PyWIB: A Python Library for a Multi-Modal Approach to Web Interaction Behavior Analysis

Guillermo Dylan Carvajal-Aza[1], Alejandro Alvarez-Varela[1], Javier De Andres[2], Martin Gonzalez-Rodriguez[1], Daniel Fernandez-Lanvin[1], Mercedes Paino[3]

[1] Dept. of Computer Science.
[2] Dept. of Accounting.
[3] Dept. of Psychology.

University of Oviedo, Oviedo, Spain
Email: {carvajalguillermo, avarela, jdandres, martin, dflanvin, mpaino}@uniovi.es

Abstract— **This paper addresses the need for efficient tools in multi-modal web interaction analysis, as existing frameworks often lack integration for diverse data streams. PyWIB (Python Web Interaction Behavior) is a new Python library designed for extracting and analyzing metrics from user interactions with web pages. It aims to streamline the analysis of user interaction data in the field of Human-Computer Interaction, providing researchers with a comprehensive set of methods to process event logs and compute metrics essential for such research. PyWIB is open-source and available on GitHub. The results demonstrate that PyWIB effectively unifies mouse-tracking and keystroke dynamics into a single processing pipeline.**

Keywords-Human-Computer Interaction; interaction analysis; web interaction patterns; web behavior; python.

I. INTRODUCTION

The analysis of web interaction has been a cornerstone of Human-Computer Interaction (HCI) research for over two decades, evolving from early clickstream discovery [1] to the automated usability evaluation frameworks that characterize modern research [2]. Among the various techniques used to analyze the interaction of a user with a web page, mouse-tracking and keystroke dynamics [3] are the two most commonly used methods for obtaining insights into the user's specific characteristics.

These two types of interaction analysis have been widely conducted in HCI to infer several aspects of the user's behavior and interaction with web and computer interfaces. Both dynamics have been proven useful in different fields of human behavior. Mouse dynamics have been employed in the analysis of human behavioral patterns and factors [4], cognitive and physical conditions affecting the user [5], [6], [7], in user identification [8] and in studies aiming to measure the influence of age or gender in common computer tasks [9], [10]. In contrast, keystroke dynamics have been widely used for the recognition of users' emotions and personalities [5], as well as user authentication, behavioral biometrics, and affective computing [11].

These types of analysis require a lengthy implementation process, from data collection to its processing; the data generated from a mouse-tracking session of 60 seconds can record up to 900 entries (recording mouse events with intervals of 100 milliseconds), making it difficult to scale and manage without appropriate computational methods.

As data science gains importance within the field of HCI and scientific tools become increasingly accessible, there is a clear need for a platform that provides a straightforward way to extract metrics from datasets. Furthermore, a well-documented repository of user interaction metrics is essential to help researchers identify which ones are best suited for their specific studies.

While there already exist other software libraries dedicated to extracting metrics from users' data (e.g., Mousetrap) [12], to the extent of our knowledge, none have been implemented in a language as versatile as Python, which allows the creation of Application Programming Interfaces (APIs) that can be deployed in web environments and integrated into complex data science pipelines. Consequently, PyWIB facilitates a more efficient research workflow.

Additionally, visualization tools for mouse and keystroke events are not often available to everyone, as the existing tools tend to hide these features behind paywalls after their free trial (such as Hotjar [13], Mouseflow [14] or Microsoft Clarity [15]). These visualization tools are essential in some cases for the interpretation of users' results or analyses carried out using computer vision techniques [4], forcing researchers to implement their own solution to this problem.

To address the existing software limitations, PyWIB has been developed by the Human Communication Interaction Research Group from the University of Oviedo as an open-source implementation available for download as a Python library. This library has been designed as a modular library, which provides the user with a set of functions, for which we have made available extensive documentation on both their technical and mathematical implementation, on the official documentation site [16].

By using this modular approach, we have created a library that can not only be easily expanded to incorporate new computations but can also be used by other programs through the creation of an API, enhancing interoperability.

To demonstrate its utility, PyWIB facilitates representative use cases in common HCI research contexts. First, it supports usability and interaction efficiency studies, where mouse-trajectory metrics quantify motor precision during pointing or drag-and-drop tasks [10], allowing researchers to objectively compare interaction strategies

across different interface designs. Additionally, the library supports research on cognitive and affective states. Keystroke dynamics can be analyzed to investigate cognitive load, emotional patterns, or error-correction behavior, providing insights into user performance beyond basic execution times [5], [17].

The remainder of this paper is structured as follows. Section II describes metric calculation and validation. Section III details the software design and framework. Section IV explains the segmentation process. Section V presents visualization features. Finally, Section VI discusses limitations, concludes the paper, and outlines future work.

II. METRICS CALCULATION AND VALIDATION

One of the main problems when dealing with a library that aims to cover the computation of the most common metrics in HCI research is the validation of those metrics. For this reason, PyWIB has been developed considering the most relevant metrics used in research works, which have been proven to be representative of user behavior in different contexts. The following metrics are available for computation using tracking datasets in the current PyWIB release:

A. **Velocity:** Rate of change of position over time.
B. **Acceleration:** Rate of change of the velocity over time.
C. **Jerkiness:** Rate of change of the acceleration over time.
D. **Path:** Total distance traveled.
E. **Area Under the Curve (AUC) Ratio:** Deviation of the user's trajectory with respect to the optimal path.
F. **Maximum Absolute Deviation (MAD):** Maximum perpendicular offset from the ideal straight-line trajectory.
G. **Average Absolute Deviation (AAD):** Mean perpendicular displacement from the optimal straight-line path.
H. **Mouse:** Click counts and activation-to-release deviations.
   a. Number of clicks
   b. Click Slip
I. **Keyboard:** Typing dynamics and error-correction patterns.
   a. Typing Duration
   b. Typing Speed
   c. Backspace Usage
J. **Time:** Execution, active movement, and pause count.
   a. Execution Time
   b. Movement Time
   c. Number of Pauses

A. Velocity

Velocity is the rate of change of position of the user's cursor with respect to the observed time of the different points that mark the user's interaction events, measured in pixels per second [18].

The applied formula for this calculation is as follows:

$$v_i = \frac{\sqrt{(x_i + x_{i-1})^2 + (y_i + y_{i-1})^2}}{t_i + t_{i-1}} \quad (1)$$

where (x_i, y_i) are the coordinates and t_i is the timestamp on the analyzed point.

B. Acceleration

Acceleration is calculated as the change of velocity with respect to time [18]. For this metric, it is imperative to compute the velocity of each session trace by applying the previous formula.

The applied acceleration formula is as follows:

$$a_i = \frac{v_i - v_{i+1}}{t_i - t_{i-1}} \quad (2)$$

where v_i is the velocity at the examined time and t_i is the value of the timestamp for the given point in the user's interaction.

C. Jerkiness

Jerkiness is a metric derived from acceleration, computed as the change in acceleration per unit of time [19]. This metric, while not widely used, has been applied in studies to identify bursts of speed and identify irregularities in motor control, reflecting the degree of movement fluidity or hesitation.

$$j_i = \frac{a_i - a_{i+1}}{t_i - t_{i-1}} \quad (3)$$

D. Path

The path is the total distance travelled, a key metric computed as the total distance between consecutive points in the user interaction [4], [6], [18], [20], calculated as:

$$\sum_{i=0}^{n} (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \quad (4)$$

E. Area Under the Curve

The AUC is a metric that quantifies the deviation of the user's trajectory with respect to the optimal path from the start point to the end point of his interaction. This metric has been widely used in HCI when analyzing users' trajectory [6], [18]. PyWIB provides two approaches for obtaining this metric. Given the presence of loops and backtracking in the user's trajectory, these approaches allow researchers to choose whether to penalize motor inefficiencies and user control by accounting for such behaviors or to mitigate their impact through a geometrical simplification of the trajectory.

F. Maximum Absolute Deviation

The MAD is a metric calculated as the maximum perpendicular distance from any point on the user's actual path to the straight line that connects the starting and ending points of his movement (that is, the optimal trajectory the user should follow) [4], [6], [18].

G. Average Absolute Deviation

The AAD consists of the average of the perpendicular distances from each point of the trajectory to the straight line connecting the start and end points of the movement (the optimal trajectory), as defined in [6].

H. Mouse

1) Number of Clicks

This metric consists of the total number of times an event of type EVENT_ON_CLICK is registered for the user's session or trace [21].

2) Click Slip

The click slip is a metric that measures the distance of click positions from the intended targets, as defined by [7], computed taking the initial point where an event of type EVENT_ON_MOUSE_DOWN takes place and the distance to its next event of type EVENT_ON_MOUSE_UP.

I. Keyboard

The metrics obtained from keyboard interaction have been widely used in various applications, including user authentication, behavioral biometrics, affective computing, and HCI studies [3], [5], [17], [22].

To the best of our knowledge, there is limited research on the use of keystroke dynamics and behavioral user patterns. The keystroke metrics here defined are derived from the works developed by Katerina and Nicolaos [4] and Khanna and Sasikumar [17], where the authors explore the use of keystroke dynamics in relation to the recognition of behavioral user attributes from their stroke patterns.

1) Typing Duration

A time-based metric for keystrokes that captures the time interval between an event of a key-press and an event of type key-release [4], [17] .

2) Typing Speed

This metric refers to the average number of typed Characters Per Minute (CPM) by a user during a session, defined and used in research by [17].

3) Backspace Usage

The number of times a user presses the backspace (or any deletion key) during a session or trace. This metric can be helpful to assess a negative emotional state and users' error-correction behavior as done by [17].

J. Time

1) Execution Time

Also referred to as "total time" or "session time", as it represents the overall duration of a user session from the first recorded event to the last, as described in [7]. It can be useful when filtering sessions based on their length if the task being studied has a defined expected minimum duration. It is important to note that this metric does not account for periods of inactivity or pauses within the session.

2) Movement Time

Defined by [7], this metric focuses on active movement events during a session, excluding idle times, and provides a more accurate representation of the time spent in motion by the user. It is useful when focusing on the time of the actual interaction, rather than filtering based on total session duration.

3) Number of Pauses

As the name suggests, the number of pauses refers to occurrences in which the user halts interaction for a minimum duration. This threshold is typically measured in milliseconds, commonly set between 100 and 200 milliseconds [7], [17], as these values represent the boundary between automatic motor execution and cognitive processing delays.

Regarding the correctness of metric computation, PyWIB has been designed with validation as a core principle. The implementation of all metrics follows their formal mathematical definitions as reported in the literature, and input data are systematically validated to ensure consistency and completeness prior to computation. Moreover, unit tests have been implemented for all metric modules using the pytest framework, covering both valid and invalid input scenarios to verify numerical correctness and appropriate error handling.

Not all metrics are equally valid for all experimental contexts. For example, metrics that are valid for analyzing mouse movements in a desktop web application may not be valid for analyzing touch interactions on mobile devices, as it could make no sense to analyze users speed if they can only tap the screen or drag and drop elements. Therefore, it is crucial to consider the context in which the metrics will be applied and to verify that their use is correct.

Moreover, the setup of an experiment itself can influence the validity of certain metrics [23], [24], which is why PyWIB encourages users to verify the correct use of the metrics they compute in their specific contexts and experimental setups.

III. SOFTWARE DESIGN, FRAMEWORK AND FEATURES

The PyWIB package has been built using a modular approach in Python 3.9, inspired by mathematical and data analysis libraries like NumPy and Matplotlib, which employ a hybrid approach with Object Orientation and Procedural programming aspects. Even if, at the start, our library has only found the need for describing modules, this does not mean it cannot lean towards a hybrid approach, following the necessities of the scientific community and the specific cases it tackles in the field of HCI.

The code also adopts a defensive programming paradigm [25], where all inputs are validated prior to processing. This ensures early detection of invalid states, such as missing columns or *null* DataFrames. Unit tests have also been implemented to verify the correct workflow of inputs using the Python testing framework *pytest*. In these tests, validation functions were tested against both valid and invalid inputs to ensure correct exception handling and error reporting, among

all metrics provided by the library. Together, the validation layer and unit testing framework ensure that the software behaves predictably under a wide range of conditions, reducing runtime errors and increasing reproducibility.

The PyWIB library has dependencies on the following packages, with a minimum requirement of their versions:

- Pandas' versions equal or greater than 1.3.0
- NumPy versions equal or greater than 1.21.0
- Matplotlib versions equal or greater than 3.5.0
- OpenCV-python versions equal or greater than 4.12.0.88
- Seaborn version equal or greater than 0.13.2

As for its current version (1.0.0), its modular approach allows for easy implementation of new modules and flexibility to be combined with deeper algorithms or new visualization techniques.

The structure of this library has been separated into packages based on use cases related to the tracking of user interaction. The main packages that compose the library are represented in Figure *1*, which presents a UML package diagram of the software framework. This separation of concerns improves maintainability, extensibility, and testability.

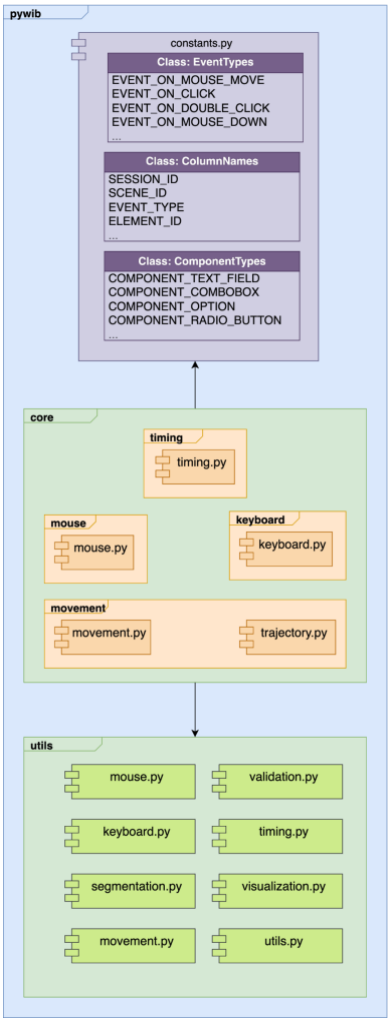

Figure 1. PyWIB Package Diagram.

These main packages can be described as follows:

- From PyWIB core package:
  - Keyboard: containing metrics related to keystroke analysis.
  - Mouse: containing metrics related to mouse clicking, such as number of clicks
  - Movement: containing metrics related to both the movement the user describes over the web interface and his described trajectory. The metrics included in this package are applicable to both mouse-tracking on computers and tracking over mobile and tablet devices.
  - Timing: containing metrics related to time, such as execution time and movement time (time the user spends moving through the web interface, without idle times).
- From PyWIB utils package:
  - Visualization: containing methods used to create visual representations of the user trajectory, heatmaps over his keystrokes, etc.
  - Segmentation: containing methods related to the segmentation of datasets into different traces.

The library also contains a package that lists a set of constants related to events, key codes, columns used to compute metrics and component types used during preprocessing and analysis. These constants have been set according to the ones produced by the Interaction Lab tool [26], used by our team to facilitate compatibility with the tool, which are intended to represent numerically all possible JavaScript events on computers and mobile devices.

IV. SEGMENTATION

One of the key elements of PyWIB is the analysis based on the segmentation of long user sessions into small movement traces. Given a Data Frame that contains the user's interaction, a trace is considered as a sequence of consecutive (non-stopped) mouse/touch movements between two non-movement events.

This type of segmentation has been widely employed in HCI research to obtain better insights into the users' movement and trajectories [7], [27].

This movement segmentation is implemented over two functions: *extract_trace* and *extract_traces_by_session*, both take a Data Frame as input and return a list or a dictionary containing each session as a key and the list of traces as a value.

These segmentations are not limited to metrics related to movement, as for the current version, there is also a method that extracts traces from keystroke interaction.

The method *extract_keystroke_traces_by_session* reduces a Data Frame to a dictionary that contains, as values, lists of segments of the user's interaction in which the interactions occur only through the keyboard. This method can be useful in cases where keystroke dynamics require analysis for speed or other time-based metrics.

V. VISUALIZATION

In its current version, PyWIB offers three types of visualization that can be created from the interaction data, by making use of the libraries seaborn and matplotlib.

Trace visualization, through method *visualize_trace*, creates an image using the user's movement to represent the movement of the mouse over the screen. This method can be used either with a full DataFrame, containing the entirety of the user's interaction, or with segmented traces, to analyze a concrete movement, as seen in Figure 2.

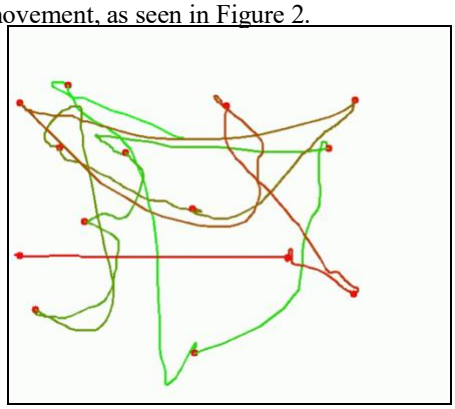

Figure 2. Visualization of the Mouse over the Screen.

Using a similar approach, *video_from_trace* method generates an mp4 video from the user's interaction using its timestamps to show his movement over time.

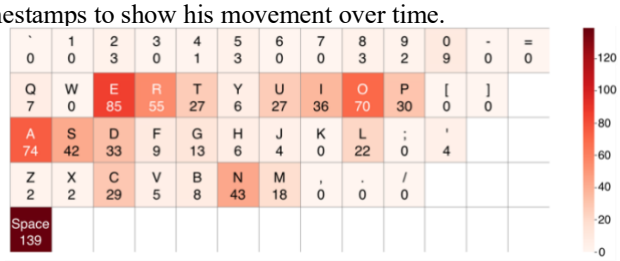

Figure 3. Heatmap of Keystrokes.

Finally, a keystroke heatmap is generated via the *keyboard_heatmap* function to visualize typing intensity across the layout. This visualization is exemplified in Figure 3, showcasing the spatial distribution of key presses.

VI. LIMITATIONS, CONCLUSION AND FUTURE WORK

In this paper, we have presented PyWIB, a novel Python library designed to support HCI researchers in the preprocessing and analysis of user interaction tracking data. Its modular architecture facilitates seamless integration with existing Python scripts, APIs, or other tools, making it a flexible and extensible solution that enables users to manipulate data according to their specific needs. Moreover, it offers a transparent and accessible alternative to proprietary software, providing useful visualization tools such as heatmaps and trace reconstruction without the constraints of paywalls. By adopting a defensive programming approach and offering standardized and validated metrics, PyWIB enhances reproducibility.

While PyWIB is intended to ease the workload of HCI researchers alongside their existing toolchains, its current implementation still leaves room for improvement. Specifically, further optimization is required to reduce computation time when handling large-scale datasets.

As PyWIB was made publicly available as an open-source project on GitHub in January 2026, increasing its recognition and adoption within the research community constitutes an important direction for future work. Planned efforts include dissemination through scientific publications and conference venues, as well as the development of example-driven documentation and research-oriented use cases to promote adoption by researchers and practitioners.

Future work will focus on the systematic evaluation of PyWIB to empirically validate its effectiveness. Planned studies will assess metric correctness, computational efficiency, and robustness using large-scale, real-world interaction datasets to demonstrate the library's practical utility in HCI research.

To directly address these computational limitations, future development of this software should primarily focus on optimizing performance, leveraging available hardware (e.g., GPUs), and integrating frameworks such as PySpark to enable real-time, large-scale data processing in a distributed environment using Python and Apache Spark. Looking ahead, there is also significant potential to further enhance PyWIB by expanding its computational capabilities and incorporating new metrics, such as fixation duration for eye-tracking integration, scroll-depth analysis, or touch-gesture complexity for mobile environments, as the needs of the scientific community grow. Additionally, adding multi-format export options (e.g., JSON and CSV) and basic statistical built-in functions, such as T-tests or ANOVA, for automated metric comparison, will ensure better compatibility and expanded analysis capabilities.

ACKNOWLEDGMENT

This research has been partially funded by the Carlos III Health Institute (reference FISS-23-PI23-00416) co-funded by the European Union, and by the Ministry of Science, Innovation and Universities and co-funded by the European Regional Development Fund (ERDF) under project PID2024-155586OB-I00 (MCIU/AEI/10.13039/501100011033). Additional support was provided by the Government of the Principality of Asturias through grant GRU-GIC-24-070.

REFERENCES

[1] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan, "Web usage mining: discovery and applications of usage patterns from Web data," *SIGKDD Explor Newsl*, vol. 1, no. 2, pp. 12–23, Jan. 2000, doi: 10.1145/846183.846188.

[2] M. Y. Ivory and M. A. Hearst, "The state of the art in automating usability evaluation of user interfaces," *ACM Comput Surv*, vol. 33, no. 4, pp. 470–516, Dec. 2001, doi: 10.1145/503112.503114.

[3] L. M. Vizer, L. Zhou, and A. Sears, "Automated stress detection using keystroke and linguistic features: An exploratory study," *Int. J. Hum.-Comput. Stud.*, vol. 67, no. 10, pp. 870–886, Oct. 2009, doi: 10.1016/j.ijhcs.2009.07.005.

[4] T. Katerina and P. Nicolaos, "Mouse behavioral patterns and keystroke dynamics in End-User Development: What can they tell us about users' behavioral attributes?," *Comput. Hum. Behav.*, vol. 83, pp. 288–305, Jun. 2018, doi: 10.1016/j.chb.2018.02.012.

[5] I. A. Khan, W.-P. Brinkman, N. Fine, and R. M. Hierons, "Measuring personality from keyboard and mouse use," in *Proceedings of the 15th European conference on Cognitive ergonomics: the ergonomics of cool interaction*, Funchal Portugal: ACM, Jan. 2008, pp. 1–8. doi: 10.1145/1473018.1473066.

[6] J. Rhim, J. Kim, and G. Gweon, "Using Geometric Features of Drag-and-Drop Trajectories to Understand Students' Learning," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI 2023)*, New York: Assoc Computing Machinery, 2023, p. 706. doi: 10.1145/3544548.3581143.

[7] V. Pandey, N. C. Khan, A. S. Gupta, and K. Z. Gajos, "Accuracy and Reliability of At-Home Quantification of Motor Impairments Using a Computer-Based Pointing Task with Children with Ataxia-Telangiectasia," *ACM Trans. Access. Comput.*, vol. 16, no. 1, pp. 1–25, Mar. 2023, doi: 10.1145/3581790.

[8] M. Karim and Md. Hasanuzzaman, "A Study on Mouse Movement Features to Identify User," *Sci. Res. J.*, vol. 08, no. 04, pp. 77–82, Apr. 2020, doi: 10.31364/SCIRJ/v8.i4.2020.P0420766.

[9] P. Kratky and D. Chuda, "Estimating Gender and Age of Web Page Visitors from the Way They Use Their Mouse," in *Proceedings of the 25th International Conference Companion on World Wide Web - WWW '16 Companion*, Montreal, Quebec, Canada: ACM Press, 2016, pp. 61–62. doi: 10.1145/2872518.2889384.

[10] B. Pariente-Martinez, M. Gonzalez-Rodriguez, D. Fernandez-Lanvin, and J. De Andres-Suarez, "Measuring the role of age in user performance during interaction with computers," *Univers. Access Inf. Soc.*, vol. 15, no. 2, pp. 237–247, Jun. 2016, doi: 10.1007/s10209-014-0388-6.

[11] F. Monrose and A. D. Rubin, "Keystroke dynamics as a biometric for authentication," *Future Gener. Comput. Syst.*, vol. 16, no. 4, pp. 351–359, Feb. 2000, doi: 10.1016/S0167-739X(99)00059-X.

[12] D. U. Wulff, P. J. Kieslich, F. Henninger, J. M. B. Haslbeck, and M. Schulte-Mecklenbeck, "Movement tracking of psychological processes: A tutorial using mousetrap," *Behav. Res. Methods*, vol. 57, no. 11, p. 307, Oct. 2025, doi: 10.3758/s13428-025-02695-2.

[13] Hotjar Ltd., *Hotjar*. (2026). [Online]. Available: https://www.hotjar.com

[14] Mouseflow ApS, *Mouseflow*. (2026). [Online]. Available: https://mouseflow.com

[15] Microsoft Corporation, *Microsoft Clarity*. (2026). [Online]. Available: https://microsoft.com

[16] Human Communication Interaction Group, *pywib: Human Communication Interaction*. (2026). [Online]. Available: https://humancommunicationinteraction.github.io/pywib

[17] P. Khanna and M. Sasikumar, "Recognising Emotions from Keyboard Stroke Pattern," *Int. J. Comput. Appl.*, vol. 11, no. 9, pp. 1–5, Dec. 2010, doi: 10.5120/1614-2170.

[18] P. J. Kieslich, F. Henninger, D. U. Wulff, J. M. B. Haslbeck, and M. Schulte-Mecklenbeck, "Mouse-Tracking: A Practical Guide to Implementation and Analysis 1," in *A Handbook of Process Tracing Methods*, 2nd ed., Routledge, 2019.

[19] G. Muthumari, R. Shenbagaraj, and M. Blessa Binolin Pepsi, "Mouse gesture based authentication using machine learning algorithm," in *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, Ramanathapuram, India: IEEE, May 2014, pp. 492–496. doi: 10.1109/ICACCCT.2014.7019492.

[20] A. Seelye *et al.*, "Computer mouse movement patterns: A potential marker of mild cognitive impairment," *Alzheimers Dement. Amst. Neth.*, vol. 1, no. 4, pp. 472–480, Dec. 2015, doi: 10.1016/j.dadm.2015.09.006.

[21] A. Strzelecki and A. Miklosik, "Device-dependent click-through rate estimation in Google organic search results based on clicks and impressions data," *Aslib J. Inf. Manag.*, vol. 77, no. 3, pp. 513–529, Apr. 2025, doi: 10.1108/AJIM-04-2023-0107.

[22] C. Epp, M. Lippold, and R. L. Mandryk, "Identifying emotional states using keystroke dynamics," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vancouver BC Canada: ACM, May 2011, pp. 715–724. doi: 10.1145/1978942.1979046.

[23] M. Schoemann, M. Lüken, T. Grage, P. J. Kieslich, and S. Scherbaum, "Validating mouse-tracking: How design factors influence action dynamics in intertemporal decision making," *Behav. Res. Methods*, vol. 51, no. 5, pp. 2356–2377, Oct. 2019, doi: 10.3758/s13428-018-1179-4.

[24] E. Kuric, P. Demcak, M. Krajcovic, and P. Nemcek, "Is mouse dynamics information credible for user behavior research? An empirical investigation," *Comput. Stand. Interfaces*, vol. 90, p. 103849, Aug. 2024, doi: 10.1016/j.csi.2024.103849.

[25] J.-L. Boulanger, "Technique to Manage Software Safety," in *Certifiable Software Applications 1*, Elsevier, 2016, pp. 125–156. doi: 10.1016/B978-1-78548-117-8.50006-4.

[26] D. Fernández-Lanvin, J. De Andrés, M. González-Rodríguez, and P. Torre, "Interaction Lab: Web User Interaction Tracking and Analysis Tool:," in *Proceedings of the 18th International Conference on Web Information Systems and Technologies*, Valletta, Malta: SCITEPRESS - Science and Technology Publications, 2022, pp. 152–158. doi: 10.5220/0011567100003318.

[27] K. Gajos, K. Reinecke, and C. Herrmann, "Accurate measurements of pointing performance from in situ observations," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Austin Texas USA: ACM, May 2012, pp. 3157–3166. doi: 10.1145/2207676.2208733.