformer model. For example, the second plot shows that for the majority of methods, the copy attention weights of the CNN model have a rank correlation between 0.25 and 1.0 with the humans who inspect the same method.

For both models, the regular attention weights show a weak correlation with humans, with means of 0.08 and -0.20. In contrast, the copy attention weights of both models show a moderate to strong human-machine correlation, with means of 0.49 and 0.47. As an example, in Figure 2 both

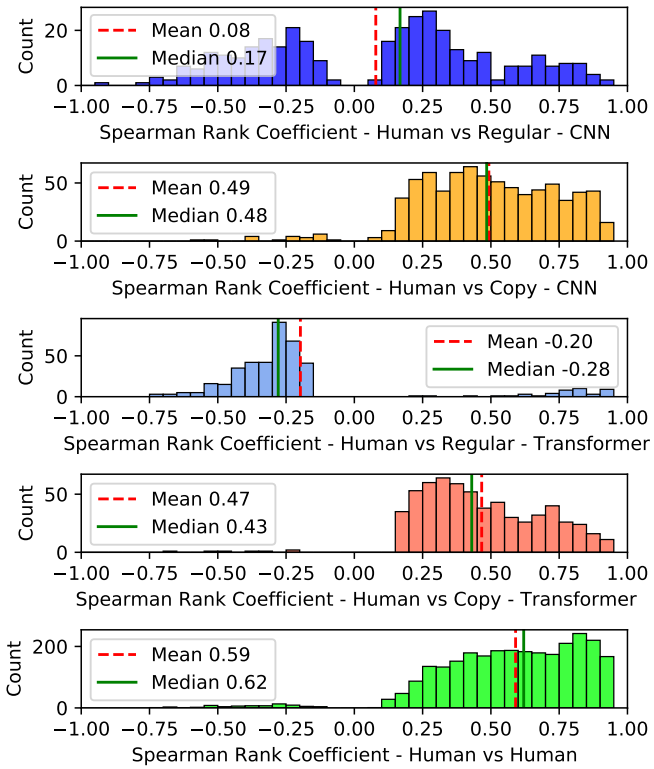

Fig. 3. Correlation between machine attention and human attention.

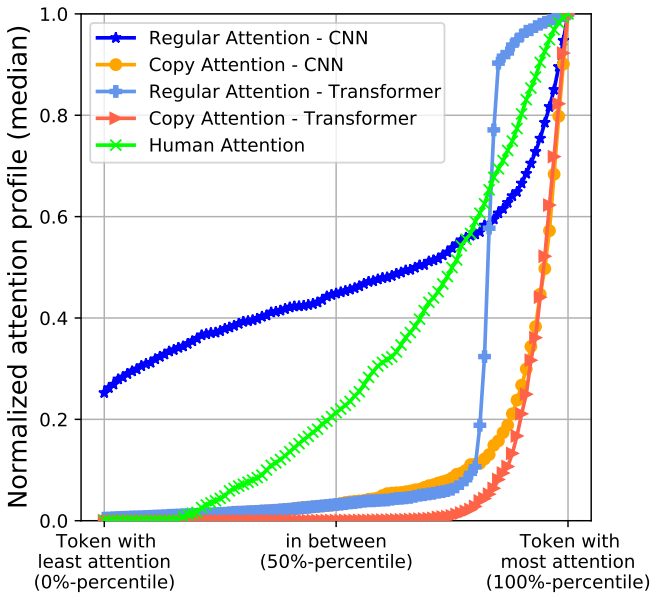

Fig. 4. Median value of normalized attention profiles across all studied methods.

the copy attention of the CNN model (bottom-left) and the human (bottom-right) are paying most of the attention to the last few tokens of the method body. Another interesting observation is that the human shows interest for the string `"testSynchronousDestination"`, which the model seems to overlook. We provide a separate discussion on strings in Section IV-C.

As a point of reference, the last plot in Figure 3 shows the correlation between different humans who inspect the same method. The human-human agreement can be considered an upper bound of the expected model-human agreement, as it would be unrealistic to expect a neural model to be closer to the average human than another human. The mean human-human correlation is 0.59, which shows two points. First, it confirms that different participants in our study tend to attend to similar tokens, which is a prerequisite for comparing models against "humans" as a group. Second, it shows that the mean correlations on copy attention are relatively high, as they are only 0.10–0.12 points lower than the human-human correlation.

Insight 1: Neural models and humans often agree about what tokens to copy verbatim from the input to the output, but less on what other tokens to attend to. The relatively high correlation for copy attention gives an empirical justification to the copy attention mechanisms used by many neural models.

B. RQ2: Distribution of Attention Across Tokens

The following aims at understanding how models and humans distribute their attention across the tokens in a method. Intuitively, we are interested in how much the attention is focused in a few highly relevant tokens, as opposed to being roughly uniformly distributed. To quantify and visualize the different attention distributions, we compute a normalized attention profile from each vector of attention weights:

Definition 4 (Normalized attention profile)**.** Given a vector of attention weights $\vec{a}$, its normalized attention profile is

$$\vec{p} = percentiles(sort(normalize(\vec{a})))$$

where $normalize$ divides all elements by the maximum value in $\vec{a}$, $sort$ sorts the vector in increasing order, and $percentiles$ projects a vector of arbitrary length into a sequence of 100 percentiles (with linear interpolation).

For example, suppose a very small method with five tokens and an attention vector $\vec{a} = [0, 4, 0, 2, 3]$. Normalizing and sorting the attention vector yields $[0, 0, 0.5, 0.75, 1]$, which is then mapped into percentiles to give the a normalized attention profile where, e.g., the 20%-percentile is zero and the 50%-percentile is 0.5.

Figure 4 shows the normalized attention profiles for the five kinds of attention vectors we study: four from the neural models and one from the humans. Each curve is the median value across the attention vectors of all methods in our dataset. Intuitively, the more a curve is dented toward the lower-right corner, the more focused the attention vectors are on a small number of tokens. The results show that copy attention tends to be clearly more focused then regular attention, i.e., the models take a clear decision about which tokens may be worth copying

to the output. In contrast, the regular attention, especially of the CNN model, are more uniformly distributed with respect to their own copy counterparts. Both the regular and copy attention of the transformer disregards a large number of tokens.

The human profile is in between the two transformer attention mechanisms for the top-most attended token (top-right in Figure 4), and then decreases almost constantly until it, perhaps surprisingly, reaches zero. Reaching zero means that the humans often completely ignore some tokens, i.e., our participants could summarize a method based on only a subset of all its tokens. During manual inspection of attention records, we notice that, especially on longer methods, humans often focus on variables declared at the beginning and then skim read to the end of the method, paying most attention to return statements and assertions.

Insight 2: No attention mechanism, among those studied, closely mimics the way humans distributes their attention on the tokens of the method body.

Insight 3: The transformer model seems to be overspecialized in attending only a small subset of tokens, as compared to the convolutional model.

Insight 4: Sometimes the humans do not fully explore the entire method, but base their answers on a subset of the tokens in the methods, typically the beginning (esp. variable declarations) and end (esp. returns and assertions).

Suggestion 1: Future human-inspired transformer models should be trained with an objective to attend to a larger portion of the source code, rather than overspecializing to a few tokens.

C. RQ3: Categories of Tokens

Source code is composed of different categories of tokens, e.g., identifiers, separators, keywords, and strings. For the code in our dataset, the most common token categories are identifiers (46%) and separators (39.7%). To quantify how much attention a specific token category receives, we define the following notion. Intuitively, it indicates how much more or less attention a specific token category receives compared to uniformly distributed attention.

Definition 5 (Distance from uniformity)**.** Given a vector of attention weights $\vec{a}$ for a sequence T of tokens, the distance from uniformity of a subset S of T is:

$$DFU(S) = \frac{\frac{\sum_{t \in S} a_t}{\sum_{t \in T} a_t} - \frac{|S|}{|T|}}{\frac{|S|}{|T|}}$$

where a_t is the attention weight assigned to token t.

A DFU value of zero indicates that the token category is getting exactly the attention of uniformly distributed attention. A positive or negative value indicates that the token category is receiving more or less attention, respectively. The lower bound is -1, where the token category is receiving no attention at all.

Figure 5 presents the DFU for different token categories. The results show significant differences between the models and the humans. For example, the humans give more

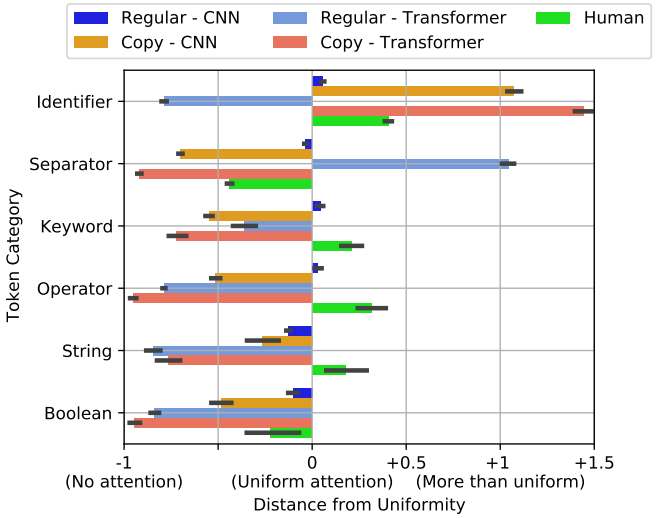

Fig. 5. Distance from uniformity (DFU) for different token categories.

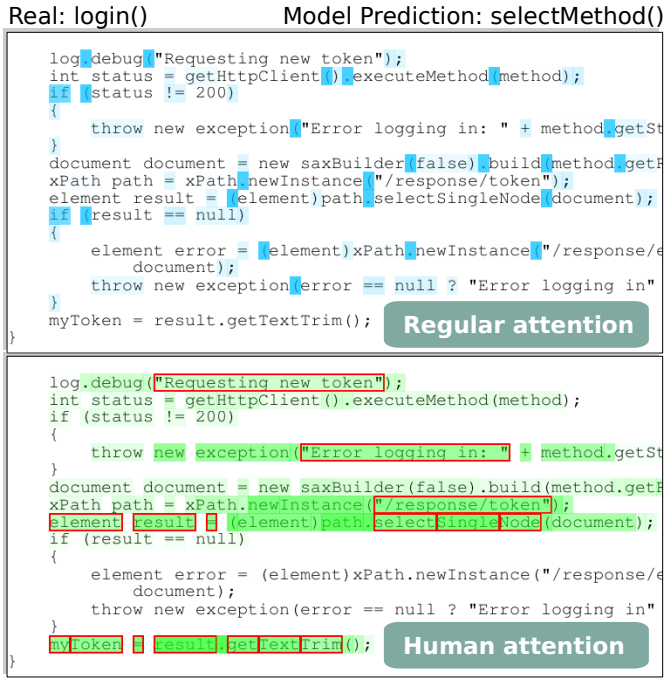

Fig. 6. Attention maps from the transformer model and humans.

importance to keywords, operators, and strings, whereas the models pay less attention to these token categories. This difference suggests that these tokens play an important role in comprehending code, and they should not be left unattended by neural models. One can also observe differences between the different kinds of model attention. For example, the regular attention by the transformer model is surprisingly high for separators and relatively small for identifiers. Some token categories, especially Booleans, are mostly ignored both by the models and the humans.

A manual inspection of various attention maps confirms

our quantitative results. For example, consider the code in Figure 6, which shows the attention maps of the CNN model's regular attention (top) and of the humans who studied this method (bottom). The humans pay a lot more attention to string literals, which indeed contain information relevant for the method summarization task, whereas the model is overlooking their importance in favor of other tokens. The observation illustrated by this example motivates work on neural models that "understand" string literals, which currently are often abstracted away [27] or split with empty spaces [35], when other splits might be more effective (e.g., with slash).

Another finding, is that humans tend to overlook curly braces, which is visible in Figure 6 and also confirmed by the DFU of curly braces being close to its lower limit ($DFU \leq -0.9$). In the upper part of Figure 6 we see how the transformer model prefers syntactic tokens, such as dot, comma, and open parenthesis. After a thorough manual inspection, we confirmed this to be a general characteristic of the regular attention of the transformer model, which focuses on tokens that proceed and follow method calls, in an attempt to isolate method calls. This is explaining also the attention profile of the regular transformer in Figure 4, where the plateau on the top right is made of those tokens (i.e., dots, commas, and open parentheses) used to isolate method calls.

Insight 5: High attention paid by the copy attention to identifiers confirms their effectiveness in focusing primarily on tokens that might be copied verbatim into the method name.

Suggestion 2: The focus of copy attention on identifiers could be stressed even more by masking the other kinds of tokens.

Insight 6: Strings, keywords, and operators are often overlooked by the models, whereas the humans give more attention to them.

Suggestion 3: Future neural models could pay more attention to strings, keywords, and operators, which humans consider important during the method summarization task.

Insight 7: Block-level separators, such as curly braces, are attended mostly by the models, whereas the humans get this information implicitly from the indentation of the formatted code.

Suggestion 4: Future models of code could encode basic syntactic information into token embeddings, preventing the model to focus its attention on purely syntactic tokens, such as curly braces.

Insight 8: Regular attention of transformers specializes in the recognition of separators that proceed and follow method invocations. This confirms and extends an analogous specialization of multi-head attention observed also in natural language processing [10]. We here confirm that transformers are attentive to separators also when applied to code-related tasks. We hypothesize that recognizing separators is important for the model to understand the role of individual tokens, e.g., that a specific token represents the name of a called method. An interesting future work could be to further explore the root causes of this phenomenon.

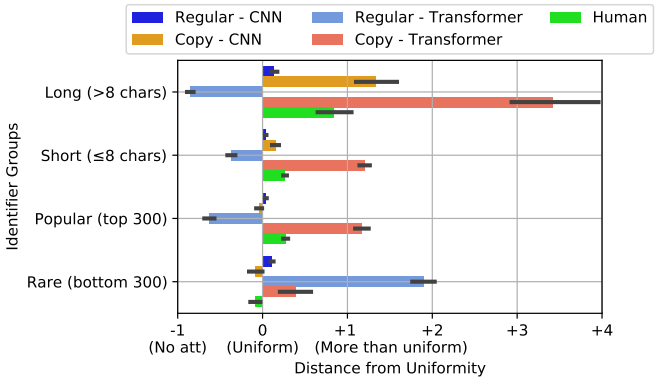

Fig. 7. Distance form uniformity (DFU) for different groups of identifiers.

Given the high attention given by models to identifiers, we further analyze different groups of identifiers based on their length and popularity. Figure 7 shows how copy attention for both models prefers long and popular identifiers over short and rare ones, respectively.

Insight 9: Long and frequent identifiers are good candidates to be copied verbatim into the method name by the analyzed models.

D. RQ4: Perceived Difficulty vs Model Effectiveness

Understanding which examples are more difficult for humans and models could reveal in which measure they are similar and if they can complement each other. We use the rating given by the participant on how easy it was to name each method to create a per-method average rating. We aggregate the ratings of five different annotators on the same method to get a more stable human ground-truth. Comparing these ratings to the F1-score of the models shows a positive correlation (Pearson correlation 0.45 and 0.49 for CNN and transformer, respectively).

Insight 10: The neural models and the humans agree on which methods are more difficult to name.

To investigate further which are those difficult methods, we also compare the performance of humans and models on different method lengths and different groups of methods. Figure 8 shows how both humans and models are good on shorter methods, whereas predicting the name of longer ones is more challenging.

To better understand what kinds of methods models and humans are successful on, we analyze five groups of methods: *getter*, *setter*, *checker*, *test* starting respectively with "get", "set", "is" or "has", and "test", and *other* for all the remaining methods. Figure 9 reports the models effectiveness, measured with F1-score, and the percentage of correctly (and pseudo-correctly) named methods by humans. It shows how getters and setters are the easiest to predict for both humans and models. An interesting finding is that humans are almost as good on checkers as they are on getters, whereas this is not the case for the models, which struggle with identifying the name of a method that is checking some property on the

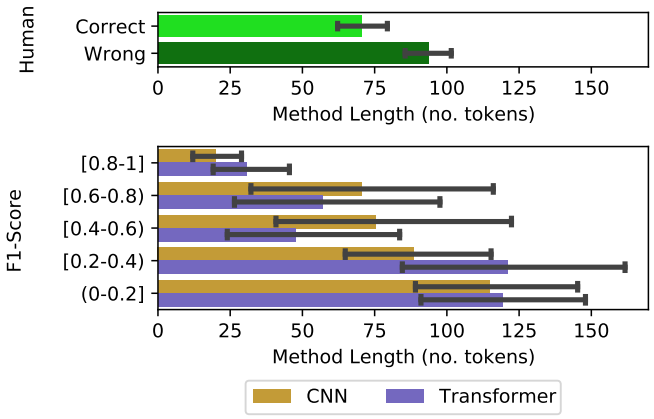

Fig. 8. Method length for different performance levels for human (correct or wrong) and model (f1-score). The error bars measure the uncertainty around the estimate.

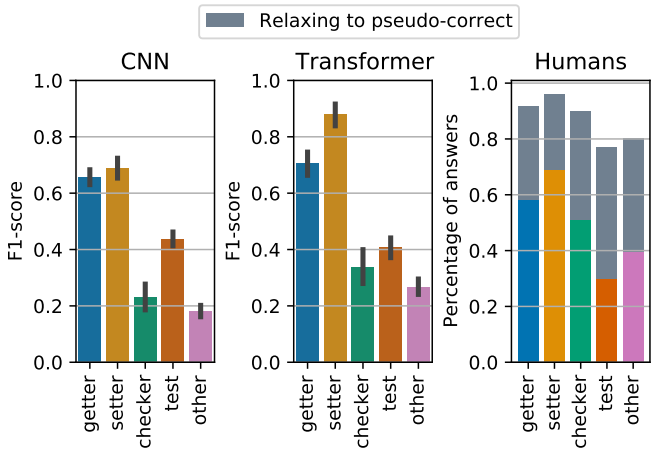

Fig. 9. Model effectiveness and human performance on predefined groups. The error bars measure the uncertainty around the estimate.

object. For test methods and other methods we see a lower percentage of correctly named methods for both humans and models. We note that humans have a high portion of pseudo-correct answers for the test methods, which could be due to the presence of multiple reasonable method names for them.

Insight 11: Beside getters and setters, neural models often struggle to predict more challenging kinds of methods, such as checkers and test methods, whereas the humans are successful across a wider range of methods.

Suggestion 5: Models could learn from the human to name more challenging types of methods on which humans perform better, e.g., by using human attention traces during training.

In Table I, we consider both characteristics, presenting the length of the different groups in terms of lines of code (LOCs). We see that test and others methods, which are hard for models, are generally longer than other groups, showing once again the impact of method length on model effectiveness.

Insight 12: Longer methods are harder to summarize, both for models and humans.

Suggestion 6: To obtain models that better complement human reasoning, future training datasets should include a larger portion of "difficult" examples for a more effective training, or at least provide different sub-datasets of increasing difficulty. To establish the difficulty of a method, we envision the use of heuristics or human labeling. In particular to select more difficult methods, we propose the following strategies: include methods with high cyclomatic complexity [41], reduce the percentage of getter and setter methods, reduce the percentage of methods for which the method body contains many or all subtokens present in the method name (i.e., methods where the model can exploit the copying mechanism), and increase the percentage of longer methods. In addition to these automated strategies for increasing the difficulty, humans could rate a small pool of methods based on their notion of difficulty, which could then serve as a curated benchmark.

TABLE I
METHOD LENGTH OF METHOD BODIES FOR FIVE PREDEFINED GROUPS.

	Lines of Code (LOCs)			
Group	Min	Median	Mean	Max
Getter (20%)	3	3.0	8.6	107
Setter (12%)	3	3.0	4.2	13
Checker (4%)	3	4.0	4.9	11
Test (21%)	1	10.0	14.7	61
Other (43%)	3	8.0	14.8	142
Entire dataset	1	5.0	11.9	142

E. RQ5: Human-Model Agreement vs. Model Effectiveness

Given the moderate to strong correlation between neural models and humans, a reader may wonder whether a stronger correlation coincides with more accurate predictions by the model. We address this question in two ways. One of them is in Figure 10, which shows for each method in our dataset two pieces of information: (i) on the horizontal axis, the accuracy of the model's prediction, measured as the F1-score (Section III-C1) and (ii) on the vertical axis, the human-model agreement, measured as in RQ1. The four plots show the CNN model on top and the transformer model at the bottom, with regular attention and copy attention on the left and right, respectively. Each plot also shows the linear regression trend between two the axes. Overall, we observe a moderate correlation between human-model agreement and model effectiveness. In particular, for the regular attention of both models the Pearson correlation coefficients are 0.19 and 0.40 (p-values $5 \cdot 10^{-4}$ and $2 \cdot 10^{-17}$).

As a second way of addressing the question, we repeat the measurements from RQ1, i.e., how much models and humans agree about what tokens to attend to, for those methods where the models make accurate predictions. "Accurate" here means that the F1 score of a prediction is at least 0.5. Table II compares the human-model correlation across all methods with those methods where the respective model predicts the

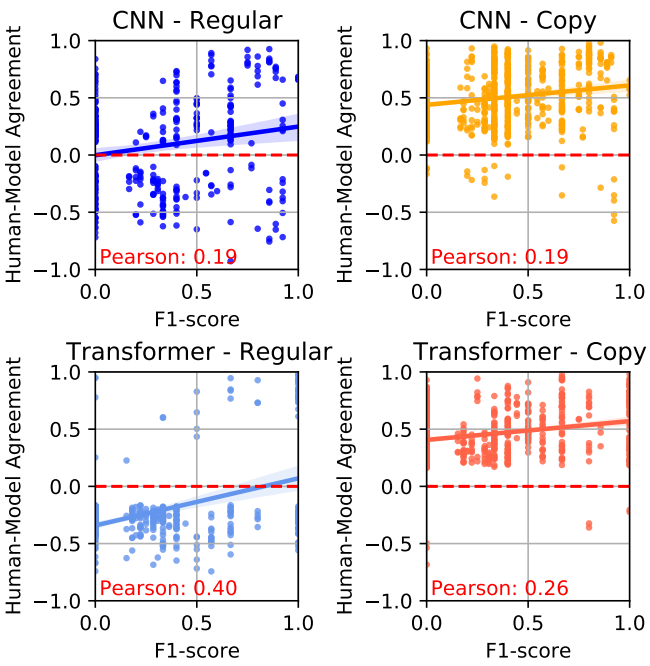

Fig. 10. Human-model correlation vs. model effectiveness.

TABLE II
HUMAN-MODEL CORRELATION FOR ALL VS. ACCURATE PREDICTIONS.

	Spearman rank correlation (mean)	
	All methods	Methods with F1 $\geq$ 0.5
CNN, regular attention	0.08	0.24
CNN, copy attention	0.49	0.55
Transformer, regular attention	-0.20	0.02
Transformer, copy attention	0.47	0.55

name accurately. The results show that the human-model correlation is clearly higher for accurate predictions, e.g., increasing from 0.08 to 0.24 for the regular attention paid by the CNN model.

Insight 13: A higher human-model correlation coincides with more effective predictions by the neural models.

Suggestion 7: Creating models that more closely mimic the human attention seems a promising way toward more effective models. Future work could use human attention datasets during model training [51], [54] or use loss functions that nudge the model's attention to mimic humans.

V. Threats to Validity

1) Internal Validity: Several factors may influence our results. First, the human task of choosing the correct name among seven alternatives differs from the model's task of predicting the entire name. Because both tasks are strongly related and require to understand the meaning of a method, we consider the resulting attention vectors to allow for a meaningful comparison. Moreover, we ensure the human task to be challenging by providing alternatives similar to the correct name (Section III-A). Second, using a crowd platform to hire participants risks getting submissions of mixed quality. We carefully filter all human attention records based on the overall performance of a participant (Section III-B). A manual inspection of the dataset confirms that realistic code explorations are retained by the filtering. Third, computing human attention based on the time a token is visible can only approximate actual attention. In computer vision, mouse tracking has been established as a scalable way of capturing visual attention [14], [33], and an in-depth study could assess the accuracy of it on source code-related tasks in the future. Fourth, our HRR interface introduces some trade-offs compared to the eye tracking-based studies: On the one hand, we lose the pixel-level precision of an eye tracker and are not able to capture if a participant indeed looks at the unblurred code. On the other hand, HRR enables easier remote participation, supports code snippets of arbitrary length, and automatically captures token-level attention. A further study of strengths and weaknesses of the two approaches for software engineering studies will be interesting future work. Fifth, the choice of the neighborhood size of unblurred tokens, which is three tokens before and three tokens after the clicked token in our experiments, could lead to different results. Sixth, truncating the input to transformers to the first 150 tokens may influence our results since the model cannot look beyond that limit. Finally, method length and code style may confound our findings. We partially study the influence of method length in Figure 8, but do not consider code style as it is difficult to quantify. To mitigate the impact of both possible confounders, we randomly sample from multiple repositories methods with different lengths and code styles.

2) Threats to External Validity: Several factors may influence the generalizability of our results. First, the participants of our study may not fully represent other humans. We mitigate this threat by recruiting participants through different ways and by retaining only participants that show high performance. Second, findings on the code summarization task might not generalize to other code-related prediction tasks. As the task has been proved to be a good benchmark for the abstraction abilities of models of code [39], we envision our findings to at least partially generalize to other tasks. By making our Human Reasoning Recorder available, we facilitate future work to study different tasks. Third, we select a CNN-based and a transformer-based model with token-level attention as subjects for the study. Other models, e.g., those based on AST paths [6], [7], may expose other attention patterns. Our dataset of human attention records is available for comparisons with other models.

VI. Related Work

A. Capturing Human Attention

Capturing the human reasoning has always been a challenging and demanding task, as witnessed by previous studies on computer vision [14], [50] and natural language processing [9], [15], [38], [49], [57]. Indirect experiments, such as ours, have been also used in these fields [14], [57]. In software engineering, eye tracking [55] has been the basis of

code comprehension studies [12], [20]. With 432 human eye tracking records, the dataset by Bednarik et al. [12], was the largest dataset on human attention on code available so far. Two tools have been proposed to ease the collection of eye data [23], [37]. However, due to the equipment and calibration requirements, eye tracking is not easily compatible with remote participation. Neuroimaging is another interesting, yet also very involved way to measure the activity of programmers during program comprehension tasks [45], [56]. Our work contributes a deblurring-based interface for capturing human attention on code, which complements existing techniques by providing a lightweight and scalable technique that is compatible with remote participation.

B. Comparing Models with Developers

Previous work [52] studies which parts of a method are of interest for a group of ten Java developers performing a code summarization task, and compares their eye tracking data against a tf-idf method [25]. They also show that eye tracking data from programmers can improve the tf-idf model. A first attempt to compare neural models of code against human attention captured via eye tracking [30] compares the gaze of single human participant against the Code2vec model [7]. Our work contributes the first in-depth, multi-participant study to compare neural models of code with human attention.

C. Studies of Attention Mechanisms

The role of attention layers as an explanation technique is still under active study, e.g., by measuring the effectiveness of attention layers against other explanation techniques [32], [66]. Instead of comparing multiple explanation techniques with each other, our work compares a model against human attention records. Bui et al. study attention in neural models of code by comparing attention weights with a metric based on the perturbation of the input program [13]. Their paper mentions that "evaluations with real programmers can be more convincing in validating whether [their] results match the actual importance viewed by human", which matches the motivation for our work. Trying to understand the attention of transformer models, some work highlights how the various attention heads of a transformer are redundant and that some attention heads specialize in attending syntactic tokens, such as punctuation [10]. A similar observation is that much of the attention of transformer-based models is assigned to punctuation tokens [53]. Arous et al. show that integrating human rationales into an attention-based model for NLP can improve its effectiveness [8]. The attention records gathered in our work could serve as a basis of similar future work on models of code.

D. Neural Models of Code

There are various neural models of code, and we refer to a recent review article [46] and a survey [2] for a detailed discussion. Many models consider source code as a sequence of tokens [24], [26], [59], [67], which also is the representation underlying our study. Some tree-based models [6] and graph-based models [17] also adopt an attention mechanism, which would be interesting to compare against our human attention records.

VII. Conclusion

Motivated by the success of neural models of code, combined with the difficulties to understand what exactly the models are learning, this paper presents the first comprehensive study to compare human and model attention. The results show how the copy mechanism is empirically very similar to the human attention. Moreover, we have pointed out important differences between models and humans, e.g., the different attention weights given to basic syntactic tokens, such as curly braces, but also a tendency of neural models to underestimate the value of strings. We reveal that neural models generally struggle on longer methods, and on methods beyond getters and setters, whereas the humans successfully understand a wider range of methods. Our work also highlights that human-model agreement positively correlates with accurate predictions, which calls for neural models that even more closely mimic human reasoning. Together with the study, we release a novel dataset of 1,508 human attention maps on the code summarization task, collected via the newly proposed Human Reasoning Recorder, which could serve as an enabler of further research and human studies. Ultimately, we envision the usage of our dataset and our tool to produce ground-truth human annotations to fuel human-inspired neural models.

Acknowledgment

This work was supported by the European Research Council (ERC, grant agreement 851895), and by the German Research Foundation within the ConcSys and Perf4JS projects. We also thank the anonymous reviewers for their insightful comments and suggestions, which have contributed to improve the quality of this work.

References

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A Transformer-based Approach for Source Code Summarization. *arXiv:2005.00653 [cs, stat]*, May 2020.

[2] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):81, 2018.

[3] Miltiadis Allamanis, Earl T. Barr, Soline Ducousso, and Zheng Gao. Typilus: Neural type hints. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, pages 91–105, New York, NY, USA, June 2020. Association for Computing Machinery.

[4] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to Represent Programs with Graphs. *arXiv:1711.00740 [cs]*, May 2018.

[5] Miltiadis Allamanis, Hao Peng, and Charles Sutton. A Convolutional Attention Network for Extreme Summarization of Source Code. In *International Conference on Machine Learning (ICML)*, May 2016.

[6] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. Code2seq: Generating Sequences from Structured Representations of Code. *International Conference on Learning Representations*, February 2019.

[7] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. Code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL):40:1–40:29, January 2019.

[8] Ines Arous, L. Dolamic, Jie Yang, Akansha Bhardwaj, Giuseppe Cuccu, and P. Cudré-Mauroux. MARTA: Leveraging Human Rationales for Explainable Text Classification. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2021.

[9] Pepa Atanasova, Jakob Grue Simonsen, Christina Lioma, and Isabelle Augenstein. A Diagnostic Study of Explainability Techniques for Text Classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3256–3274, Online, November 2020. Association for Computational Linguistics.

[10] Joris Baan, Maartje ter Hoeve, Marlies van der Wees, Anne Schuth, and Maarten de Rijke. Understanding Multi-Head Attention in Abstractive Summarization. *arXiv:1911.03898 [cs]*, November 2019.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016.

[12] Roman Bednarik, Teresa Busjahn, Agostino Gibaldi, Alireza Ahadi, Maria Bielikova, Martha Crosby, Kai Essig, Fabian Fagerholm, Ahmad Jbara, Raymond Lister, Pavel Orlov, James Paterson, Bonita Sharif, Teemu Sirkiä, Jan Stelovsky, Jozef Tvarozek, Hana Vrzakova, and Ian van der Linde. EMIP: The eye movements in programming dataset. *Science of Computer Programming*, 198:102520, October 2020.

[13] N. D. Q. Bui, Y. Yu, and L. Jiang. AutoFocus: Interpreting Attention-Based Neural Networks by Code Perturbation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 38–41, November 2019.

[14] Abhishek Das, Harsh Agrawal, Larry Zitnick, Devi Parikh, and Dhruv Batra. Human Attention in Visual Question Answering: Do Humans and Deep Networks Look at the Same Regions? *Computer Vision and Image Understanding*, 163:90–100, October 2017.

[15] Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C. Wallace. ERASER: A Benchmark to Evaluate Rationalized NLP Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4443–4458, Online, July 2020. Association for Computational Linguistics.

[16] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. Hoppity: Learning Graph Transformations to Detect and Fix Bugs in Programs. In *International Conference on Learning Representations*, September 2019.

[17] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. Hoppity: Learning graph transformations to detect and fix bugs in programs. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[18] Janet Feigenspan, Christian Kästner, Sven Apel, Jörg Liebig, Michael Schulze, Raimund Dachselt, Maria Papendieck, Thomas Leich, and Gunter Saake. Do background colors improve program comprehension in the #ifdef hell? *Empirical Software Engineering*, 18(4):699–745, August 2013.

[19] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, November 2020. Association for Computational Linguistics.

[20] Thomas Fritz, Andrew Begel, Sebastian C. Müller, Serap Yigit-Elliott, and Manuela Züger. Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 402–413, New York, NY, USA, May 2014. Association for Computing Machinery.

[21] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. Bottom-Up Abstractive Summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[22] D. T. Guarnera. Enhancing Eye Tracking of Source Code: A Specialized Fixation Filter for Source Code. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 615–618, September 2019.

[23] Drew T. Guarnera, Corey A. Bryant, Ashwin Mishra, Jonathan I. Maletic, and Bonita Sharif. iTrace: Eye tracking infrastructure for development environments. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, ETRA '18, pages 1–3, New York, NY, USA, June 2018. Association for Computing Machinery.

[24] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish K. Shevade. Deepfix: Fixing common C language errors by deep learning. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1345–1351. AAAI Press, 2017.

[25] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus. On the Use of Automated Text Summarization Techniques for Summarizing Source Code. In *2010 17th Working Conference on Reverse Engineering*, pages 35–44, October 2010.

[26] Vincent J. Hellendoorn, Christian Bird, Earl T. Barr, and Miltiadis Allamanis. Deep learning type inference. In Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu, editors, *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, pages 152–162. ACM, 2018.

[27] Vincent J. Hellendoorn and Premkumar Devanbu. Are deep neural networks the best choice for modeling source code? In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 763–773, New York, NY, USA, August 2017. Association for Computing Machinery.

[28] Vincent J. Hellendoorn, Charles Sutton, Rishabh Singh, Petros Maniatis, and David Bieber. Global relational models of source code. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[29] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *arXiv:1909.09436 [cs, stat]*, June 2020.

[30] T. D. Itoh, T. Kubo, K. Ikeda, Y. Maruno, Y. Ikutani, H. Hata, K. Matsumoto, and K. Ikeda. Towards Generation of Visual Attention Map for Source Code. In *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 951–954, November 2019.

[31] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing Source Code using a Neural Attention Model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany, August 2016. Association for Computational Linguistics.

[32] Sarthak Jain and Byron C. Wallace. Attention is not Explanation. *arXiv:1902.10186 [cs]*, May 2019.

[33] M. Jiang, S. Huang, J. Duan, and Q. Zhao. SALICON: Saliency in Context. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1072–1080, June 2015.

[34] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy. An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.

[35] Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, and Andrea Janes. Big Code != Big Vocabulary: Open-Vocabulary Models for Source Code. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1073–1085, October 2020.

[36] M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.

[37] Chris Lankford. Gazetracker: Software designed to facilitate eye movement analysis. In *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ETRA '00, pages 51–55, New York, NY, USA, November 2000. Association for Computing Machinery.

[38] Piyawat Lertvittayakumjorn and Francesca Toni. Human-grounded Evaluations of Explanation Methods for Text Classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5195–5205, Hong Kong, China, November 2019. Association for Computational Linguistics.

[39] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. *arXiv:2102.04664 [cs]*, March 2021.

[40] Rabee Sohail Malik, Jibesh Patra, and Michael Pradel. NL2Type: Inferring JavaScript function types from natural language information. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 304–315, 2019.

[41] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.

[42] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956.

[43] Kyosuke Nishida, Itsumi Saito, Kosuke Nishida, Kazutoshi Shinoda, Atsushi Otsuka, Hisako Asano, and Junji Tomita. Multi-style Generative Reading Comprehension. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2273–2284, Florence, Italy, July 2019. Association for Computational Linguistics.

[44] Jibesh Patra and Michael Pradel. Semantic bug seeding: a learning-based approach for creating realistic bugs. In Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta, editors, *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, pages 906–918. ACM, 2021.

[45] Norman Peitek, Janet Siegmund, Sven Apel, Christian Kstner, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and Andr Brechmann. A look into programmers heads. *IEEE Transactions on Software Engineering*, 46(4):442–462, 2020.

[46] Michael Pradel and Satish Chandra. Neural software analysis. *Communications of the ACM*, 2021. To appear.

[47] Michael Pradel, Georgios Gousios, Jason Liu, and Satish Chandra. TypeWriter: Neural type prediction with search-based validation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 209–220, New York, NY, USA, November 2020. Association for Computing Machinery.

[48] Michael Pradel and Koushik Sen. DeepBugs: A learning approach to name-based bug detection. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):147:1–147:25, October 2018.

[49] Grusha Prasad, Yixin Nie, Mohit Bansal, Robin Jia, Douwe Kiela, and Adina Williams. To what extent do human explanations of model behavior align with actual model behavior? *arXiv:2012.13354 [cs]*, December 2020.

[50] Arijit Ray, Yi Yao, Rakesh Kumar, Ajay Divakaran, and Giedrius Burachas. Can You Explain That? Lucid Explanations Help Human-AI Collaborative Image Retrieval. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, 7(1):153–161, October 2019.

[51] Laura Rieger, Chandan Singh, William Murdoch, and Bin Yu. Interpretations are Useful: Penalizing Explanations to Align Neural Networks with Prior Knowledge. In *International Conference on Machine Learning*, pages 8116–8126. PMLR, November 2020.

[52] P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan. An Eye-Tracking Study of Java Programmers and Application to Source Code Summarization. *IEEE Transactions on Software Engineering*, 41(11):1038–1054, November 2015.

[53] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A Primer in BERTology: What We Know About How BERT Works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020.

[54] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 2662–2670, Melbourne, Australia, August 2017. International Joint Conferences on Artificial Intelligence Organization.

[55] Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25(5):3128–3174, September 2020.

[56] Janet Siegmund, Norman Peitek, André Brechmann, Chris Parnin, and Sven Apel. Studying programming in the neuroage: just a crazy idea? *Communications of the ACM*, 63(6):30–34, 2020.

[57] Ekta Sood, Simon Tannert, Diego Frassinelli, Andreas Bulling, and Ngoc Thang Vu. Interpreting Attention Models with Human Visual Attention in Machine Reading Comprehension. In *ACL SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, October 2020.

[58] C. Spearman. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, 100(3/4):441–471, 1987.

[59] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. On learning meaningful code changes via neural machine translation. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 25–36, Montreal, Quebec, Canada, May 2019. IEEE Press.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. *Advances in Neural Information Processing Systems*, 30:5998–6008, 2017.

[61] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. *Advances in Neural Information Processing Systems*, 28, 2015.

[62] Yaza Wainakh, Moiz Rauf, and Michael Pradel. Idbench: Evaluating semantic representations of identifier names in source code. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 562–573. IEEE, 2021.

[63] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S. Yu. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 397–407, New York, NY, USA, September 2018. Association for Computing Machinery.

[64] Yu Wang, Ke Wang, Fengjuan Gao, and Linzhang Wang. Learning semantic program embeddings with graph interval neural network. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):137:1–137:27, November 2020.

[65] Jiayi Wei, Maruth Goyal, Greg Durrett, and Isil Dillig. LambdaNet: Probabilistic Type Inference using Graph Neural Networks. In *International Conference on Learning Representations*, September 2019.

[66] Sarah Wiegreffe and Yuval Pinter. Attention is not not Explanation. *arXiv:1908.04626 [cs]*, September 2019.

[67] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Yanjun Pu, and Xudong Liu. Learning to handle exceptions. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020.