Do Machines and Humans Focus on Similar Code? Exploring Explainability of Large Language Models in Code Summarization

Jiliang Li
Vanderbilt University
Nashville, Tennessee, USA
jiliang.li@vanderbilt.edu

Yifan Zhang
Vanderbilt University
Nashville, Tennessee, USA
yifan.zhang.2@vanderbilt.edu

Zachary Karas
Vanderbilt University
Nashville, Tennessee, USA
z.karas@vanderbilt.edu

Collin McMillan
University of Notre Dame
Notre Dame, Indiana, USA
cmc@nd.edu

Kevin Leach
Vanderbilt University
Nashville, Tennessee, USA
kevin.leach@vanderbilt.edu

Yu Huang
Vanderbilt University
Nashville, Tennessee, USA
yu.huang@vanderbilt.edu

ABSTRACT

Recent language models have demonstrated proficiency in summarizing source code. However, as in many other domains of machine learning, language models of code lack sufficient explainability — informally, we lack a formulaic or intuitive understanding of what and how models learn from code. Explainability of language models can be partially provided if, as the models learn to produce higher-quality code summaries, they also align in deeming the same code parts important as those identified by human programmers. In this paper, we report negative results from our investigation of explainability of language models in code summarization through the lens of human comprehension. We measure human focus on code using eye-tracking metrics such as fixation counts and duration in code summarization tasks. To approximate language model focus, we employ a state-of-the-art model-agnostic, black-box, perturbation-based approach, SHAP (SHapley Additive exPlanations), to identify which code tokens influence that generation of summaries. Using these settings, we find no statistically significant relationship between language models' focus and human programmers' attention. Furthermore, alignment between model and human foci in this setting does not seem to dictate the quality of the LLM-generated summaries. Our study highlights an inability to align human focus with SHAP-based model focus measures. This result calls for future investigation of multiple open questions for explainable language models for code summarization and software engineering tasks in general, including the training mechanisms of language models for code, whether there is an alignment between human and model attention on code, whether human attention can improve the development of language models, and what other model focus measures are appropriate for improving explainability.

CCS CONCEPTS

• **Computing methodologies → Artificial intelligence**.

KEYWORDS

Neural Code Summarization, Language Models, Explainable AI, SHAP, Human Attention, Eye-Tracking

ACM Reference Format:
Jiliang Li, Yifan Zhang, Zachary Karas, Collin McMillan, Kevin Leach, and Yu Huang. 2024. Do Machines and Humans Focus on Similar Code? Exploring Explainability of Large Language Models in Code Summarization. In *32nd IEEE/ACM International Conference on Program Comprehension (ICPC '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3643916.3644434

1 INTRODUCTION

Recent language models for code have shown promising performance on several code-related tasks [31]. Among these tasks is neural code summarization, where a language model generates a short natural language summary describing a given code snippet. This is often an indicative task demonstrating a model's ability to comprehend code. Currently, the majority of assessments for how well a language model understands code directly measures the quality of code summaries generated by the models, and compares them with human-written summaries [31]. Comparatively little is known about why and how the language models reason about code to generate such summaries. Similar to many other downstream domains of machine learning in software engineering, understanding and explaining how and why language models for code work (or fail) is critical to improving model architecture, reducing bias, and preventing undesirable model behavior.

Human programmers typically achieve a strong understanding of code. Thus, proficient language models might be explained if they focus on the same parts of code that humans would [21]. Eye-tracking studies have been conducted to analyze programmers' visual patterns while reading code [2, 22]. Specifically, the duration and frequency of a programmer's eye gaze on a part of code in a spatially-stable manner, referred to as *fixation duration* and *fixation count* respectively, are indicative of cognitive load [24]. Thus, these measures of eye-tracking can indicate the parts of code on which human programmers focus. In contrast, there is a lack of consensus on how to measure a language model's reasoning about code (see Section 2.2). Most existing works extract the self-attention layers in language models for code to measure the model attention [21, 20, 9].

Such methods require direct access to the internal layers of a language model, limiting the possibility to investigate interpretability of many state-of-the-art proprietary models (e.g., ChatGPT).

In this paper, to investigate how proprietary language models reason about code, we employ a state-of-the-art perturbation-based method, SHAP [15] (SHapley Additive exPlanations), that treats each language model as a black-box function. With SHAP, we analyze the feature attribution (i.e., which parts of code are deemed important by the model) in six different state-of-the-art language models for code. We use a set of Java methods to task both the language models and human programmers with writing code summaries. The feature attribution in the language models, measured by SHAP, is then compared with human developers' focus, collected from eye-tracking. We hypothesize that sufficiently large models may learn to focus on parts of code similarly to humans. If validated, language model behavior can thus be described in terms of human behavior, ultimately helping to explain and improve language models. However, we find that explainability cannot be provided through this lens and find no statistically significant evidence suggesting the hypothesized alignment. Furthermore, we did not find that language models' focus exhibits a statistically significant correlation with human focus in general. For future research that aims to explore the explainability of language models for code summarization, especially for those leveraging human attention, our findings might suggest the following: (1) though widely used in AI, SHAP may not be an optimal method to investigate where language models focus during code summarization, or alternatively, (2) a misalignment between language models and human developers in reasoning about code may provide insights for improving AI models for code summarization.

2 BACKGROUND AND RELATED WORK

2.1 Neural Models for Code Summarization

Advancements in deep learning have enabled machine learning models to generate summaries for source code. Among the state-of-the-art models, NeuralCodeSum (NCS) first introduced the use of Transformers in neural code summarization [3]. With the rise of large language models (LLMs), ServiceNow and HuggingFace released a 15.5B parameter LLM for code, StarCoder [13], and Meta released a 7B parameter LLM, Code LLama [23], both of which can serve to summarize code. Although not inherently an LLM for code, GPT3.5 [18] and GPT4 [19] are also capable of code summarization. In this paper, we investigate how all the aforementioned models reason about code when tasked to generate code summaries.

2.2 Interpretability of Language Models

Existing works on interpretable language models generally seek to investigate the relative importance of each input token for model performance [29, 8, 16]. Such works can be commonly categorized into two types: white-box vs. black-box. White-box approaches require access to a language model's internal layers [25, 28], often directly investigating the self-attention scores in Transformer-based models [7, 33, 32]. However, Transformer-based models' inherent complexity has led to a lack of consensus on how to aggregate attention weights [30, 35, 33]. For the general research community, white-box approaches preclude proprietary models (e.g., ChatGPT).

In contrast, state-of-the-art black-box approaches like SHAP [15] (SHapley Additive exPlanations) apply game-theoretic principles to assess the impact of input variations on a model's output. SHAP evaluates the effects of different combinations of input features — such as tokens in a text sequence — by observing how their presence or absence (simulated by token masking) alters the model's prediction from an expected result. This process helps to ascertain the relative contribution of each feature to the output, allowing for an analysis of the model without requiring access to its internal architecture [14, 11]. In this paper, to investigate proprietary models, we employ SHAP to measure where language models focus on code.

2.3 Comparing Human vs. Machine Attention

Previous papers have examined the alignment between human and model attention in code comprehension tasks. Paltenghi et al. [20] found that CodeGen's [17] self-attention layers attend to similar parts of code compared to human programmers' visual fixations when answering comprehension questions about code. Similarly, Huber et al. [9] discovered overlaps in attention patterns between neural models and humans when repairing buggy programs. Notably, Paltenghi and Prasdel [21] compared language models' self-attention weights and humans' visual attention during code summarization. They found that model attention, measured by self-attention weights, does not align well with human attention. However, this work is limited by investigating only small CNN and transformer models. Most importantly, all aforementioned studies used white-box approaches towards interpretability of open-source models, limiting applicability to state-of-the-art proprietary models.

Recently, Kou et al. [11] utilized both white-box and black-box perturbation-based approaches to measure LLMs' focus in code generation tasks, and discovered a consistent misalignment with humans' attention. In general, these works have demonstrated that whether human and machine attention align depends heavily on the methods employed to approximate machine focus, as well as the specific code comprehension task examined. In this paper, we build upon former works by examining whether human attention correlates with feature attribution in language models, measured by a black-box perturbation-based approach, in code summarization.

3 EXPERIMENTAL DESIGN

3.1 Measuring Human Visual Focus

We used eye-tracking data measuring human attention from a controlled human study with 27 programmers. The study obtained IRB approval, and asked participants to read Java methods and write accompanying summaries [4]. Each participant summarized 24–25 Java methods from the FunCom dataset [12], yielding 671 trials of eye-tracking data in total. Considering data quality, two authors with five and eight years of Java experience cooperatively removed participant data associated with five summaries that did not demonstrate an understanding of the Java code.

In this work, we sought to measure where humans and language models focus on code as they summarize it. We first used the `srcML` parser to convert each Java method into its corresponding Abstract Syntax Tree (AST) representation [6]. The AST provides structural context for each token literal (i.e., 'Hello World' → String Literal). With the gaze coordinates collected from the eye-tracker [1], we measured humans' focus on each AST token. Typically, researchers use *fixations* to quantify human visual focus [24]. A fixation is defined as a spatially stable eye-movement lasting 100–300ms. Most cognitive processing occurs during fixations [24], so researchers consider their frequency and duration in making inferences about human cognition. In our analyses, we computed the average count and duration of programmers' fixations on each AST token. Consequently, for each Java method, we obtained two visual focus vectors with lengths equal to the number of AST tokens, respectively, which represent fixation counts and durations on each token[1].

3.2 Measuring Model Focus

As mentioned in Section 2.2, we choose SHAP's official, default implementation of the TeacherForcing method to measure feature attribution in language models, treating each as a black-box function. For each language model, we pass in each of the 68 Java methods (also read by human programmers) as input, along with necessary prompting for the model to output summaries of source code. For each Java method passed into each language model, we let i denote an input token (in code) and o denote an output token (in summary). For each (i, o) pair, SHAP produces an importance score, denoted $v_{(i,o)}$, signifying how much i's presence or absence alters the presence of o. Then, the importance score of each input token[2], v_i, is calculated such that $v_i = \sum_o |v_{(i,o)}|$. Note that now v_i is associated with a language model token, and each AST token may consist of several language model tokens. Thus, for each AST token, we calculate its score $\frac{1}{n}\sum_{j=1}^{n} v_j$, where $v_1, \cdots, v_n$ are scores of language model tokens constituting the AST token. Consequently, for each language model on each Java method, we obtain a focus vector (with a length equal to the number of AST tokens) representing how influential each AST token is to the model.

In total, we investigated the model focus of six different models: GPT4, GPT-few-shot, GPT3.5, StarCoder, Code Llama, and NCS. Here, GPT-few-shot is a GPT3.5 model, but in an attempt for the model to produce code summaries more similar to those of humans, we used few-shot prompting to instruct the model to provide summaries similar to two randomly selected human-written summaries. The other five state-of-the-art LLMs are introduced in Section 2.1 and implemented with their default parameters.

3.3 Comparing Human and Model Foci

For brevity, we refer to two human visual focus measurements (i.e., fixation duration and count) and six language models as eight "focus sources." For each source, we obtained 68 focus vectors, each corresponding to a Java method. These vectors were normalized to sum to 1, and reflect how important each AST token is for the human/model. We answer these research questions:

- RQ1: Is there a general correlation between human and machine focus patterns for code summarization?
- RQ2: Do the code summaries increase in quality when machine focus becomes more aligned with that of humans?

3.3.1 RQ1. We assess the correlation between human and machine foci across the 68 Java methods. Specifically, for each pair of focus sources, we iterate through each Java method and calculate the Spearman's rank coefficient (ρ) [27] between the two sources' vectors for that method. Then, for each pair of focus sources, we report: (1) The mean and standard deviation of Spearman's ρ across all Java methods where correlation is statistically significant ($p \le 0.05$), and (2) the proportion of Java methods demonstrating a statistically significant correlation ($p \le 0.05$).

In addition, we group all AST tokens into 18 semantic categories (e.g., method call, operator, etc.) and investigate how much humans[3] and language models focus on each semantic category. The focus score assigned to each semantic category is the sum of the focus scores assigned to each AST token belonging to that semantic category. To counter biases where certain semantic categories contain more AST tokens or appear more frequently, we report the relative difference between machine and human foci for each semantic category. That is, we average the six language models' focus scores per category and report $|\frac{foc_{machine} - foc_{human}}{foc_{human}}|$.

3.3.2 RQ2. Here, a human expert provides quality ratings for summaries generated by each language model for every Java method using four criteria: accuracy, completeness, conciseness, and readability. Next, we calculate the Spearman's ρ between each language model's focus vector and the human fixation duration vector across all Java methods where correlation is significant ($p \le 0.05$). We then append all such statistically significant ρ's to form a vector, denoted v_{cor}, to represent the degrees of alignment between machine and human foci across the Java methods investigated[4].

Subsequently, we determine whether this alignment is correlated with the rated quality of summaries. Specifically, we construct four other vectors, $\{v_{acc}, v_{com}, v_{con}, v_{rea}\}$, containing the accuracy, completeness, conciseness, and readability scores respectively. At each index i, $\{v_{cor}[i], v_{acc}[i], v_{com}[i], v_{con}[i], v_{rea}[i]\}$ are respectively the Spearman's ρ, summary accuracy, completeness, conciseness, and readability of the same language model applied on the same Java method. We then measure and report the Spearman's rank correlation between v_{cor} and v_i, where $v_i \in \{v_{acc}, v_{com}, v_{con}, v_{rea}\}$.

4 RESULTS

4.1 RQ1: General Correlation

As shown in Table 1, there is a general lack of correlation between human and machine foci. We highlight that the means and standard deviations in Table 1 are only calculated from Java methods where the Spearman's ρ is statistically significant (with $p \le 0.05$). In practice, between any pair of human-LLM focus sources, at most 22% of the 68 Java methods yield a Spearman's ρ with $p \le 0.05$. As a baseline, the Spearman's ρ has $p \le 0.05$ for all Java methods between human duration and fixation focus vectors, and for 85% of Java methods between any two language model's focus vectors. This implies that any existing correlation between human and machine foci is not widespread across the Java methods studied.

Furthermore, among those Java methods where the correlation is statistically significant, the mean Spearman's ρ is small with a large

[1]Our analyses do not include brackets or semi-colons, or other such syntactic elements.
[2]We use the absolute value by choice, without which experiments show similar results.

[3]We use fixation durations to represent human focus. We empirically verify that using fixation count yields similar results.
[4]Note that v_{cor} contains Spearman's ρ's obtained from all six language models. We empirically verify that conducting the analogous analysis for each language model separately yields a similar result.

Table 1: Pair-wise correlation among focus sources; "Duration" and "Count" are human visual focus. Each cell shows the means and standard deviations of Spearman's ρ for all Java methods showing significant correlation ($p \leq 0.05$).

	Duration	Count	GPT4	GPT-few	GPT3.5	StarCoder	CodeLlama	NCS
Duration	1.00±0.00	0.88±0.06	-0.11±0.41	-0.13±0.42	-0.09±0.52	-0.18±0.48	-0.18±0.42	-0.24±0.40
Count	—	1.00±0.00	0.01±0.45	-0.24±0.33	-0.10±0.48	-0.31±0.29	-0.13±0.43	-0.33±0.33
GPT4	—	—	1.00±0.00	0.68±0.12	0.76±0.12	0.67±0.14	0.67±0.14	0.55±0.13
GPT-few	—	—	—	1.00±0.00	0.72±0.12	0.62±0.15	0.64±0.15	0.55±0.13
GPT3.5	—	—	—	—	1.00±0.00	0.65±0.16	0.67±0.15	0.58±0.13
StarCoder	—	—	—	—	—	1.00±0.00	0.66±0.15	0.59±0.11
Code Llama	—	—	—	—	—	—	1.00±0.00	0.56±0.14
NCS	—	—	—	—	—	—	—	1.00±0.00

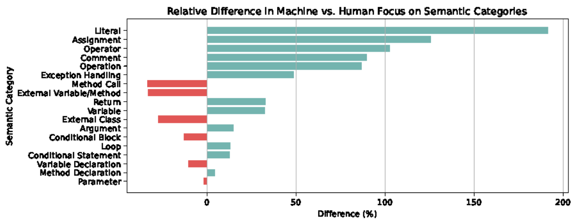

Figure 1: How much more/less do language models focus on each semantic category compared to humans?

standard deviation. In fact, for most such methods where a model and human show significant correlation in focus, the Spearman's ρ is often either around 0.5 or -0.5, but rarely in between. This suggests the relationship between human and machine foci varies significantly depending on the specific Java method.

Interestingly, although few-shot-alignment in GPT-few-shot renders the model's generated summaries more similar to those of humans, this does not lead to higher correlations between model and human foci. In addition, feature attribution in all language models is moderately or strongly positively correlated with each other on a majority of Java methods, which intuitively makes sense since all six models are based on the Transformer architecture.

We also investigate how language models' focus on each semantic category differs from that of humans. As shown in Figure 1, language models' generation of code summaries seems to be more reliant on comments, return values, and specific statements such as literals and assignments, and less reliant on method calls and variables/methods not defined explicitly within the Java method.

Discussion Point 1: We find no evidence that feature attribution in language models is correlated with programmers' visual focus. Several possible interpretations can be inferred: (1) Alternative methods may be needed to assess feature influence in black-box language models for code summarization, aiming for better alignment with human attention. (2) Access to the internal workings of proprietary models might become critical if white-box models offer more human-aligned insights into explainable language models for code [20]. (3) It is possible that language models and humans reason about code differently when summarizing source code.

Table 2: Correlation between human-machine focus alignment and summary quality (assessed by four metrics).

	Accuracy	Completeness	Conciseness	Readability
Spearman's ρ	-0.1279	0.1309	0.0194	-0.0717
p-value	0.3862	0.3753	0.8960	0.6280

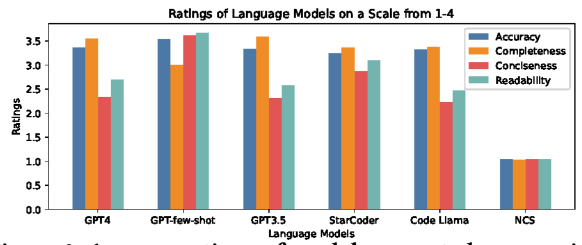

Figure 2: Average ratings of model-generated summaries.

4.2 RQ2: Summary Qualities

There is also a lack of correlation between the quality of summaries generated by language models and how well their focus on code aligns with humans'. The large p-values in Table 1 suggest that, regardless of which metric is used to assess summary quality, there is a lack of statistically significant correlation between the quality of a model-generated summary on a Java method and how well the model's focus aligns with that of humans on that Java method. Furthermore, Figure 2 shows that NCS produces worse summaries than the other five models. Although Table 1 seems to suggest that NCS's focus is more negatively aligned with human attention, we find no statistically significant metrics supporting such a claim, partially due to the small sample size of Java methods yielding statistically significant Spearman's ρ.

In general, Table 1 suggests that feature attribution in NCS is still moderately positively aligned with that in other language models on a majority of Java methods. This indicates the likelihood that aspects other than feature attribution are more indicative of and critical to a language model's performance in code summarization.

Discussion Point 2: With a substantial body of work in NLP showing that aligning neural models with human visual patterns can lead to performance improvement [26, 34, 10, 5], we contain our conclusion to the SHAP measure of feature attribution and the human attention as measured in an eye-tracking experiment. The link between human attention and feature attribution to machine models is a subject of intense scientific investigation. We contribute to the debate with this finding that SHAP did not correlate with human eye attention in the measures or models we studied.

5 CONCLUSION

In this paper, we use a state-of-the-art, black-box, perturbation-based method to assess feature attribution in language models on code summarization tasks. We then compare the model-determined important AST tokens with those identified by human visual focus, as measured through eye-tracking. The results suggest that using SHAP to measure feature attribution does not provide explainability of language models through establishing correlations between machine and human foci. Generally, our work can be interpreted in two ways. First, feature attribution measured by SHAP may not be the best way to interpret a language model's focus during code summarization as it fails to establish similarities with human focus. Alternatively, it may be the case that machines reason about code differently from humans when tasked to summarize source code.

6 ACKNOWLEDGEMENT

This research was supported by NSF CCF-2211429, NSF CCF-2211428, NSF CCF-2100035 and NSF SaTC-2312057.

REFERENCES

[1] Tobii pro fusion user manual, Jun 2023.

[2] Abid, N. J., Maletic, J. I., and Sharif, B. Using developer eye movements to externalize the mental model used in code summarization tasks. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications* (2019), pp. 1–9.

[3] Ahmad, W. U., Chakraborty, S., Ray, B., and Chang, K.-W. A transformer-based approach for source code summarization. *arXiv preprint arXiv:2005.00653* (2020).

[4] Bansal, A., Su, C.-Y., Karas, Z., Zhang, Y., Huang, Y., Li, T. J.-J., and McMillan, C. Modeling programmer attention as scanpath prediction. *arXiv preprint arXiv:2308.13920* (2023).

[5] Barrett, M., Bingel, J., Hollenstein, N., Rei, M., and Søgaard, A. Sequence classification with human attention. In *Proceedings of the 22nd conference on computational natural language learning* (2018), pp. 302–312.

[6] Collard, M. L., Decker, M. J., and Maletic, J. I. srcml: An infrastructure for the exploration, analysis, and manipulation of source code: A tool demonstration. In *2013 IEEE International conference on software maintenance* (2013), IEEE, pp. 516–519.

[7] Galassi, A., Lippi, M., and Torroni, P. Attention in natural language processing. *IEEE transactions on neural networks and learning systems 32*, 10 (2020), 4291–4308.

[8] Hooker, S., Erhan, D., Kindermans, P.-J., and Kim, B. Evaluating feature importance estimates.

[9] Huber, D., Paltenghi, M., and Pradel, M. Where to look when repairing code? comparing the attention of neural models and developers. *arXiv preprint arXiv:2305.07287* (2023).

[10] Klerke, S., Goldberg, Y., and Søgaard, A. Improving sentence compression by learning to predict gaze. *arXiv preprint arXiv:1604.03357* (2016).

[11] Kou, B., Chen, S., Wang, Z., Ma, L., and Zhang, T. Is model attention aligned with human attention? an empirical study on large language models for code generation. *arXiv preprint arXiv:2306.01220* (2023).

[12] LeClair, A., and McMillan, C. Recommendations for datasets for source code summarization. *arXiv preprint arXiv:1904.02660* (2019).

[13] Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023).

[14] Liu, Y., Tantithamthavorn, C., Liu, Y., and Li, L. On the reliability and explainability of automated code generation approaches. *arXiv preprint arXiv:2302.09587* (2023).

[15] Lundberg, S. M., and Lee, S.-I. A unified approach to interpreting model predictions. *Advances in neural information processing systems 30* (2017).

[16] Molnar, C. *Interpretable machine learning.* Lulu. com, 2020.

[17] Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., and Xiong, C. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474* (2022).

[18] OpenAI. Introducing chatgpt. https://openai.com/blog/chatgpt/, 2023. Accessed: 11/20/2023.

[19] OpenAI, R. Gpt-4 technical report. arxiv 2303.08774. *View in Article 2* (2023).

[20] Paltenghi, M., Pandita, R., Henley, A. Z., and Ziegler, A. Extracting meaningful attention on source code: An empirical study of developer and neural model code exploration. *arXiv preprint arXiv:2210.05506* (2022).

[21] Paltenghi, M., and Pradel, M. Thinking like a developer? comparing the attention of humans with neural models of code. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2021), IEEE, pp. 867–879.

[22] Rodeghero, P., and McMillan, C. An empirical study on the patterns of eye movement during summarization tasks. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (2015), IEEE, pp. 1–10.

[23] Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[24] Sharafi, Z., Shaffer, T., Sharif, B., and Guéhéneuc, Y.-G. Eye-tracking metrics in software engineering. In *2015 Asia-Pacific Software Engineering Conference (APSEC)* (2015), IEEE, pp. 96–103.

[25] Shrikumar, A., Greenside, P., and Kundaje, A. Learning important features through propagating activation differences. In *International conference on machine learning* (2017), PMLR, pp. 3145–3153.

[26] Sood, E., Tannert, S., Müller, P., and Bulling, A. Improving natural language processing tasks with human gaze-guided neural attention. *Advances in Neural Information Processing Systems 33* (2020), 6327–6341.

[27] Spearman, C. The proof and measurement of association between two things.

[28] Sundararajan, M., Taly, A., and Yan, Q. Axiomatic attribution for deep networks. In *International conference on machine learning* (2017), PMLR, pp. 3319–3328.

[29] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems 30* (2017).

[30] Wang, Y., Wang, K., and Wang, L. Wheacha: A method for explaining the predictions of code summarization models. *arXiv preprint arXiv:2102.04625* (2021).

[31] Xu, F. F., Alon, U., Neubig, G., and Hellendoorn, V. J. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (2022), pp. 1–10.

[32] Zeng, Z., Tan, H., Zhang, H., Li, J., Zhang, Y., and Zhang, L. An extensive study on pre-trained models for program understanding and generation. In *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis* (2022), pp. 39–51.

[33] Zhang, K., Li, G., and Jin, Z. What does transformer learn about source code? *arXiv preprint arXiv:2207.08466* (2022).

[34] Zhang, Y., and Zhang, C. Using human attention to extract keyphrase from microblog post. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019), pp. 5867–5872.

[35] Zhang, Z., Zhang, H., Shen, B., and Gu, X. Diet code is healthy: Simplifying programs for pre-trained models of code. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2022), pp. 1073–1084.