

Modeling Programmer Attention as Scanpath Prediction

Aakash Bansal*, Chia-Yi Su*, Zachary Karas[†], Yifan Zhang[†], Yu Huang[†], Toby Jia-Jun Li*, Collin McMillan*

*University of Notre Dame, USA

{abansal1,csu2,toby.j.li,cmc}@nd.edu

[†]Vanderbilt University, USA

{z.karas, yifan.zhang.2, yu.huang}@vanderbilt.edu

Abstract—This paper launches a new effort at modeling programmer attention by predicting eye movement scanpaths. Programmer attention refers to what information people intake when performing programming tasks. Models of programmer attention refer to machine prediction of what information is important to people. Models of programmer attention are important because they help researchers build better interfaces, assistive technologies, and more human-like AI. For many years, researchers in SE have built these models based on features such as mouse clicks, key logging, and IDE interactions. Yet the holy grail in this area is scanpath prediction – the prediction of the sequence of eye fixations a person would take over a visual stimulus. A person’s eye movements are considered the most concrete evidence that a person is taking in a piece of information. Scanpath prediction is a notoriously difficult problem, but we believe that the emergence of lower-cost, higher-accuracy eye tracking equipment and better large language models of source code brings a solution within grasp. We present an eye tracking experiment with 27 programmers and a prototype scanpath predictor to present preliminary results and obtain early community feedback.

Index Terms—scanpath prediction, human attention, eye tracking, neural networks, artificial intelligence

I. INTRODUCTION

A machine model of human programmer attention is a computer’s prediction of what information a person needs to solve a programming task. Typically, these models input the source code of the software the programmer is developing, and attempt to predict what elements are most important for that programmer’s understanding of the code. These models are academically interesting on their own to improve our knowledge of human cognition [1], but are also important building blocks to better interfaces [2], assistive technologies for individuals with disabilities [3], and even more human-like attention mechanisms in neural networks [4]. Predicting human attention has been a core component of automated software engineering techniques for years [5].

Efforts to model programmer attention have trended towards predictions of actual human behaviors. Traditionally, researchers studied interactions such as mouse clicks, key strikes, or use of features in the programmers’ Integrated Development Environments (IDE). Yet over time, more studies have focused on synthetic visual attention (e.g., people selecting screen blurs with a mouse [6]), eye fixations and tracking [7], and even live scans of brain activity [8]. The gold

standard of modeling human attention is widely considered to be from the visual system, as a person’s eye movements are considered the most concrete evidence available that a person is taking in a particular piece of information [2], [9]. The trend in the literature is towards understanding these eye movements to understand what information a person needs.

Scanpath prediction refers to predicting the sequence of *eye fixations* a person takes over a visual stimulus. An eye fixation occurs when a person looks at a location of the stimulus long enough to intake the information at that location (on the order of 100ms [10]). Scanpath prediction is a notoriously difficult problem because it depends on a person’s thought process in addition to specific environmental factors [11]. It may be easy to predict that a person will look at a bold, red word in a block of text, but it is harder to predict what word they will look at next to answer a particular intellectual question. Therefore, scanpath prediction is emerging as a major research target [12].

In this paper, we launch a new effort at modeling programmer attention via scanpath prediction. Our new effort combines new eye tracking technologies, which allow more high-accuracy eye tracking experiments to be possible at lower cost, with new Large Language Models (LLMs) of source code, which have shown much better capabilities for code comprehension than previous generations. We present:

1. An eye-tracking experiment with 27 programmers during a Java code comprehension task. We ask programmers to read source code and write short summaries describing that code. This task requires programmers to understand a snippet of code. We use equipment that is available for <US\$10k and a web browser interface, which is a quarter of the price and much simpler setup than what was used in previous studies, allowing more data samples to be collected [13].
2. A prototype scanpath predictor that is based on a 350m parameter language model of Java source code. We frame the scanpath prediction problem as a fine-tuning objective in which the model receives the Java code as a prompt and the scanpath as a followup sequence of tokens to be predicted. The idea is for the language model to learn to mimic the thought process of the programmers, insofar as the sequence of tokens that the programmers read.

Our goal is to present this paper as a paradigm for scanpath prediction of programmers, and obtain community feedback.

II. RELATED WORK

Techniques for scanpath prediction have been developed for decades. Initially, these consisted of either bio-inspired techniques that use neurophysical knowledge of the human visual system and processing [14]–[16] or statistical models that used various distributions derived from a limited amount of data to model the eye gaze pattern [17], [18]. Since 2017, neural network based eye gaze models have offered an alternative to the more traditional statistical models. In particular, generative models like PathGAN [19] and DeepGazeIII [20] have achieved state of the art performance. More recently, self-supervised techniques have been proposed that combine bio-inspired techniques and neural networks [21]. Although closely related to our problem, these models use saliency techniques specific to image processing that are often not transferable to textual data like source code.

Although eye-trackers have been intermittently used in automated SE research for over two decades, they have mainly been used as investigative tools to understand programmer cognition and behavior [22]–[26]. Models of programmer attention in software engineering research have typically used mouse cursors and keystrokes as proxy for programmer attention [27], [28]. Earlier this year, Bansal *et al.* [7] introduced a novel approach for predicting human attention using data from a 2014 eye-tracking study [13]. They introduced a model to predict programmer attention, in the form of total fixation duration. They cite the unavailability of recent eye-tracking data as a limitation for their approach. Therefore, in this paper we conduct a larger eye-tracking study designed to inform an attention prediction model. To the best of our knowledge, this paper is only the second attempt at modeling programmer attention using eye trackers, and the first to provide a prototype for scanpath prediction which is a more challenging problem.

III. EYE-TRACKING EXPERIMENT

We perform an eye-tracking experiment to extract the scanpath data of programmers for modeling programmer attention. Figure 2 shows an overview of our experiments: 1) this eye-tracking experiment, 2) the prototype in Section IV, and 3) preliminary experiment in Section V.

A. Program Comprehension Task

To record eye gaze data and extract the scanpath, we recruited programmers for a program comprehension task, specifically, code summarization. Code summarization is the task of writing natural language summary for a snippet of code, a Java method in this instance. We chose this task, because it requires the programmer to bridge the gap between the lower level understanding of source code and the high-level concepts that benefit from human cognition and attention [29].

We populate our study using the `funcom-java-long` dataset [7]. The dataset consists of high-quality Java method-summary pairs from Javadocs. This dataset is also excluded from the `jm52m` [30] dataset that was used for pre-training of our prototype, detailed in Section IV. From the roughly 8800 Java methods in their test set, we picked 68 methods at



Fig. 1. A screenshot of our interface.

random for our experiment. Each participant saw 25 randomly selected methods from the subset.

In Figure 1 we show an example of the web interface and the task we asked each participant to complete. Each task consists of a Java method displayed on the left, and a text box on the right. For this study, we recruited 27 programmers, each with 3-10 years of programming experience, and a minimum of 1 year of Java programming experience. We asked each participant to complete 25 tasks, with an average study duration of 1.5 hours. We asked participants to take a break every 20 minutes to re-calibrate the eye-tracker and minimize the effects of exhaustion [24].

B. The Eye-Tracking Hardware & Software

The eye-tracking hardware we use includes a Tobii Pro Fusion eye-tracker and a 24 inch monitor at 1920x1080 resolution and a refresh rate of 60Hz. We kept the workstation costs below \$10K, much lower than \$40K in previous studies [13]. Lower costs and the mobile nature of the eye-tracker setup allowed us to collect more data at two locations. The eye-tracker is mounted on the monitor, which is beneficial because the participants did not have to change their work-flow and were largely left to work as they would without the eye-tracker. We use the Tobii Pro Python SDK to access raw gaze data.

C. Data Collection & Processing

In Figure 2 area 1, we provide an overview of the eye-tracker data processing. We performed this study at two institutions in parallel (anonymized for review). We used the same data set, interface, hardware, and protocol for both studies. First we apply a velocity-based fixation filter, then we use a low-pass filter to remove noisy peaks [31], [32]. Next, we merge the data from both institutions. The only difference between the two setups is that one eye tracker collected the data at a sampling rate of 120Hz, while the other collected data at 60Hz due to software mismatch. This difference in sampling rates does not affect our study as we cluster fixations using methods recommended in related work for eye-tracking data with different sampling frequencies [33], [34]. The final result of this processing is 680 data points, each containing a sequences of first n fixations from a participant over a method.

IV. SCANPATH PREDICTOR PROTOTYPE

We design a prototype scanpath predictor to model programmer attention using the scanpath data we extract from the eye-tracking experiment. We provide an overview of our design in Figure 2 area 2, where we frame the problem as a fine-tuning task for an LLM. We use LLMs because they are pre-trained on large task-agnostic datasets and can be finetuned for a prediction task given a small number of examples. Although our eye-tracking apparatus is much cheaper than previous

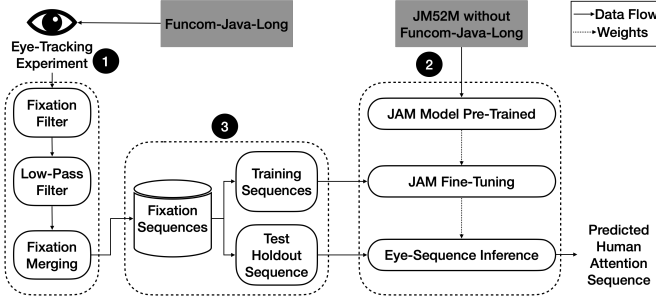


Fig. 2. Overview of our experimental setup.

```

TDAT: public void testNegativeParseCases() {
    verbose("--->Negative parse tests START");
    for (int i = 0; i < negativeParseTests.length; i++) {
        parseFilter(negativeParseTests[i], false);
    }
    checkDelete(); }
SEQ: <s> testNegativeParseCases </s>

```

Fig. 3. Finetuning prompt for the participant ID 133, method ID 31696447.

generation of eye-trackers, recruitment of programmers for the study is still a cost-limiting factor. Therefore, LLMs are good candidates for our prototype. For our prototype we use the `jam` [35] language model. We chose this specific pre-trained LLM for two reasons. First, at $\sim 350m$ parameters the model is large enough to produce meaningful results but small enough to fit on a single 24GB GPU [35]. Second, the model is pre-trained on the `jm52m` [30] dataset of 52 million Java methods which excludes the `funcom-java-long` dataset that we use for our eye-tracking experiment. Therefore, by using `jam`, we can be reasonably sure that the Java methods we use for our experiment and subsequent testing, have not been previously seen by the model during pre-training.

We finetune our prototype on the training set extracted from the eye-tracking experiment. During finetuning we provide the model with a prompt which consists of raw Java code, followed by the scanpath sequence. Figure 3 shows an example of our finetuning prompt. We use the following configurations to finetune our model:

<i>c</i>	block size	256
<i>b</i>	batch size	4
<i>e</i>	embedding dimension	768
<i>L</i>	number of layers	12
<i>h</i>	attention heads	12
<i>a</i>	accumulation steps	32
<i>r</i>	learning rate	3e-5
<i>s</i>	pre-trained iterations	27200
<i>i</i>	iterations for finetuning	200

After finetuning, we provide the model with the test set for inference. The input prompt is similar to the prompt that we use for finetuning in Figure 3, except that we stop the prompt at “SEQ:”. The prototype predicts `<s> scanpath </s>`, where `{scanpath}` is a sequence of n predictions, and `<s>` and `</s>` are start and end tags for the sequence.

V. PRELIMINARY EXPERIMENT

We conduct a preliminary experiment to evaluate the efficacy of our scanpath predictor prototype. Recall, we design our prototype to predict first n words in the scanpath. For this experiment we evaluate predictions at $n = \{1, 2, 3, 4\}$.

Before we dive into the experimental details, we introduce the research questions for this experiment:

RQ1 How accurately can our prototype predict the first n words that a programmer would look at when summarizing a Java method previously seen during finetuning?

RQ2 How accurately can our prototype predict the first n words that programmers would look at when summarizing a previously unseen Java method?

The rationale behind RQ1 is to evaluate the accuracy of the scanpath predictor prototype against reference data from a particular programmer. Note, the model has seen the scanpath of the other participants over the same Java method during finetuning. The goal is to evaluate the correlation between the predicted scanpath and a human programmer’s scanpath.

The rationale behind RQ2 is to challenge the scanpath predictor prototype. Note, the model has never seen this Java method during pre-training or finetuning. The goal is to evaluate how accurately the prototype predicts the scanpath over an unseen method, compared to the human programmers.

A. Holdout Experiment Setup

In Figure 2 area 3, we provide an overview of our preliminary experiment. Our goal is to compare the predictions from our prototype against reference scanpath from human subjects. However, different programmers might look at different words in the sequence. Therefore, we do not test our approach against any one programmer’s data. Instead, we use a one-holdout approach and create 95 versions of our dataset.

For RQ1, we take a `participant-holdout` approach, where we holdout all the datapoints from one participant as the test set. The training data in each instance has 26 participants, of which 1 participant is held out for validation. The test data in each instance has scanpaths from 1 participant over each of the 25 methods they saw during the experiment.

For RQ2, we take a `method-holdout` approach, where we holdout all the datapoints from for one Java method as the test set. The training data in each instance has 67 methods, of which 1 method is held out for validation during finetuning. The test data in each instance has scanpaths from 1 method over a variable number of participants because not every participant saw every method. Recall from Section III, we have 27 participants and 68 methods. Therefore, we processed 27 datasets for the `participant-holdout` experiments and 68 datasets for the `method-holdout` experiment.

B. Metrics

We report two metrics from related work to evaluate the performance of our prototype scanpath predictor.

Levenshtein is the string-matching metric that uses the Levenshtein edit distance, which is a popular scanpath metric [36], to compute character level string comparison. We use the `TheFuzz` [37] implementation.

Gestalt is a popular pattern-matching metric that uses the Gestalt Pattern Matching algorithm to compute the similarity between two sequences. We use the `pymatcher` [38] implementation.

VI. RESULTS FOR PRELIMINARY EXPERIMENT

We present the results for both of our preliminary experiments to evaluate our prototype scanpath predictor. In Table I we show the average scores over for each experiment.

A. RQ1: Participant Holdout Experiment

For the `participant-holdout` experiment, we observe that the Levenshtein score is the highest at 0.46 when $n = 4$, which means that the prototype is slightly better at predicting longer scanpaths than shorter $n \leq 3$. This is a promising result because we expect future research to focus on predicting scanpaths at $n > 4$. On the other hand, for the Gestalt metric, the highest score is 0.442 at $n = 1$. We observe that the Levenshtein scores show a consistent increase from $n = 1 - 4$, while the Gestalt metric is less comparative between $n = 2 - 4$. Each of the scores in Table I is an average over 680 prediction-reference pairs, therefore we dig a little deeper into the score distribution to understand these scores.

In Figure 4 (a), we show a frequency distribution bar-chart of our Levenshtein scores. We observe that for $n = 1$, the highest frequency is at the Levenshtein score of 1.0, which means a perfect match for 168 out of 680 samples. This is a promising result for our prototype, because it shows that it learns to pick the first word correctly out of all the words in the Java method for 25% of the samples. For $n = 2, 3, 4$ we observe a sharp decrease in perfect matches with score 1, but a considerable increase in number of samples with scores in the $[0.4 - 0.8]$ range, which is above the average scores we saw in Table I. Overall, we find that although our prototype achieves higher average scores for $n = 4$, the number of perfect matches are very few (< 5). One possible explanation is that scanpaths vary between participants and neighboring words in a code sequence may not match the target word, making a perfect match hard to achieve for longer scanpaths.

B. RQ2: Method Holdout Experiment

For the `method-holdout` experiment, we observe a highest Levenshtein scores are less comparative for $n = 1 - 4$ with the highest score of 0.343 for $n = 3$. The Gestalt score is highest for $n = 1$, and is more comparative with a consistent trend of decreasing values for $n = 1 - 4$. Overall, we see that the average scores are lower compared to RQ1. We expect this because predicting scanpath over an unseen method is a harder task. The model did not see this method during pre-training, making it more challenging to learn word embeddings for unique words such as variable names.

To investigate further, we show a frequency distribution bar-chart of Levenshtein scores in Figure 4 (b). We note that for $n = 2, 3, 4$ we observe a sharp decrease in the number of perfect matches when compared to $n = 1$, and

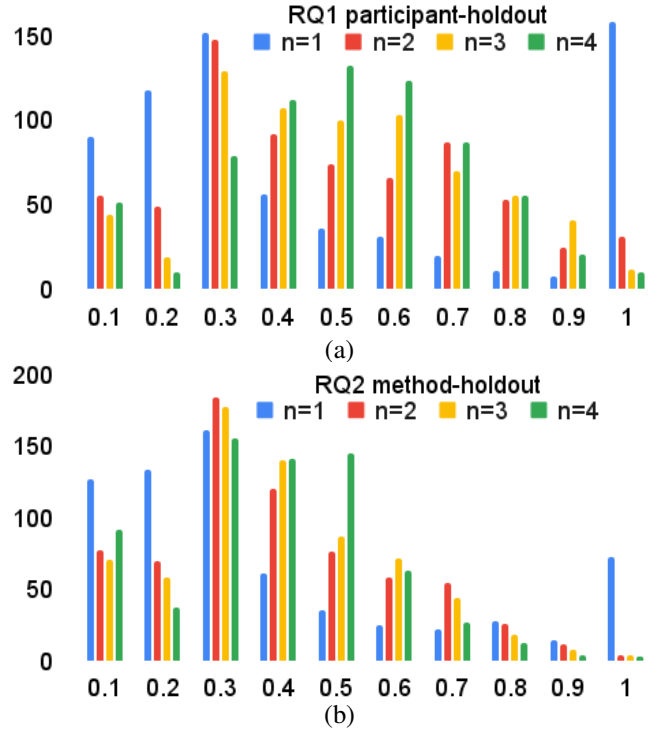


Fig. 4. Frequency charts for (a) participant-holdout and (b) method-holdout experiments. There are 680 samples. The x axis indicates bins of Levenshtein score, the y axis indicates number of samples. a significant increase in number of samples with scores in the $[0.4 - 0.8]$ range. This aligns with our expectations, in that predicting scanpaths over unseen methods is a harder task, specially for longer scanpaths. We expect future work with bigger eye-tracking datasets to improve these results because our prototype may benefit from seeing scanpaths of similar methods during finetuning and learn from them.

VII. CONCLUSION

In this new ideas paper we make three main contributions to model programmer attention. First, we present an eye-tracking experiment designed to extract scanpath data from programmers. Second, we frame the scanpath prediction problem as a finetuning task for an LLM to develop a novel prototype. Third, we show how well our prototype correlates with human scanpaths with a preliminary experiment. The main goal of this paper is to provide a framework for future research towards modeling programmer attention, specifically by automatically predicting eye movements such as scanpaths. Potential applications of this work are towards building more human-like neural networks [39], SE virtual assistants that consider human factors [40], and assisted systems for low vision and disabled programmers [41], [42] to name a few.

We provide a repository for the replication of our results at: <https://github.com/apcl-research/scanpathpred>

ACKNOWLEDGMENT

This work is supported in part by the NSF CCF-2100035 and CCF-2211428. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

TABLE I
AVERAGE GESTALT AND LEVENSHTEIN SCORES FOR RQ1 AND RQ2.

Experiment	Levenshtein				Gestalt			
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
participant-holdout	0.431	0.434	0.452	0.460	0.442	0.421	0.429	0.424
method-holdout	0.337	0.342	0.343	0.332	0.347	0.332	0.315	0.295

REFERENCES

- [1] S. Eivazi and R. Bednarik, "Predicting problem-solving behavior and performance levels from visual attention data," in *Proc. workshop on eye gaze in intelligent human machine interaction at IUI*, 2011, pp. 9–16.
- [2] F. Lu and X. Chen, "Person-independent eye gaze prediction from eye images using patch-based features," *Neurocomputing*, vol. 182, pp. 10–17, 2016.
- [3] P. Majaranta, *Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies: Advances in Assistive Technologies*. IGI Global, 2011.
- [4] Q. Lai, S. Khan, Y. Nie, H. Sun, J. Shen, and L. Shao, "Understanding more about human and machine attention in deep neural networks," *IEEE Transactions on Multimedia*, vol. 23, pp. 2086–2099, 2020.
- [5] N. Peitek, S. Apel, C. Parnin, A. Brechmann, and J. Siegmund, "Program comprehension and code complexity metrics: An fmri study," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 524–536.
- [6] M. Paltenghi and M. Pradel, "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)*. IEEE, 2021, pp. 867–879.
- [7] A. Bansal, B. Sharif, and C. McMillan, "Towards modeling human attention from eye movements for neutral source code summarization," *Proceedings of ACM Human-Computer Interaction*, Vol. 7, 2023.
- [8] N. Peitek, J. Siegmund, C. Parnin, S. Apel, J. C. Hofmeister, and A. Brechmann, "Simultaneous measurement of program comprehension with fmri and eye tracking: A case study," in *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*, 2018, pp. 1–10.
- [9] F. Huttmacher, "Why is there so much more research on vision than on any other sensory modality?" *Frontiers in psychology*, vol. 10, p. 2246, 2019.
- [10] B. R. Manor and E. Gordon, "Defining the temporal threshold for ocular fixation in free-viewing visuocognitive tasks," *Journal of neuroscience methods*, vol. 128, no. 1-2, pp. 85–93, 2003.
- [11] C. M. Privitera, "The scanpath theory: its definition and later developments," in *Human vision and electronic imaging xi*, vol. 6057. SPIE, 2006, pp. 87–91.
- [12] M. Kümmerer and M. Bethge, "State-of-the-art in human scanpath prediction," *arXiv preprint arXiv:2102.12239*, 2021.
- [13] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D'Mello, "Improving automated source code summarization via an eye-tracking study of programmers," in *Proceedings of the 36th international conference on Software engineering*, 2014, pp. 390–401.
- [14] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.
- [15] D. Zanca, S. Melacci, and M. Gori, "Gravitational laws of focus of attention," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 12, pp. 2983–2995, 2019.
- [16] R. Engbert, H. A. Trukenbrod, S. Barthelmé, and F. A. Wichmann, "Spatial statistics and attentional dynamics in scene viewing," *Journal of vision*, vol. 15, no. 1, pp. 14–14, 2015.
- [17] X. Sun, H. Yao, R. Ji, and X.-M. Liu, "Toward statistical modeling of saccadic eye-movement and visual saliency," *IEEE Transactions on Image Processing*, vol. 23, no. 11, pp. 4649–4662, 2014.
- [18] A. D. Clarke, M. J. Stainer, B. W. Tatler, and A. R. Hunt, "The saccadic flow baseline: Accounting for image-independent biases in fixation behavior," *Journal of vision*, vol. 17, no. 11, pp. 12–12, 2017.
- [19] M. Assens, X. Giro-i Nieto, K. McGuinness, and N. E. O'Connor, "Pathgan: Visual scanpath prediction with generative adversarial networks," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.
- [20] M. Kümmerer, M. Bethge, and T. S. Wallis, "Deepgaze iii: Modeling free-viewing human scanpaths with deep learning," *Journal of Vision*, vol. 22, no. 5, pp. 7–7, 2022.
- [21] L. Schwinn, D. Precup, B. Eskofier, and D. Zanca, "Behind the machine's gaze: Neural networks with biologically-inspired constraints exhibit human-like visual attention," *Transactions on Machine Learning Research*.
- [22] B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, 2012, pp. 381–384.
- [23] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc, "A systematic literature review on the usage of eye-tracking in software engineering," *Information and Software Technology*, vol. 67, pp. 79–107, 2015.
- [24] Z. Sharafi, T. Shaffer, B. Sharif, and Y.-G. Guéhéneuc, "Eye-tracking metrics in software engineering," in *2015 Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2015, pp. 96–103.
- [25] Z. Sharafi, Y. Huang, K. Leach, and W. Weimer, "Toward an objective measure of developers' cognitive activities," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–40, 2021.
- [26] N. Tang, M. Chen, Z. Ning, A. Bansal, Y. Huang, C. McMillan, and T. J.-J. Li, "An empirical study of developer behaviors for validating and repairing ai-generated code," in *13th Annual Workshop at the Intersection of PL and HCI (PLATEAU 2023)*, 2023.
- [27] D. Huber, M. Paltenghi, and M. Pradel, "Where to look when repairing code? comparing the attention of neural models and developers," *arXiv preprint arXiv:2305.07287*, 2023.
- [28] H. Liu, O. N. N. Fernando, and J. C. Rajapakse, "Predicting affective states of programming using keyboard data and mouse behaviors," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, 2018, pp. 1408–1413.
- [29] T. J. Biggerstaff, B. G. Mitbender, and D. E. Webster, "Program understanding and the concept assignment problem," *Communications of the ACM*, vol. 37, no. 5, pp. 72–82, 1994.
- [30] A. P. C. L. at Notre Dame, "jm52m Dataset," <https://huggingface.co/datasets/apcl/jm52m>, 2023.
- [31] A. Olsen, "The tobii i-vt fixation filter," *Tobii Technology*, vol. 21, pp. 4–19, 2012.
- [32] D. J. Mack, S. Belfanti, and U. Schwarz, "The effect of sampling rate and lowpass filters on saccades—a modeling approach," *Behavior Research Methods*, vol. 49, pp. 2146–2162, 2017.
- [33] A. D. Ouzts and A. T. Duchowski, "Comparison of eye movement metrics recorded at different sampling rates," in *Proceedings of the symposium on eye tracking research and applications*, 2012, pp. 321–324.
- [34] J. Trabulsi, K. Norouzi, S. Suurmets, M. Storm, and T. Z. Ramsøy, "Optimizing fixation filters for eye-tracking on small screens," *Frontiers in Neuroscience*, vol. 15, p. 578439, 2021.
- [35] C.-Y. Su, A. Bansal, V. Jain, S. Ghanavati, and C. McMillan, "A language model of java methods with train/test deduplication," 2023.
- [36] R. Fahimi and N. D. Bruce, "On metrics for measuring scanpath similarity," *Behavior Research Methods*, vol. 53, pp. 609–628, 2021.
- [37] SeatGeek, "The fuzz," 2020. [Online]. Available: <https://github.com/seatgeek/thefuzz>
- [38] A. Niyas, "pyymatcher," 2020. [Online]. Available: <https://github.com/abdulniyaspm/pyymatcher>
- [39] B. Peters and N. Kriegeskorte, "Capturing the objects of vision with neural networks," *Nature human behaviour*, vol. 5, no. 9, pp. 1127–1144, 2021.
- [40] Z. Eberhart, A. Bansal, and C. McMillan, "A wizard of oz study simulating api usage dialogues with a virtual assistant," *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 1883–1904, 2020.
- [41] A. Armaly, P. Rodeghero, and C. McMillan, "A comparison of program comprehension strategies by blind and sighted programmers," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 788–788.
- [42] V. Potluri, T. Grindeland, J. E. Froehlich, and J. Mankoff, "Ai-assisted ui design for blind and low-vision creators," in *the ASSETS'19 Workshop: AI Fairness for People with Disabilities*, 2019.