Cognitive Attention Map: Semantic Tokenization of Source Code via OCR-Based AOI Identification across Performance Levels

Wudao Yang
Universiti Malaya
Kuala Lumpur, Malaysia
Yunnan Minzu University
Kunming, China
s2137045@siswa.um.edu.my

Mohammad Asaduzzaman Rasel
School of Information Technology
Monash University
Kuala Lumpur, Malaysia
rasel.mdas@gmail.com

Seng Yue Wong
Universiti Malaya
Kuala Lumpur, Malaysia
wongsengyue@um.edu.my

Unaizah Obaidellah
Universiti Malaya
Kuala Lumpur, Malaysia
unaizah@um.edu.my

Abstract

Understanding how visual attention is attended by programmers is critical to progress in the field of programming education. Conventional eye-tracking studies typically employ basic and coarsely defined Areas of Interest (AOIs), which fail to capture the underlying semantic content of source code. We present an automated system that produces Cognitive Attention Maps using Optical Character Recognition (OCR) technology-based AOIs and semantic tokens. Our system extracts bounding boxes from rendered code, enabling the alignment of gaze data with semantically relevant code components, such as identifiers, operators, keywords, and literals. The proposed framework facilitates multi-dimensional visual attention profiling (frequency and duration). Our framework, tested on programming tasks, reveals visual attention and performance-based differences that are not possible with conventional AOIs. Our study offers an in-depth and sophisticated perspective on programmer cognition. Future work will aim to enhance scanpath prediction by improving the semantic tokens' integration to achieve more precise visual behavior prediction.

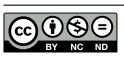
CCS Concepts

• **Human-centered computing** → **Visual analytics**.

Keywords

Eye-tracking (ET), Program comprehension, Cognitive attention map (CAM), Optical character recognition (OCR), Semantic tokenization, Areas of Interest (AOI), Programming performance, Visual attention analysis

ACM Reference Format:
Wudao Yang, Mohammad Asaduzzaman Rasel, Seng Yue Wong, and Unaizah Obaidellah. 2026. Cognitive Attention Map: Semantic Tokenization of Source Code via OCR-Based AOI Identification across Performance Levels. In *2026 Symposium on Eye Tracking Research and Applications (ETRA '26), June 01–04, 2026, Marrakesh, Morocco.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3797246.3806014



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
ETRA '26, Marrakesh, Morocco
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2519-7/26/06
https://doi.org/10.1145/3797246.3806014

1 Introduction

Eye Tracking (ET) has emerged as a powerful method for decoding the complex and multifaceted cognitive processes involved in program comprehension. By capturing real-time visual attention patterns, researchers gain valuable insights into how developers navigate and interpret source code. However, the analytical depth of most ET studies is hampered by conventional definitions of Areas of Interest (AOIs). AOI definitions have often relied on traditional coarse-grained or manually delineated regions, such as lines of code. This broad-brush approach obscures the underlying structural and semantic structure of the stimulus, leading to ambiguous results that lack the precision required for deep cognitive modeling. Furthermore, manual AOI definition is a labor-intensive process requiring significant effort and time. This inability to scale makes it challenging to apply this method to large-scale data or varied experimental stimuli. In such a restrictive framework, it has been difficult for researchers to isolate the factors that affect visual attention using the combined entities of semantic tokens, identifiers, operators, literals, and keywords. We refer semantic tokens as a collection of relevant tokens that make up a line of code such as the function call (i.e. `print(num)`), identifier keywords (i.e. `int, print, while`), and operator (i.e. `!=, *!`). To address these bottlenecks, we propose a novel framework to generate Cognitive Attention Maps (CAMs) using automated OCR-based token-level AOI identification[Yang et al. 2025b] and semantic tokenization. Our proposed framework (Fig. 1) utilizes OCR techniques to automatically extract bounding box AOIs for each token, which are then grouped together to form semantically meaningful tokens. By mapping ET fixations to these individual AOIs, we are then able to quantify the frequency and duration of fixations to each functional code element. Our proposed framework consists of two phases: Phase 1 consists of automated data preparation and semantic token creation, while Phase 2 consists of executing attention distribution analysis and visualization. Our proposed framework will be analyzed across several tasks with increasing levels of complexity to address three research questions:

- **RQ1:** Can OCR reliably extract token-level semantic units for ET analysis?

- **RQ2:** How is programmers' attention distributed across semantic tokens by frequency and duration?
- **RQ3:** What performance-related differences emerge in multi-dimensional attention profiles?

This research will contribute to the advancement of the field of fine-grained and high-resolution cognitive modeling by enhancing methodological precision and ecological validity. The proposed CAM framework will improve the analysis granularity and lay a solid groundwork for the design of a gaze-aware adaptive learning system and a performance-sensitive instructional design methodology.

2 Related Work

Methodological research on eye-tracking (ET) in program comprehension has established a strong empirical foundation for understanding how expertise shapes gaze behavior. Busjahn et al.[Busjahn et al. 2015] have studied gaze patterns related to debugging by novices and experts. Busjahn et al. [Busjahn and Schulte 2013] and Sharafi et al. [Sharafi et al. 2015] have studied visual navigation strategies employed by programmers at various expertise levels. Jessup et al. [Jessup et al. 2021] have found that experts and novices differ in fixation count for code comprehension, though differences in fixation duration were not significant. Cheng et al. [Cheng et al. 2024] have employed eye-tracking in combination with retrospective think-aloud protocols to show that high-performing students use more sophisticated self-regulated learning strategies. Mansoor et al. [Mansoor et al. 2024] demonstrated that bug fixing requires higher cognitive load than feature addition, while Andrzejewska et al. [Andrzejewska and Kotoniak 2020] demonstrated that programming students improve their efficiency in developing comprehension strategies over time. McChesney et al. [McChesney and Bond 2019] have found that programmers with dyslexia show gaze patterns that differ from those for natural language reading. Yang et al. [Yang et al. 2025a] have employed a combination of DBSCAN clustering and Sankey diagrams to analyze eye-tracking data for expertise and performance-level differences in fixation patterns, showing cognitive processing differences in novice programmers.

Complementing this methodological body of work, a parallel line of research has focused on tool-based and applied systems for gaze analysis in software engineering contexts. The iTrace-Toolkit [Behler et al. 2023] provides support for mapping eye-tracking data with source code tokens for further analysis. However, iTrace operates within structured source code environments and relies on predefined mappings between gaze coordinates and IDE-resident code elements. It does not support fully automated OCR-based extraction of token-level AOIs from rendered code stimuli in images format, which constrains its applicability when code is presented in image form or outside of an IDE context. Furthermore, the coarse granularity of its default AOI definitions limits the depth of semantic analysis that can be performed. In parallel, OCR-based approaches have been developed to support more automated and flexible gaze analysis. Yang et al. [Yang et al. 2025b] have proposed a cognitive modeling framework, Gaze2Code, that employs optimized OCR-based AOI generation for eye-tracking attention modeling. However, the Gaze2Code method only introduces the methodological pipeline of OCR-based AOI identification without comparing the attentional levels between different performance levels (high vs low performers). The proposed work aims to bridge this gap. Jin et al. [Jin et al. 2022] proposed the Text-VQA method, which reduces the impact of OCR errors by using token-word contrastive learning with multimodal transformers. Ryabinin et al. [Ryabinin et al. 2022] proposed an automated solution using OCR and computer vision to find areas of interest in mobile eye-tracking video streams.

More recently, application-oriented systems have begun to integrate gaze information directly with machine learning models for downstream software engineering tasks. Zhang et al. [Zhang et al. 2024] proposed EyeTrans, a system that integrates human gaze features with the self-attention mechanisms of Transformer models to improve neural code summarization performance. EyeTrans demonstrates the practical value of incorporating gaze signals into deep learning pipelines; however, its primary objective is optimizing a downstream ML task rather than modeling fine-grained cognitive attention at the semantic token level. It does not provide mechanisms for analyzing how individual code tokens, categorized by semantic types, evaluating programmers' attention during comprehension, nor does it address automated AOI extraction from rendered code stimuli in images format.

Existing approaches either rely on coarse AOIs or focus on downstream machine learning tasks, limiting their ability to model fine-grained cognitive attention. In contrast, our approach introduces OCR-based token-level AOI extraction and multi-dimensional attention modeling (i.e. fixation metrics based on frequency or duration). This provides a more precise analysis of programmers' comprehension strategies which is particularly useful in identify novice programmers across diverse performance levels.

3 Technical Approach: OCR-Based Semantic Attention Pipeline

This section presents our proposed technical approach for constructing Cognitive Attention Maps (CAMs) through OCR-based token-level AOI identification. Unlike traditional eye-tracking pipelines that rely on manually defined or line-based Areas of Interest (AOIs), our approach introduces an automated and fine-grained semantic tokenization process. The novelty of this pipeline lies in three key aspects: (1) the use of OCR to automatically extract token-level AOIs from rendered code, eliminating manual annotation, (2) the organization of AOIs into semantically meaningful token categories that enable analysis at the level of program logic, and (3) the integration of fixation frequency and duration at the semantic level to support multi-dimensional modeling of visual attention across performance levels.

The proposed pipeline consists of two phases and introduces a novel mechanism for transforming raw eye-tracking data into semantically meaningful attention representations.(Fig. 1) ET data are collected and fixation features are extracted. A key novelty in this phase is the use of OCR to automatically detect token-level AOIs, eliminating the need for manual annotation and enabling scalable analysis across diverse code stimuli. These tokens are categorized into semantic groups such as identifiers, keywords, operators, and literals, and the fixation data are aligned with the corresponding token locations. Unlike prior approaches that treat AOIs as coarse spatial regions (e.g., lines of code), our method models attention

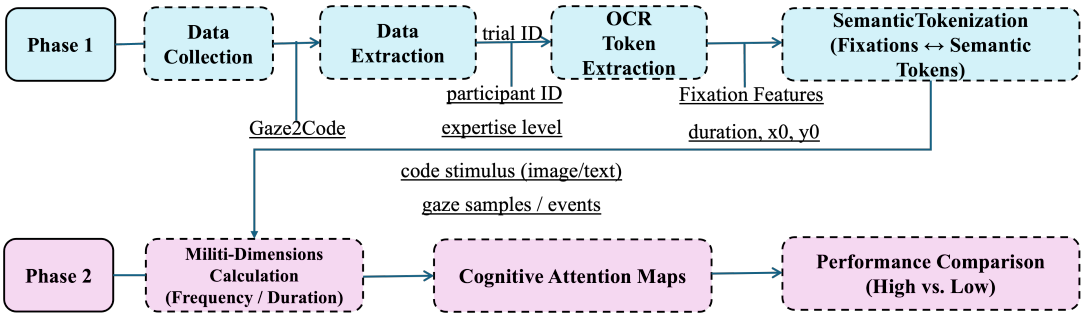


Figure 1: Overview of the proposed pipeline for constructing Cognitive Attention Maps. ET data are processed to extract fixation features, while OCR identifies token-level Areas of Interest (AOIs) from code stimuli. Fixations are aligned with semantic tokens to compute token-level metrics (frequency and duration), enabling visualization and performance comparison.

at the semantic token level, allowing more precise alignment between gaze behavior and program structure. Fixation metrics are aggregated to compute attention measures, specifically fixation frequency and fixation duration. Another novel aspect of the pipeline is the integration of fixation frequency and duration at the semantic level, enabling multi-dimensional attention profiling and more comprehensive characterization of cognitive strategies. These metrics are used to generate Cognitive Attention Maps (CAMs) and support performance comparisons between two different programmer groups.

Compared to line-based AOIs, which treat each line of code as a single unit of analysis, the proposed approach operates at the level of semantic tokens. For example, a single line of code such as `print(num)` would be treated as one AOI in line-based approaches, whereas in our proposed method the respective line of code is decomposed into semantically meaningful components such as the function call (i.e. `print(num)`), identifier keywords (i.e. `int, print, while`), and operator (i.e. `!=, *!`). This enables a more precise alignment between gaze behavior and program semantics.

While tools such as iTrace support mapping gaze data to source code elements, they typically rely on structured code representations and predefined mappings. In contrast, our approach uses OCR to automatically extract token-level AOIs directly from rendered code stimuli, enabling greater flexibility and scalability without manual annotation or dependency on source code structure.

4 Method

4.1 Participants

The participants of the study were composed of 84 undergraduate students of an introductory Python programming course at Yunan Minzu University. This study was conducted in collaboration with Yunnan Minzu University and Universiti Malaya. They were volunteers. The final set of data consisted of 84 participants, with an age range of 19 to 22 years (M_{age} = 19.89, SD_{age} = 0.73). Among them, 35 participants were male (M_{age} = 19.89 years, SD_{age} = 0.72), and 49 participants were female (M_{age} = 19.90 years, SD_{age} = 0.74). The subjects had basic programming knowledge gained from their course work. They were familiar with basic Python programming concepts. During the experiment, they carried out a series of Python code comprehension tasks. The eye movements of the participants of the study were recorded using a Tobii Pro Nano eye tracker with a sampling rate of 60 Hz. The participants were grouped based on their aggregated scores of their performance across all five questions answered. Participants were identified as a high performance if they scored at least 59% of 100%. There were a total of 25 low-performer, and 59 high-performers participants.

4.2 Experimental Procedure and Setup

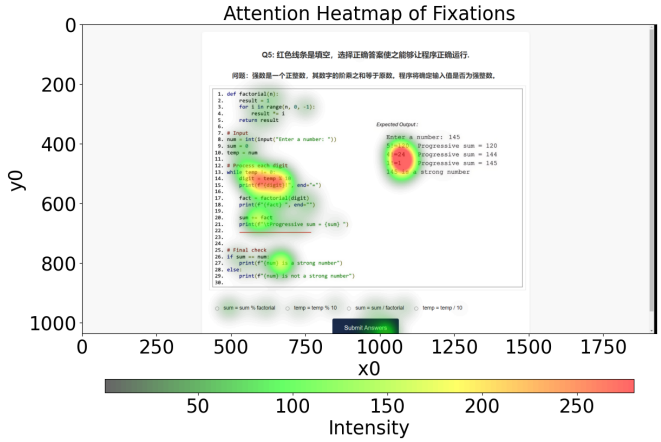
Before recording participants' eye movements, a standard eye calibration using the eye tracker was conducted. Afterwards, the subjects were presented with Python code comprehension questions (Q1-Q5), in which code snippets were displayed in image form along with multiple-choice questions. The code comprehension questions were presented in a set order, progressing from simple syntax understanding to more complex code comprehension. They were allowed to answer at their own pace without time limits while their eye movements were being recorded. After the experiment, the collected ET data and code stimuli were jointly processed to enable fine-grained analysis. First, fixation events were identified from raw gaze data using the Tobii I-VT fixation detection algorithm. Meanwhile, the code stimuli were processed using OCR to automatically extract token-level AOI and semantic token categories (e.g., identifiers, keywords, operators, literals, punctuation). Each fixations was then spatially aligned with the corresponding token AOI based on the fixation coordinates (x_0, y_0) and the AOI bounding boxes $(x, y, width, height)$. This alignment allowed each fixation to be linked with a specific code token, thus enabling the collection of text from the token, the AOI attributes, the fixation duration, the participant identity, and the expertise level. The fixation metrics were then aggregated at the token level, thus converting the gaze data into attention units with semantic meaning and enabling the construction of the CAMs for further analysis. For instance, a fixation in the region of a semantic expression in the code, such as `print(num)`, is associated with the entire expression rather than individual tokens, thus allowing the analysis of attention based on the expression.

5 Results and Analysis

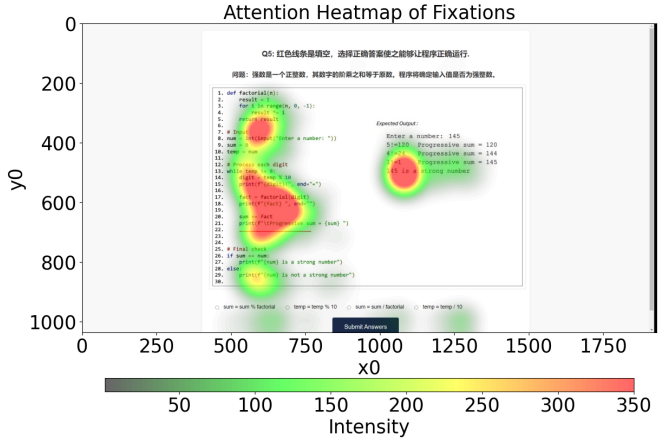
This section reports the distribution of visual attention among high- and low-performance programmers while comprehending code snippets based on Cognitive Attention Maps (CAMs). As the proposed framework uses the fine-grained semantic token level, the CAMs allow us to visually and quantitatively analyze the distribution of visual attention among high- and low-performance programmers. Our results reveal that the proposed OCR-based semantic tokenization framework is able to capture the fine-grained distribution of visual attention, which is not possible with traditional line-based AOIs. Although some tokens may visually align with code lines, the underlying representation operates at a finer semantic granularity, allowing attention to be analyzed at the level of individual code elements rather than coarse line-based regions. Qualitatively, the heatmap plots reveal that high-performance programmers show more focused visual attention on semantically important code elements, while low-performance programmers show more distributed visual attention. Quantitatively, the low-performance programmers receive more fixated frequency and longer fixated duration on the output statements and literals, while high-performance programmers show more focused visual attention on control structures, operators, and function calls. These trends become more pronounced as the complexity of the tasks increases from Q1 to Q5. However, due to space limitations, only Q5 results are shown. Due to space constraints, we present detailed results for Q5, which represents the most challenging code comprehension task in the experiment. Q5 involves more complex semantic structures and control flow , making it particularly suitable for highlighting differences in attention patterns across performance levels. However, the results for Q1-4 (not reported in this paper) remain consistent between the two groups (with less differences for Q1 due to similar score performance for both groups).

5.1 Heatmap Visualization

The heatmaps shown in Fig. 2 are generated by aggregating fixation data across participants within each performance group. The high-performance group consists of 83 participants, while the low-performance group consists of 83 participants. Heatmaps were created by mapping the fixation points on the code stimulus. Fixation points from all participants within each group are combined and projected onto the same code stimulus, and a Gaussian Kernel Density Function (GKDF) is applied to generate the aggregated attention distribution. Fig. 2 is a comparative analysis of the attention based on the fixation points among high and low performance programmers. In the high performance group shown in Fig. 2a, more focal attention is distributed across the semantically important parts of the code. However, the low performance group shown in Fig. 2b, exhibited more dispersed and less efficient allocation visual attention. Although individual differences in gaze behavior exist, aggregation allows us to identify consistent group-level attention patterns and highlight systematic differences between high- and low-performance programmers. To mitigate the effect of individual variability, fixation data are normalized before aggregation, ensuring that the resulting heatmaps reflect generalizable trends rather than outlier behaviors. We acknowledge that aggregation may obscure fine-grained individual differences; however, the purpose of Fig. 2 is to provide an overall comparative visualization of group-level attention strategies. Detailed individual-level analysis is further supported by the token-level frequency and duration analysis presented in subsequent sections.



(a) High Performance



(b) Low Performance

Figure 2: Fixation Attention Heatmaps by Performance Level

5.2 Semantic Tokenization Comparisons

In this section, Fig. 3 and Fig. 4 present quantitative comparisons of fixation frequency and fixation duration, respectively, across semantic token types for high- and low-performance programmers. To address concerns about the readability and quantitative interpretability of earlier visualizations, the alternative word cloud representations Fig. 3(a-b) and 4 (a-b) have been presented with grouped bar charts (Fig. 3c and 4d, which allow for direct numerical comparison across token categories between performance groups. These plots summarize the distribution of the attention over the token types and do not directly relate to the spatial locations of tokens within specific lines of code.

5.2.1 By Frequency. Fig. 3 presents a quantitative comparison of fixation frequency across semantic token types for high- and low-performance programmers. As shown in Fig. 3, high-performance programmers exhibit higher fixation frequencies on structurally and semantically significant tokens, including function calls, key identifiers, and operators, reflecting a selective attention strategy focused on program logic. This reveals that high-performance programmers concentrate more on the tokens that directly impact the logical structure of the program. In contrast, low-performance programmers show markedly higher fixation frequencies on output-related statements, literals, and result tokens across nearly all token categories. This reveals that low-performance programmers repeatedly look at the same tokens while trying to comprehend the program. They use inefficient strategies while comprehending the code. The bar chart enables a direct quantitative reading of these differences: low-performance programmers accumulate substantially more total fixations and distribute them across a wider range of token types, while high-performance programmers apply more concentrated and efficient attention.

5.2.2 By Duration. Fig. 4 presents a quantitative comparison of fixation duration across semantic token types for high- and low-performance programmers. As shown in the Fig. 4, the fixation durations of high-performance programmers are, on average, shorter and more concentrated on semantically important tokens, including function calls, identifiers, and control structures, indicating efficient and targeted cognitive processing. On the other hand, low-performance programmers exhibit substantially longer fixation durations across nearly all token categories, with the most pronounced differences observed on output statements and literals, particularly `print()`. The bar chart representation makes these quantitative differences directly readable, with low-performance participants' durations exceeding those of high-performance participants for most token types. Compared to the frequency-based representation, the fixation duration-based representation shows a wider range of attended semantic tokens (i.e. more tokens attentional coverage - see x-axis of Fig. 4c), indicating the effectiveness of fixation duration as a metric for evaluating the allocation of cognitive resources for code comprehension on different and potentially more meaningful tokens.

6 Frequency vs. Duration

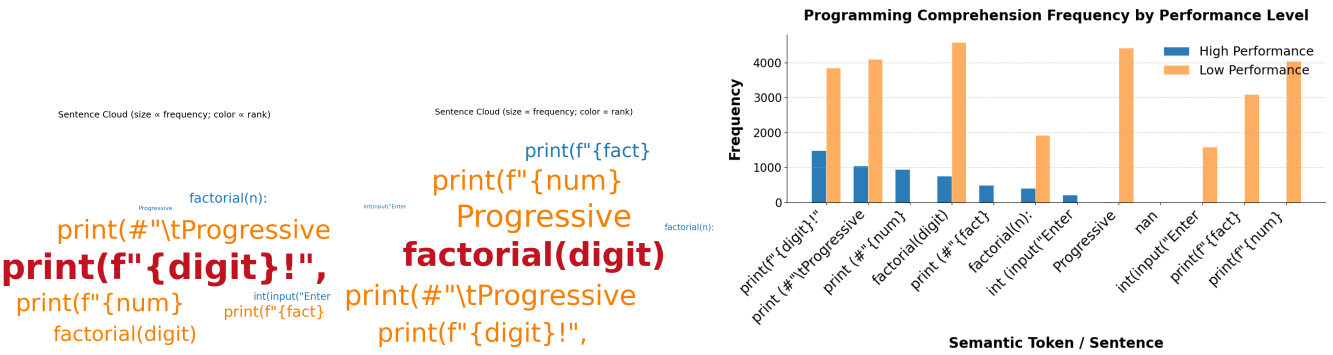
The radar chart in Fig. 5, plots the fixation frequency and duration for selected and comparable tokens between high- and low-performance programmers comparing the relevant lines of codes (whereas Fig. 4 highlights the amount of attention invested at the token-level within the lines of code). It can be observed in Fig. 5a that low-performance participants have much higher fixation frequencies on almost all tokens, especially on `factorial(digit)` and output statements like `print(f"digit!")` and `print(f"num")`, while high-performance programmers have relatively fewer fixations. Fig. 5b shows the fixation duration, which has an even clearer distinction, showing that low-performance participants have longer fixation duration on almost all tokens, while high-performance programmers have shorter fixation duration on structurally important tokens. While both fixation frequency and duration can be used to differentiate high- and low-performance programmers, the latter performs better, indicating that programming performance is related to efficient visual attention.

7 Discussion

While this study addresses three research questions, they are closely related and collectively examine semantic-level attention in program comprehension. Rather than representing separate lines of inquiry, they provide complementary perspectives on the same underlying phenomenon. This research aimed to determine whether there is a deeper level of understanding that can be obtained from analyzing eye-tracking (ET) data in terms of its semantics in relation to a programmer's visual attention. The results of this research answered all research questions raised in this paper, thus proving the worth of the proposed OCR-based Cognitive Attention Map (CAM) framework. For RQ1, the results of this research revealed that it is indeed possible to reliably obtain Areas of Interest (AOIs) from rendered code stimuli using the proposed OCR-based framework. The results also revealed that it is possible to accurately match eye-tracking data with the obtained AOIs. The automatic detection of tokens in code images using bounding boxes eliminates the tedious task of manually annotating AOIs in code images. The results also revealed that it is indeed possible to obtain a more precise level of analysis by extracting tokens rather than using traditional line-based AOI extraction. However, more studies (across stimulus type, and participants expertise) need to be performed to validate the findings.

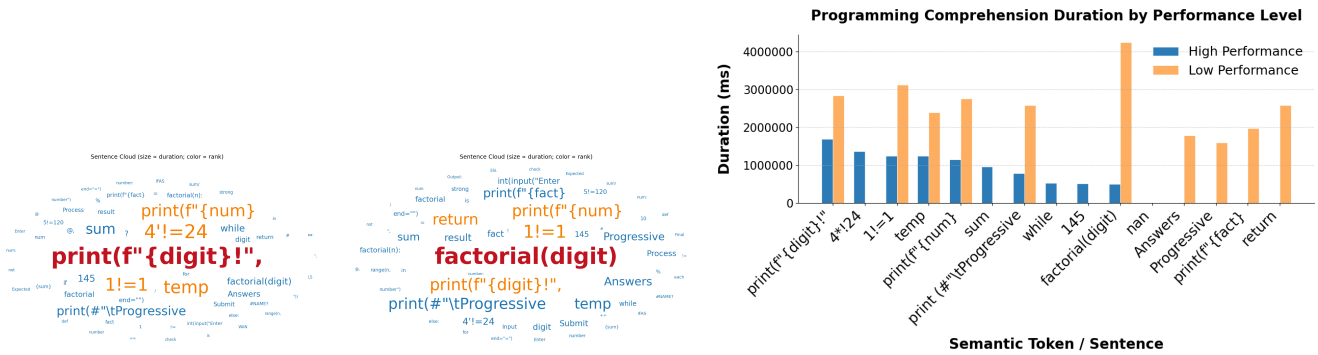
For RQ2, the results revealed that there are differences in how a programmer's visual attention is allocated to different tokens in terms of their semantics. The results also revealed that low-performing programmers tend to allocate more visual attention to output statements, literals, and result-oriented tokens. On the other hand, the results of the high-performing programmers showed more focus on structurally significant elements, such as identifiers, operators, and control structures. This implies a greater cognitive focus on program logic and structure. These findings are consistent with several existing literature [Cheng et al. 2024; Jessup et al. 2021; Obaidellah and Haek 2018]. Collectively, these results indicate the effectiveness of token-level analysis for the identification of important cognitive strategies during program comprehension.

With regards to RQ3, the results of the fixation frequency and fixation duration analysis showed significant differences between the performance levels of the programmers. As shown in the radar charts, the low-performing programmers showed significantly higher levels of visual attention to most of the token types, especially for the output and literal types. This implies more effort and less efficient cognitive strategies for program comprehension. Besides, the results of the present study showed the effectiveness of the proposed OCR-based framework for the extraction of cognitive attention maps. These results showed significant differences between the cognitive strategies of the novices and the more skilled programmers. Hence, imply the effectiveness of cognitive attention maps as a cognitive process for the development of gaze-based educational technologies and adaptive systems.



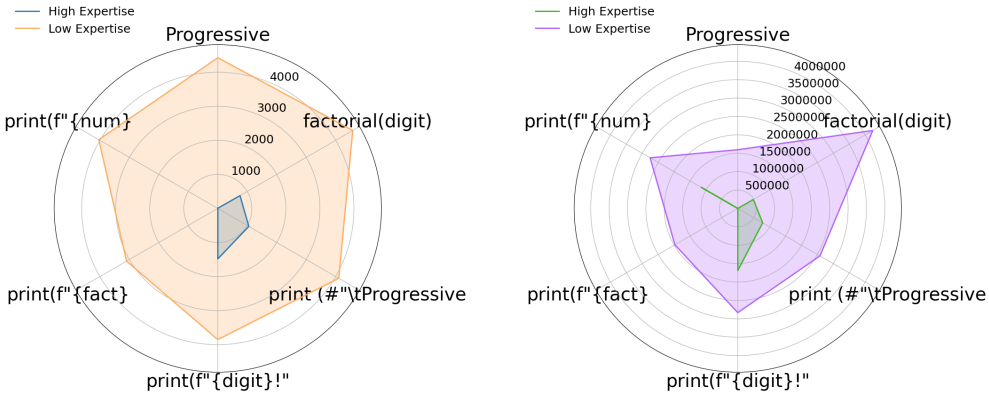
(a) High Performer (b) Low Performer (c) Token Frequency Distribution (High vs. Low)

Figure 3: Semantic Token Frequency Distribution by Performance Level



(a) High Performer (b) Low Performer (c) Token Duration Distribution (High vs. Low)

Figure 4: Semantic Token Duration Comparison Across Performance Level



(a) Frequency (b) Duration

Figure 5: Radar Chart Comparison of Fixation Frequency and Duration by Performance Level

8 Conclusion

This paper proposed a novel framework for the generation of CAMs based on the identification of AOIs via OCR-based techniques and semantic tokenization. Our experiments with varying levels of performance demonstrated the technical viability and the capacity of the framework to identify subtle differences in the way programmers process code. For example, the progression from the literal level and the focus on outputs among novices to the logic level

and the focus on control structures among experts speaks to the potential of this framework to inform the design of models that capture the progression of program comprehension. Future work will seek to improve the alignment between the OCR-based AOIs and the gaze data to address the issue of dynamic code and the use of a range of programming languages. With the scalability of this automated process, this paper offers vast possibilities of conducting ET research at a wider level in computer science education. In the final analysis, this framework presents the prospect of designing intelligent computer programs that are capable of offering real-time support based on the unique cognitive focus of the learner.

Acknowledgments

The authors wish to thank the participants of this study.

References

Magdalena Andrzejewska and Paweł Kotoniak. 2020. Development of Program Comprehension Skills by Novice Programmers – Longitudinal Eye Tracking Studies. *Informatics in Education* 19, 4 (Dec. 2020), 521–541. doi:10.15388/infedu.2020.23

Joshua Behler, Praxis Weston, Drew T. Guarnera, Bonita Sharif, and Jonathan I. Maletic. 2023. iTrace-Toolkit: A Pipeline for Analyzing Eye-Tracking Data of Software Engineering Studies. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, Melbourne, Australia, 46–50. doi:10.1109/ICSE-Companion58688.2023.00022

Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, Florence, Italy, 255–265. doi:10.1109/ICPC.2015.36

Teresa Busjahn and Carsten Schulte. 2013. The use of code reading in teaching programming. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*. ACM, Koli Finland, 3–11. doi:10.1145/2526968.2526969

Gary Cheng, Di Zou, Haoran Xie, and Fu Lee Wang. 2024. Exploring differences in self-regulated learning strategy use between high- and low-performing students in introductory programming: An analysis of eye-tracking and retrospective think-aloud data from program comprehension. *Computers & Education* 208 (Jan. 2024), 104948. doi:10.1016/j.compedu.2023.104948

Sarah Jessup, Sasha M. Willis, Gene Alarcon, and Michael Lee. 2021. Using Eye-Tracking Data to Compare Differences in Code Comprehension and Code Perceptions between Expert and Novice Programmers. doi:10.24251/HICSS.2021.013

Zan-Xia Jin, Mike Zheng Shou, Fang Zhou, Satoshi Tsutsui, Jingyan Qin, and Xu-Cheng Yin. 2022. From Token to Word: OCR Token Evolution via Contrastive Learning and Semantic Matching for Text-VQA. In *Proceedings of the 30th ACM International Conference on Multimedia*. ACM, Lisboa Portugal, 4564–4572. doi:10.1145/3503161.3547977

Niloofar Mansoor, Cole S. Peterson, Michael D. Dodd, and Bonita Sharif. 2024. Assessing the Effect of Programming Language and Task Type on Eye Movements of Computer Science Students. *ACM Transactions on Computing Education* 24, 1 (March 2024), 1–38. doi:10.1145/3632530

Ian McChesney and Raymond Bond. 2019. Eye tracking analysis of computer program comprehension in programmers with dyslexia. *Empirical Software Engineering* 24, 3 (June 2019), 1109–1154. doi:10.1007/s10664-018-9649-y

Unaizah Obaidellah and Mohammed Al Haek. 2018. Evaluating gender difference on algorithmic problems using eye-tracker. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications (ETRA '18)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/3204493.3204537

Konstantin Valentinovich Ryabinin, Svetlana Vladimirovna Alexeeva, and Tatiana Evgenievna Petrova. 2022. Automatic Areas of Interest Detector for Mobile Eye Trackers. In *Proceedings of the 32nd International Conference on Computer Graphics and Vision*. Keldysh Institute of Applied Mathematics, 228–239. doi:10.20948/graphicon-2022-228-239

Zohreh Sharafi, Timothy Shaffer, Bonita Sharif, and Yann-Gael Gueheneuc. 2015. Eye-Tracking Metrics in Software Engineering. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, New Delhi, 96–103. doi:10.1109/APSEC.2015.53

Wudao Yang, Zubair Ahsan, M. A. Rasel, and Unaizah Obaidellah. 2025a. Visualizing Novice Programmers' Attention Distribution using DBSCAN Clustering and Sankey Diagrams. In *Proceedings of the 2025 Symposium on Eye Tracking Research and Applications*. ACM, Tokyo Japan, 1–7. doi:10.1145/3715669.3725866

Wudao Yang, Seng Yue Wong, and Unaizah Obaidellah. 2025b. Automated token-level AOIs using OCR for predicting novice programmers' visual attention. *International Journal of Modern Physics C* (Dec. 2025), 2542023. doi:10.1142/S0129183125420239

Yifan Zhang, Jiliang Li, Zachary Karas, Aakash Bansal, Toby Jia-Jun Li, Collin McMillan, Kevin Leach, and Yu Huang. 2024. EyeTrans: Merging Human and Machine Attention for Neural Code Summarization. *Proceedings of the ACM on Software Engineering* 1, FSE (July 2024), 115–136. doi:10.1145/3643732