# PosEyeDOM: Lightweight Positional Copilot Suggestion Logger for Eye-Tracking Applications using the DOM

Tarek Alakmeh
tarek.alakmeh@uzh.ch
University of Zurich
Zurich, Switzerland

Sarah D'Angelo
sdangelo@google.com
Google
Seattle, USA

Thomas Fritz
fritz@ifi.uzh.ch
University of Zurich
Zurich, Switzerland

## Abstract

Understanding the impact of GenAI code suggestions on developer productivity requires studying how developers actually interact with them. Yet, inline suggestions from assistants such as GitHub Copilot appear and disappear quickly, shift with scrolling, and range from single-line completions to multi-line overlays. Capturing interactions is especially difficult for eye-tracking, which depends on precise positional data and timing. Existing tools offer limited support, as these transient overlays may disappear without ever being integrated into the source code. PosEyeDOM addresses this gap with a lightweight Chrome extension for web-based VS Code environments such as GitHub Codespaces. By leveraging the Document Object Model (DOM), it continuously logs suggestion geometry, content, and timing in real-time, infers acceptance outcomes, and aggregates them into robust and labeled Areas of Interest (AOIs). It exports ready-to-use `.ias` files compatible with SR Research DataViewer and JSON dictionaries for custom pipelines, enabling scalable, suggestion-aware eye-tracking studies that were previously impractical. The extension further provides a live preview, remote export, and synchronization with external eye-tracking software, offering an all-in-one, plug-and-play workflow for empirical research on AI-assisted programming.

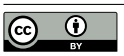
## CCS Concepts

• **Software and its engineering** → **Empirical software engineering**; • **Human-centered computing** → *Interaction techniques.*

## Keywords

Tracking Inline Suggestions, Dynamic Area of Interest, AOI, IA, Eye-tracking, GitHub Copilot, DOM

**ACM Reference Format:**
Tarek Alakmeh, Sarah D'Angelo, and Thomas Fritz. 2026. PosEyeDOM: Lightweight Positional Copilot Suggestion Logger for Eye-Tracking Applications using the DOM. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3774748.3787608




*ICSE-Companion '26, Rio de Janeiro, Brazil*

ACM ISBN 979-8-4007-2296-7/2026/04
https://doi.org/10.1145/3774748.3787608

## 1 Introduction

GenAI code completion has changed how developers write code, yet the evidence on its benefits remains mixed: studies report improvements in some tasks [6, 12], little or no effect in others [9, 15], and even negative outcomes driven by time spent debugging or fixing issues [5]. Perceived productivity can also diverge from objective gains [19]. This inconsistency suggests that apparent benefits may be outweighed by the costs of evaluating and correcting suggestions, underscoring the need for more fine-grained, in-situ analysis of how developers engage with GenAI suggestions inside Integrated Development Environments (IDEs).

Eye-tracking offers a powerful way to capture this engagement, but current methods are ill-suited for dynamic, in-editor suggestions. To rigorously analyze how suggestions are viewed, we need to know when and where they appear on screen, how long they remain visible, and whether they are ultimately accepted or rejected. Static AOIs or manual annotation cannot keep pace with suggestions that appear and disappear quickly, shift with scrolling, or split across multiple lines. Existing eye-tracking tools for software engineering, such as iTrace [11, 13] or CognitIDE [16], can map gaze to source code but cannot track suggestion overlays that never become part of the underlying code. This limitation has likely prevented more extensive, suggestion-aware eye-tracking studies: prior work either relied on precomputed stimuli or did not account for whether and how long suggestions were looked at [2, 17].

We created PosEyeDOM to alleviate current pain points by eliminating the need for manual AOI annotation, adding suggestion-aware tracking missing from existing tools, and enabling scalable, fine-grained analyses of how developers engage with AI code suggestions in IDEs such as VS Code. The tool and a demo video are publicly available[1].

## 2 Tool Functionality

PosEyeDOM provides an all-in-one solution for researchers who want to capture, inspect, and analyze Copilot suggestions in real-world coding environments. The extension is lightweight, easy to install, and designed to minimize setup overhead while delivering ready-to-use outputs for eye-tracking analysis. It bridges a common gap in empirical research by automatically transforming dynamic suggestion overlays into stable, exportable data. The following paragraphs introduce the main functionality and highlight how the tool alleviates pain points that researchers would otherwise have to overcome, such as manually drawing dynamic AOIs, aligning timestamps across systems, or reconstructing acceptance outcomes post-hoc.

[1] https://poseyedom.hasel.dev/ or [4] which provide both the source code and a README with installation instructions.

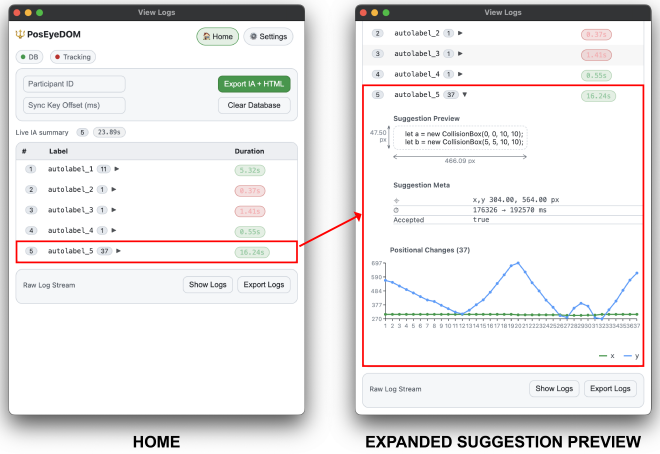



**Figure 1: User interface of `PosEyeDOM` extension: The home screen (left) enables the user to directly export ready-to-use Interest Areas (IA) and HTML Content dictionary files, informs the user about currently tracked suggestions in a real-time live preview, and offers inspection and export of the raw log stream. When clicking on a suggestion, the expanded preview opens (right) with a content preview of the suggestion, its dimensions, start and end time, and whether it was accepted. If the user scrolled while the suggestion was visible, a line chart with the positional changes is rendered.**

*Positional Tracking and Suggestion Identification.* PosEyeDOM continuously tracks each suggestion's geometry across time and scrolling events. When a suggestion moves due to user scrolling, the tool generates separate interest areas containing the new `x,y` positions with systematic suffixes (e.g., `autolabel_1_1`, `autolabel_1_2`, `autolabel_1_3`) to indicate which interest areas belong to the same suggestion. This automatic labeling ensures that researchers can reliably map eye fixations to dynamically shifting suggestions.

*Built-In Interest-Area Extender.* To improve robustness in eye-tracking analyses, PosEyeDOM includes an interest-area extender. Researchers can configure horizontal and vertical error margins that expand the bounding boxes of tracked interest areas. This compensates for potential calibration offsets, drifts, or other tracking inaccuracies in the eye tracker, ensuring that fixations near the edges of suggestions are still counted within the corresponding interest area.

*Suggestion Acceptance Tracking.* The tool automatically detects whether a suggestion has been accepted, independent of the mode of acceptance. Whether the developer presses the `TAB` key, clicks the acceptance button, or uses another interaction, the acceptance or rejection event is captured, associated with the suggestion, and logged.

*Export Ready-To-Use Dynamic Interest Area Files.* PosEyeDOM produces export files in the `.ias` format that can be directly imported into SR Research DataViewer or used in custom analysis pipelines. These dynamic interest areas map each suggestion both temporally (covering the time interval during which the suggestion was visible) and spatially (covering the full width and height of the suggestion's on-screen bounding box). This enables researchers to bypass manual AOI drawing and immediately integrate accurate suggestion-aware regions into their eye-tracking analysis.

*Live Preview of Suggestions during Experiment.* The extension provides a real-time live preview of all tracked and logged suggestions, including their dimensions, timing, and acceptance status (Figure 1). This preview can be detached into a separate browser window, allowing researchers to monitor experiment progress and track quality on a secondary display while participants work on the main task. This live monitoring ensures transparency and quality control during data collection.

*Timestamp Synchronization with External Eye-Tracking Software.* For seamless integration with external eye-tracking stimuli presentation software such as SR Research WebLink, PosEyeDOM supports timestamp synchronization. Researchers can define a synchronization key (default: `s`) that, when pressed for the first time during setup, is logged both in the external software (e.g., WebLink) and in PosEyeDOM. The researcher can then provide the reference timestamp of the external software's session log to the extension interface (*Sync Key Offset (ms)* field in Figure 1), which will apply offset-correction to the exported `.ias` files, enabling precise temporal alignment between eye-tracking data and suggestion events.

*Remote Export.* Beyond local storage, PosEyeDOM supports remote export functionality. Researchers can provide a REST endpoint, and the extension will push raw log stream updates in ten-second intervals. This enables external monitoring of suggestion data during live experiments, such as from a separate computer or monitoring dashboard.

*Additional Settings.* The extension includes a range of customizable settings to accommodate various experimental contexts. Researchers can configure custom vertical offsets for browser windows, toggle light or dark mode, and adjust CSS selectors used for detecting suggestions. These options ensure that PosEyeDOM remains flexible even if the underlying VS Code web interface changes over time.

## 3 Implementation

PosEyeDOM is implemented as a lightweight Chrome extension written in JavaScript. It follows a simple but robust pipeline that transforms raw DOM changes into analyzable suggestion data, as illustrated in Figure 2. First, the extension leverages Chrome's built-in `MutationObserver` to capture every Copilot suggestion element on `Github Codespaces` as soon as it appears or disappears, recording its HTML content, geometry, and timestamp. Second, these events are written into a raw log stream, which provides the full timeline of suggestion-related DOM activity, including synchronization key presses and acceptance checks. Third, a grouping algorithm processes this stream to merge fragmented elements into coherent suggestion objects, assigns labels, and tracks positional changes across time, while attaching acceptance or rejection outcomes. Finally, these structured groups are transformed into export-ready formats: `.ias` files that define dynamic interest areas for SR Research DataViewer, and JSON dictionaries containing content and metadata for custom pipelines. To increase robustness,

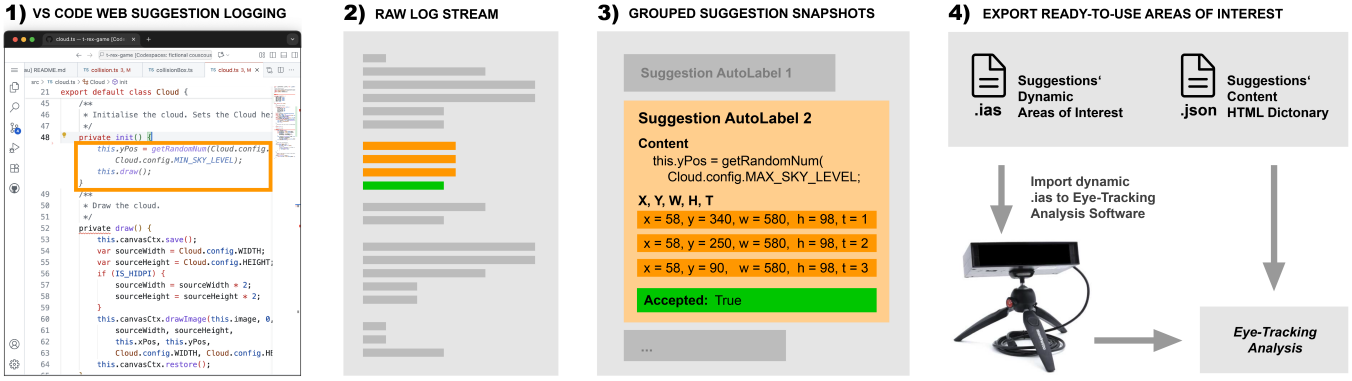



**Figure 2: Procedure of PosEyeDOM suggestion logging: (1) VS Code web suggestion logging, (2) raw log stream, (3) grouped suggestion snapshots, (4) export-ready dynamic areas of interest (.ias) and content dictionary (.json) for eye-tracking analysis.**

transparency, and verifiability, the pipeline explicitly separates the low-level logging of raw events from the higher-level grouping and matching processes that build upon them. The following paragraphs provide details for each of the four steps.

*(1) Raw Suggestion Elements Logging through MutationObserver on DOM.* PosEyeDOM operates in web-based VS Code environments such as `Github Codespaces` and targets Copilot inline suggestions. It relies on Chrome's built-in `MutationObserver` API to listen for document changes in the DOM whenever suggestion elements appear, update, or disappear. By using predefined CSS selectors (which can be customized in the extension settings), the tool identifies all `DIV` elements related to suggestions. Each such element, along with its HTML content, positional geometry (`x`, `y`, width, height), and timestamp, is written to a raw log stream as a snapshot of the on-screen state.

*(2) Raw Log Stream.* The raw log stream constitutes the base event timeline of the extension. It contains dozens of individual elements for each suggestion, as Copilot does not render suggestions as single `DIV`s but rather as multiple fragments reflecting syntax highlighting or breaks inside tokens. Alongside these visual fragments, the stream logs synchronization key events and acceptance/rejection events. Acceptance and rejection are determined immediately after a suggestion disappears by comparing the previously displayed suggestion content against the actual code now present at that location. If the code matches, the suggestion is marked as accepted; otherwise, it is logged as rejected. This procedure is mode-agnostic, working independently of whether acceptance occurred via `TAB`, mouse click, or other means. At this stage, the stream remains a low-level record of suggestion-related DOM events.

*(3) Grouping Raw Logs to Coherent Suggestion Logs.* To make the raw stream analyzable, the extension includes a grouping algorithm that merges the fragmented HTML elements of each suggestion into a coherent block. It then tracks this block across subsequent log entries, assigning a root label such as `autolabel_1`. If the suggestion reappears at new screen coordinates (e.g., due to scrolling), the algorithm generates labeled suffixes to indicate positional continuity, such as `autolabel_1_1`, `autolabel_1_2`, `autolabel_1_3`. This ensures that movements or splits of the same suggestion are preserved as part of one logical group. Acceptance or rejection events are then associated with the grouped suggestion object, yielding a structured representation of each suggestion's full life cycle.

*(4) Generating `.ias` File for Export.* Once coherent suggestion groups are available, exporting them becomes a simple transformation. PosEyeDOM generates ready-to-use `.ias` files for SR Research DataViewer, mapping each suggestion's spatial and temporal presence on screen into dynamic interest areas. Additionally, a compact JSON dictionary of suggestion content and metadata is produced, supporting custom analyses or integration into other pipelines. These exports allow researchers to directly incorporate suggestion-aware AOIs into eye-tracking studies without manual region definition or post-hoc alignment.

The immediate benefit is that researchers no longer need to hand-draw dynamic rectangles for overlays or to instrument editor internals. The tool is a single extension that runs without code changes. It provides a transparent data trail: raw logs describe exactly what was seen on screen and when, the interest areas reconstruct the conceptual suggestions as they moved, and the mapping preserves a compact content snapshot with the acceptance outcome. Researchers can use PosEyeDOM to conduct in-depth eye-tracking investigations of developers interacting with Copilot suggestions.

## 4 Evaluation

The accuracy of the tool has been verified through extensive manual testing. It was successfully used in a controlled lab study [3], where robust AOI tracking and grouping were ensured prior to conducting the study. To further validate correctness, we analyzed screen recordings from four participant sessions (each lasting 45 minutes and each containing between 100–180 suggestions). For the suggestions observed in the recordings, we verified that they were captured by the extension, checked whether the logged duration matched its on-screen visibility, and confirmed that positional changes were correctly reflected by mapping the extension's exported data onto the screen recordings. Across these tests, PosEyeDOM consistently tracked suggestions and matched acceptance outcomes. The only notable limitation was that suggestions visible

for less than 200 ms may be missed due to DOM sampling throttling; this trade-off and its mitigation are discussed in Section 6.

## 5 Related Work

Software engineering studies that use eye-tracking face a persistent challenge in defining Areas of Interest (AOIs). Existing approaches rely on defining static regions or tools tied to static code, making them unsuitable for transient overlays such as Copilot suggestions. Among existing support, tools such as iTrace [11, 13] or CognitIDE [16] enable integration with IDEs to map gaze to code tokens. However, none of these systems are able to track Copilot code suggestions that appear, move, and disappear without ever becoming part of the underlying code unless accepted. Outside software engineering, toolkits for dynamic AOIs exist [1, 8], but they remain generic and require manual setup that is impractical for the fast-changing overlays used by code assistants. Prior work has also synthesized methodologies for eye-tracking in programming [10, 14] and investigated the impact of AI coding assistants [7, 12, 18, 19], yet none provide infrastructure to link gaze to the precise appearance, movement, and acceptance of inline suggestions. To our knowledge, PosEyeDOM is the first tool to automatically generate suggestion-aware AOIs with pixel-accurate geometry and acceptance outcomes by leveraging the DOM overlay mechanics of the editor, addressing a methodological gap that has so far limited more extensive eye-tracking studies of AI-assisted programming.

## 6 Limitations and Threats to Validity

Future changes of the underlying `Github Codespaces` user interface may influence or break automatic suggestion tracking. We mitigate this risk with configurable CSS selectors, offsets, and real-time, easily accessible visual checks of logged data. Geometry can vary across devices due to the influence of zoom and DPI on eye-tracking accuracy; we expose vertical offsets and allow for IA error margins to improve the accuracy of subsequent analysis. Throttling of DOM sampling mitigates performance issues of the browsers but may miss suggestions that appear very quickly (<200ms by default); the throttling amount is adjustable through the tool settings. Acceptance inference can be ambiguous when suggestions replicate entire lines of code that are identical to pre-existing lines of code below the point where the suggestion appeared; we normalize text, search within a local window, and record both decisions and evidence for verification. The tool currently only works for web-based VS Code as used by `Github Codespaces`.

## 7 Conclusion and Future Directions

PosEyeDOM lowers a key barrier for eye-tracking research on AI-assisted programming by automatically turning Copilot suggestions into analyzable dynamic AOIs. It eliminates the need for manual AOI drawing, provides a transparent data trail from raw DOM events to export-ready files, and enables analyses that were previously impractical at scale. Future directions include supporting additional export formats, such as those compatible with Tobii eye trackers, and extending compatibility beyond `Github Codespaces` to other coding environments. These directions would broaden the applicability of PosEyeDOM and further facilitate more extensive, cross-platform studies of AI-assisted development.

## References


[1] 2025. *How to Work with Dynamic Areas of Interest.* https://connect.tobii.com/s/article/how-to-work-with-dynamic-areas-of-interest Accessed Sep 29, 2025.

[2] Naser Al Madi. 2023. How Readable is Model-generated Code? Examining Readability and Visual Inspection of GitHub Copilot. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22).* Association for Computing Machinery, New York, NY, USA, 1–5.

[3] Tarek Alakmeh, Sarah D'Angelo, and Thomas Fritz. 2026. An Eye for AI: Eye-Tracking the Micro-Interruptions of GenAI Code Suggestions. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26).* ACM, Rio de Janeiro, Brazil, 1–12. doi:10.1145/3744916.3773132

[4] Tarek Alakmeh, Sarah D'Angelo, and Thomas Fritz. 2026. *PosEyeDOM: Lightweight Positional Copilot Suggestion Logger for Eye-Tracking Applications using the DOM (Replication Package).* doi:10.17605/OSF.IO/9NM2S

[5] Joel Becker, Nate Rush, Beth Barnes, and David Rein. 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. (July 2025).

[6] Zheyuan (Kevin) Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. 2025. The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers. doi:10.2139/ssrn.4945566

[7] Omer Dunay, Daniel Cheng, Adam Tait, Parth Thakkar, Peter C. Rigby, Andy Chiu, Imad Ahmad, Arun Ganesan, Chandra Maddila, Vijayaraghavan Murali, Ali Tayyebi, and Nachiappan Nagappan. 2024. Multi-line AI-Assisted Code Authoring. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE 2024).* Association for Computing Machinery, New York, NY, USA, 150–160. doi:10.1145/3663529.3663836

[8] Y. Faraji and colleagues. 2022. A Toolkit for Wide-Screen Dynamic Area of Interest Measurements with Pupil Core. *Behavior Research Methods (Toolkit note, open-source)* (2022). https://pmc.ncbi.nlm.nih.gov/articles/PMC10616213/

[9] Trevor Fitzpatrick, Seamus Kelly, Patrick Carey, David Walsh, and Ruairi Nugent. 2025. Assessing Generative AI value in a public sector context: evidence from a field experiment. doi:10.48550/arXiv.2502.09479 arXiv:2502.09479 [q-fin].

[10] L. Grabinger and colleagues. 2024. On Eye Tracking in Software Engineering. *SN Computer Science* (2024).

[11] Drew T. Guarnera, Corey A. Bryant, Ashwin Mishra, Jonathan I. Maletic, and Bonita Sharif. 2018. iTrace: eye tracking infrastructure for development environments. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications* (Warsaw, Poland) *(ETRA '18).* Association for Computing Machinery, New York, NY, USA, Article 105, 3 pages. doi:10.1145/3204493.3208343

[12] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. (Feb. 2023). doi:10.48550/arXiv.2302.06590 arXiv:2302.06590 [cs].

[13] Timothy R. Shaffer, Jenna L. Wise, Braden M. Walters, Sebastian C. Müller, Michael Falcone, and Bonita Sharif. 2015. iTrace: enabling eye tracking on software artifacts within the IDE to support software engineering tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015).* Association for Computing Machinery, New York, NY, USA, 954–957. doi:10.1145/2786805.2803188

[14] Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. 2020. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering* 25, 5 (Sept. 2020), 3128–3174. doi:10.1007/s10664-020-09829-4

[15] Fangchen Song, Ashish Agarwal, and Wen Wen. 2025. The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot. doi:10.48550/arXiv.2410.02091 arXiv:2410.02091 [cs].

[16] Fabian Stolp, Malte Stellmacher, and Bert Arnrich. 2024. CognitIDE: An IDE Plugin for Mapping Physiological Measurements to Source Code. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering* (Porto de Galinhas, Brazil) *(FSE 2024).* Association for Computing Machinery, New York, NY, USA, 592–596. doi:10.1145/3663529.3663805

[17] Ningzhi Tang, Meng Chen, Zheng Ning, Aakash Bansal, Yu Huang, Collin McMillan, and Toby Jia-Jun Li. 2024. A Study on Developer Behaviors for Validating and Repairing LLM-Generated Code Using Eye Tracking and IDE Actions. doi:10.48550/arXiv.2405.16081 arXiv:2405.16081 [cs].

[18] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts.* ACM, New Orleans LA USA, 1–7.

[19] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (MAPS 2022).* Association for Computing Machinery, New York, NY, USA, 21–29. doi:10.1145/3520312.3534864