# From Pixels to Interaction Graphs: Automated Usability Analytics in the SmartAPSUS Case Study

Igor Vanderlei
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
igor.vanderlei@ufape.edu.br

Rodrigo Andrade
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
rodrigo.andrade@ufape.edu.br

Allysson Guimarães
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
allysson.guimaraes@ufape.edu.br

Fernando Emídio
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
fernando.emidio@ufape.edu.br

Gabriel Aquino
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
gabriel.silvaa@ufape.edu.br

Miguel Vanderlei
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
miguel.vanderlei@ufape.edu.br

Dimas Cassimiro Nascimento
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
dimas.cassimiro@ufape.edu.br

## Abstract

Usability is a critical determinant of software quality, yet the post-test analytics phase remains a significant bottleneck for usability evaluation. Manually deriving metrics such as task completion time, state sequences, and click counts from session recordings is tedious, time-consuming, and dependent on expert evaluators. To mitigate this bottleneck, we propose UxMiner, a tool that automatically extracts quantitative usability metrics directly from screen recordings by combining Optical Character Recognition, computer vision, and a local Small Language Model. We demonstrate UxMiner's feasibility through a running example on the SmartAPSUS public-health platform, where the tool processed an 8-minute session in approximately twice its duration, extracting 15 tasks, 181 interface states, 166 transitions, and 162 click events without any manual intervention.

## Keywords

Usability Test, SmartAPSUS, UxMiner, Usability Metrics.

## 1 Introduction

Usability is one of the most critical determinants of software quality and a decisive factor in product adoption, user retention, and overall success [5]. Industry standards define usability as the extent to which users can employ a product to achieve specific objectives with effectiveness, efficiency, and satisfaction within a given context of use [11]. Optimizing usability simultaneously reduces task completion times, decreases the number of user errors, and increases overall user satisfaction [8]. In contrast, software with poor usability is easily abandoned in favor of more usable alternatives, and in domains such as healthcare, where the consequences of misuse are critical, usability failures can translate into significant impacts [10].

Organizations employ numerous methods to assess system usability [14]. Among these, the usability test stands out as an empirical evaluation method in which researchers recruit representative users to perform a predefined set of tasks while observing interactions and collecting behavioral data [1]. During each session, participants are typically asked to think aloud, verbalizing their thoughts, expectations, and difficulties, allowing researchers to identify use errors, navigation problems, and points of confusion. From these sessions, researchers collect quantitative metrics such as task completion rates, task completion times, number of clicks, and frequency of errors [1, 8, 9], which inform subsequent design improvements.

Despite their widespread use, usability tests impose a burden on the post-test analytics phase. The analysis of user videos is typically a time-consuming process [22]. Researchers tend to follow ad-hoc procedures, frequently switching between spreadsheets and video players, increasing their cognitive load, and producing data structures that are difficult to reuse across studies or to benchmark against existing datasets [9].

Furthermore, most established usability evaluation techniques require the active participation of at least one experienced expert to interpret the results [4]. In practice, these techniques rely on mechanisms such as software instrumentation and interaction logging, eye-tracking and specialized hardware, or manual video inspection. Instrumentation-based approaches provide accurate interaction events but require modifications to the target application [12]. Eye-tracking and other specialized-hardware approaches [14] capture rich attentional data but depend on dedicated equipment, which increases their cost. Manual video analysis requires no instrumentation, but is labor-intensive and difficult to reproduce [9, 22]. Together, these constraints make usability testing resource-intensive and limit its scalability.

To mitigate these issues, we propose UxMiner, a tool that automatically mines quantitative usability metrics directly from screen recordings of usability test sessions. UxMiner takes a raw video as its sole input and produces a structured behavioral benchmark of the recorded interaction. The tool combines Optical Character Recognition (OCR), computer vision, and a local Small Language Model (SLM) into a pipeline that runs through two parallel tracks: one identifies observable user interface states and the transitions between them, and the other captures high-frequency interaction events such as single and double clicks. The resulting interaction graph, enriched with semantic labels, supports automated computation of standard usability metrics without manual data tabulation or software instrumentation. Therefore, we define the following research question: *Is it feasible to automatically extract quantitative usability metrics from a raw screen recording, without manual tabulation, at a practical computational cost?*

We answer this question by demonstrating the feasibility of UxMiner through a running example based on a recorded usability test on the SmartAPSUS platform, a public-health analytics system developed in partnership with the Brazilian Unified Health System (SUS) [19]. UxMiner processed an 8-minute recording end-to-end in approximately twice its duration identifying 15 tasks, 181 user interface states, 166 inter-state transitions, and 162 click events without any manual intervention. The results show that UxMiner could reduce an analytics workload that would otherwise require hours of frame-by-frame video review to a single automated run.

The remainder of this paper is organized as follows. Section 2 describes the UxMiner tool, including its architecture and its five phases. Section 3 reports a running example on SmartAPSUS [19]. Section 4 discusses limitations and concludes with final remarks.

## 2 UxMiner: Automated Usability Analytics from Screen Recordings

This section describes the UxMiner tool. We first summarize its execution workflow in Section 2.1. Next, Sections 2.2, 2.3, 2.4, 2.5, and 2.6 detail the sequential phases UxMiner executes to extract usability metrics from raw test videos.

### 2.1 Overview

For our approach to properly work, a researcher must have conducted a usability test with a user and recorded it on video. Thus, UxMiner automatically analyzes this video. Figure 1 presents an overview of our proposed pipeline for the automated extraction of metrics from usability test recordings. The pipeline is organized into four sequential phases and takes a raw video file as its sole input. It requires no manual annotation beyond the spoken task boundary markers (e.g., Task one started, Task one concluded). We explain each phase in detail next.

### 2.2 Preprocessing

In summary, this phase prepares the raw video recording for subsequent analysis. While conceptually simple, this phase establishes the temporal reference frame that governs the pipeline, that is, the subsequent operations are aligned to the timestamps produced in this stage. This ensures that audio-derived task boundaries and frame-derived user interface states share a common time base. Furthermore, this phase produces three output artifacts regarding audio extraction, frame extraction, and metadata recording.

**Audio extraction.** As a first step, our tool extracts the audio track from the raw video file. Then, it downmixes the audio to a single channel at 16 kHz. These parameters comply with input format expected by modern automatic speech recognition systems and avoid unnecessary computational overhead in the *Segmentation* phase [20]. We serialize the resulting audio stream as `audio.wav`, which preserves the temporal alignment with the original raw video.

**Frame extraction**. In parallel, UxMiner decodes and samples the visual stream at a fixed rate of two frames per second. This sampling rate is intended for Track A (the *State Extraction* phase, Section 2.4), which applies Visual Similarity, Textual Similarity, and OCR on the resulting frame sequence. We select two frames per second to balance two competing requirements: it must be sufficiently high to capture transient interface states such as modal dialogs, dropdown menus, and short-lived feedback messages, yet low enough to keep the computational processing manageable. We observe that a higher sampling rate would increase such processing time without proportionally improving the granularity of detected states. Track B (the *Click Detection* phase, Section 2.5) does not rely on this sampled sequence and instead reads the original video at its native frame rate, since the visual signature of a click is a transient event that occurs at sub-second granularity [20]. Moreover, we store the extracted frames as individually numbered PNG images in the `frames/` directory, with filenames encoding their temporal position to enable reverse-mapping from any frame to its timestamp in the original recording.

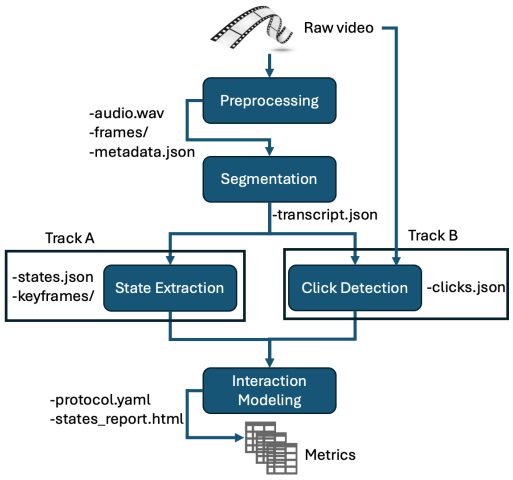


**Figure 1: Overview of UxMiner tool.**

**Metadata recording**. Finally, we generate a metadata file (`metadata.json`) to record the technical properties of the source video, including its original frame rate, total frame count, resolution, duration in seconds, container format, and the paths to the extracted audio and image frames. This file does not directly feed any subsequent phase, but it has a role in traceability and reproducibility. If necessary, we refer to this file to validate the temporal alignment between the audio and frame streams.

Together, the audio extraction, frame extraction, and metadata recording constitute the foundation upon which the remaining phases operate. The *Segmentation* phase consumes the audio stream

to delimit task intervals, while the *State Extraction* phase consumes the frame sequence to identify observable user interface states within those intervals, and the *Click Detection* phase consumes the original video to detect interaction events. By decoupling video decomposition from the remaining phases that follow, UxMiner ensures that costly extraction operations are performed exactly once per recording.

## 2.3 Segmentation

In summary, this phase partitions the extracted audio into discrete, timestamped task intervals. Users execute these tasks during the recorded usability test. Therefore, we must identify the boundary between tasks before any per-task analysis can take place. This phase relies exclusively on the audio stream produced by the *Preprocessing* phase and it operates in three sequential steps: audio chunking, speech-to-text transcription, and task boundary detection.

**Audio chunking.** Long audio streams are challenging to transcribe in a single pass, both due to memory constraints in the speech recognition model (our tool uses the Whisper model [20]) and because transcription quality degrades with utterance length [2]. To address this, UxMiner first partitions the audio into smaller chunks using silence-based segmentation. We detect these silence intervals as a noise floor of -35 dB and a minimum silence duration of 0.70 seconds.

We then construct audio chunks subject to two constraints: each chunk must be at least 8 seconds long to provide sufficient context for the transcription step and at most 60 seconds long to bound processing time and memory usage. When a silence interval exceeds the maximum chunk duration, we further subdivide it into fixed-length segments. The resulting chunks preserve a small padding interval at their boundaries to avoid clipping speech at the edges.

**Speech-to-text transcription.** UxMiner then transcribes each audio chunk independently using the Whisper automatic speech recognition model [20]. We apply voice activity detection during transcription to suppress silence intervals and discard trivial outputs that correspond to known hallucination patterns of the Whisper model, such as single-character outputs or short filler interjections in silent regions [3, 13]. For each chunk, this model returns a sequence of timestamped text segments.

**Task boundary detection.** The final step identifies the start and end of each task by scanning the transcribed segments for spoken boundary markers signaled by the researcher who conducts the usability test session. UxMiner defines two categories of markers: start markers (e.g., "begin of task") and end markers (e.g., "end of task"). Because the researcher may also enumerate tasks explicitly (e.g., "begin of task one"), we additionally detect numbered markers and use them to assign task identifiers directly. To increase robustness, the detection logic accepts multiple lexical variants of each marker class through a set of regular expressions. Furthermore, to mitigate errors, the detection logic incorporates a refractory interval that prevents a new task from being started within a short window after the previous start, and an implicit-end mechanism that ends any open task when a new start marker is encountered without an intervening end marker.

At last, these task intervals, persisted in the `transcript.json` file, constitute the structural backbone of the remaining phases. The *State Extraction* phase operates exclusively within these intervals to identify the user interface states associated with each task. The *Click Detection* phase uses the intervals to attribute each detected interaction event to the corresponding task. Finally, the *Interaction Modeling* phase uses them as the basis for computing usability metrics.

## 2.4 State Extraction

This phase identifies the observable user interface states a user passes through while executing each task and records the transitions between them. A state corresponds to a distinct configuration of the interface, that is, a particular screen, dialog, or input condition that persists long enough to be perceptually meaningful. Together, the states and transitions form a directed graph that captures the structural backbone of the user's interaction with the system, independent of the underlying source code. This phase corresponds to Track A in Figure 1 and operates exclusively on artifacts produced by previous phases. It includes four sequential steps: keyframe extraction, OCR, state consolidation, and semantic enrichment.

**Keyframe extraction.** As a first step, UxMiner partitions the frame sequence produced by the *Preprocessing* phase into per-task subsets, using the task intervals contained in `transcript.json`. For each frame, UxMiner derives its timestamp from its filename and assigns it to the task whose interval contains that timestamp. We discard frames that fall outside any task interval (e.g., between tasks or during session setup). The result is a structured collection of task-aligned frames that constitutes the input for the subsequent steps.

**Optical Character Recognition (OCR).** UxMiner then applies the Tesseract[1] OCR engine to each task-aligned frame to extract the on-screen text. Prior to OCR, our tool crops each frame to remove fixed regions of the screen that are not relevant to the usability test. For example, a browser search bar at the top of the recording. Then, UxMiner associates the resulting OCR text with its source frame and stores it for further processing.

**State consolidation.** This step is the core of the *State Extraction* phase. UxMiner compares consecutive frames within a task pairwise to decide whether a new state has begun or whether the interface remains in the same state. UxMiner uses two independent signals to detect change:

- A visual signal, computed as the Structural Similarity Index (SSIM) [15] between consecutive frames. When the visual similarity falls below 0.92, UxMiner interprets the change as a screen-level transition. For instance, navigation to a new page, the appearance of a modal dialog, or the display of a search result. Then, it commits a new state immediately.
- A textual signal, computed as the Jaccard [16] similarity between the sets of tokens extracted by OCR from consecutive frames. When the textual similarity falls below 0.75, or when new tokens appear that were not present in the previous state, UxMiner interprets the change as in-place content modification, which is typically the typing of input into a form field. Because OCR output can fluctuate during partial keystrokes, UxMiner commits textual-only changes just after the new content has remained stable across at least two consecutive frames.

This dual-signal mechanism allows the pipeline to distinguish abrupt navigational transitions (which should produce a new state

[1] https://github.com/tesseract-ocr/tesseract

immediately) from incremental input events (which should be consolidated into a single state representing the final stable content). For each committed state, UxMiner stores a representative frame in `keyframes/`, computes an MD5 [21] fingerprint over the normalized OCR text for use as a state identifier, and records the transition from the previous state.

**Semantic enrichment.** For this step, UxMiner sends the OCR text of each state to a local Small Language Model (SLM) and requests a short functional description of the interface state. Currently, we use qwen2.5:3b [23] SLM served by a local Ollama [17] instance.

We instruct this SLM, through a structured prompt, to return three fields in JSON form: a label of up to three words identifying the functional role of the state (e.g., "login form", "search results"), a summary of up to twelve words describing what the user is seeing, and a list of three to six functional keywords. To guide the model toward semantically meaningful labels, the prompt explicitly instructs it to prioritize terms that describe the functional purpose of the interface. Then, UxMiner merges the enriched labels into each state and persists in `states.json`.

Together, the keyframe extraction, OCR, state consolidation, and semantic enrichment steps produce a labeled state graph that captures the structural progression of the user through each task. Figure 2 shows an example of a labeled state graph for a simple Login task that contains five states, four transitions, and a duration of 12.4 seconds. Each node corresponds to an observable state of the user interface, and each edge corresponds to a transition between consecutive states. For the `Home page` node, S0001 is the state identifier while 0.00s is the relative timestamp of the state within its task, measured from the moment the task began. Additionally, landing, welcome, home is the set of semantic keywords produced by the SLM during the enrichment step. This graph is one of the two inputs to the *Interaction Modeling* phase (Section 2.6).

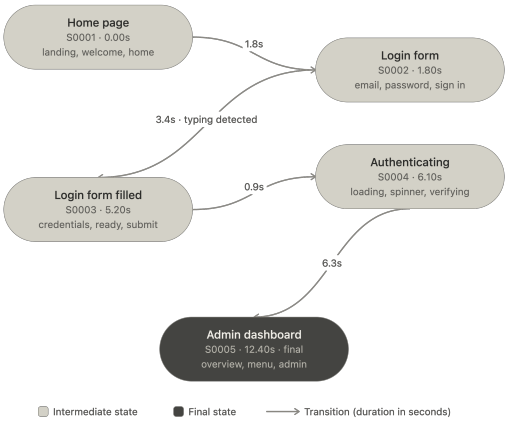


**Figure 2: State graph for the Login task.**

## 2.5 Click Detection

This phase identifies the pointer interactions a user performs while executing each task, and attributes them to the corresponding task interval. While the *State Extraction* phase captures what the interface looks like at each moment, this phase captures when and where the user acted. Together with the state graph produced by Track A, the resulting click stream allows UxMiner to compute interaction metrics such as click count per task, click rate, redundant clicking, and double-click frequency. This phase corresponds to Track B in Figure 1 and runs in parallel with Track A. It operates in three sequential steps: high-rate frame sampling, click marker detection, and post-processing and quality control.

**High-rate frame sampling.** Unlike Track A, which operates on the pre-sampled frame sequence produced by the *Preprocessing* phase, *Click Detection* reads the original video directly at 30 frames per second. A higher temporal resolution is necessary because click markers are short-lived visual events that the 2 fps sampling used for state detection would frequently miss. For each processed frame, UxMiner records its timestamp and looks up the corresponding task interval. It skips frames that fall outside any task so that only interactions performed during a task contribute to the metrics.

**Click marker detection.** UxMiner identifies clicks by detecting their characteristic visual signature in each frame. Specifically, we record the usability test video using the Cursorlux Mouse Highlighter [6], which renders a red circular marker at the cursor location at the moment of each click, which UxMiner detects through a two-stage computer vision pipeline. First, it converts the frame to the HSV color space and isolates red regions, since color-based segmentation in HSV performs well under brightness variations under different recording conditions.Second, UxMiner applies the Hough circle transform [7] to the resulting binary mask to identify circular shapes consistent with the click marker. Then, it validates each candidate circle by two additional checks: it must contain a minimum proportion of strongly red pixels in its interior, and the texture inside the circle must be sufficiently uniform. Our goal is to minimize occurrences of false positives, such as circular red icons or logos. **Post-processing and quality control.** The detected click candidates require three post-processing steps before we can use them as reliable interaction data. First, because a single physical click typically appears across multiple consecutive frames due to the persistence of the marker animation, UxMiner applies temporal and spatial debouncing: once a click is registered, any subsequent detection within 100 milliseconds and 20 pixels of the previous detection is discarded as a duplicate of the same event. Second, UxMiner performs a pass to identify double clicks by grouping any pair of consecutive clicks that occur within 250 milliseconds and 45 pixels of one another. UxMiner records double click groups explicitly in the output, since the distinction between a single click and a double click is a meaningful usability signal. For example, repeated double-clicking on a non-clickable element often indicates user confusion. Third, to prevent occasional false positives (e.g., a red circular logo) from contaminating the per-task metrics, UxMiner applies a burst failure heuristic. If it registers more than 15 clicks within a 3-second window, UxMiner marks the task as having an unreliable click count and its detection is aborted.

Together, the three steps produce a structured click stream associated with each task. Each record contains a unique click identifier, the task to which it belongs, its absolute and task-relative timestamps, the screen coordinates at which it occurred, and its membership in a double-click group when applicable.

This resulting click stream is the second of the two inputs to the *Interaction Modeling* phase (Section 2.6), which combines it with the state graph produced by Track A to compute per-task usability metrics.

## 2.6 Interaction Modeling

This phase consolidates the output of Track A and Track B into a unified description of the user's behavior during the recorded usability test session. The *Interaction Modeling* phase combines the previously produced artifacts into a single structured representation that records which states the user passed through, how long each task took, how many clicks each task required, and which clicks occurred during which transitions. This unified representation supports the subsequent computation of standard usability metrics and enables a comparison between sessions. This phase operates in two steps: task aggregation and report rendering.

**Task aggregation.** For each task interval, UxMiner builds a structured entry containing the task identifier, expert duration, ordered state sequence, final state, and the list of inter-state transitions with their durations. UxMiner then attributes every detected click to the transition whose time window contains the click's timestamp, producing a per-transition click count. A transition with many clicks might indicate a complex interaction or user confusion.

**Report rendering.** Finally, UxMiner serializes the consolidated task entries into the two output artifacts. The `protocol.yaml` file records, for each task, the expert duration, calculated timeout, state sequence, final state, and full transition list. For each state, it records the semantic label, summary, keywords, representative frame, and OCR fingerprint. The `states_report.html` file renders the same information visually to support manual verification of the extracted states and clicks.

Together, these steps produce the final output of the pipeline. The `protocol.yaml` artifact enables automated computation of metrics such as completion status, navigation efficiency, and interaction time. The `states_report.html` artifact allows researchers to visually analyze the resulting usability metrics.

# 3 Running Example

This section demonstrates UxMiner in practice through a running example. Section 3.1 describes the target system, and Section 3.2 reports the analysis that UxMiner produces from a recorded usability test session.

## 3.1 SmartAPSUS

SmartAPSUS [19] is an integrated analytics platform designed to improve strategic decision-making and support public health management within Brazil's Primary Health Care system. The platform integrates heterogeneous epidemiological, socioeconomic, and operational data from prominent public databases such as IBGE and DATASUS. The system functions through three interconnected core pillars: (i) demand forecasting to anticipate upcoming healthcare needs based on local historical series; (ii) resource allocation optimization to model the distribution of professionals and health units based on minimizing operational costs or geographical distances; (iii) interactive data visualization to track key indicators and simulate "what-if" scenarios. Ultimately, SmartAPSUS provides data-driven recommendations that help administrators maximize population coverage, lower expenses, and achieve an equitable distribution of resources throughout the Unified Health System (SUS). The time-consuming task of manually analyzing usability test videos for SmartAPSUS motivated the development of UxMiner. Thus, we use this platform as a target system to demonstrate the feasibility of UxMiner in Section 3.2.

## 3.2 Screen Recording Analysis

To demonstrate UxMiner's feasibility, we ran the tool end-to-end on a screen recording of one of the authors of this work performing a complete usability test on the SmartAPSUS platform considering 15 tasks. The recording is 494.4 seconds long (8 minutes and 14 seconds), captured at 1920×1080 resolution and 29.55 frames per second. We executed the complete pipeline on a Dell G15 notebook equipped with an Intel Core i7-13650HX CPU, 32 GB of DDR5 memory at 4800 MT/s, an NVIDIA RTX 3050 GPU, and an NVMe Gen 4 SSD, running Linux Mint 22.1 and Python 3.12.3.

From the recording, UxMiner identified 15 tasks bounded by spoken markers, with task durations ranging from 4.3 to 66.6 seconds (mean 24.5 seconds). Across all tasks, the pipeline extracted 181 distinct user interface states, 166 inter-state transitions, and 162 click events, of which 24 were classified as double-click groups. Table 1 summarizes the per-task statistics produced by UxMiner.

**Table 1: Per-task statistics produced by UxMiner for the 15 tasks detected in the SmartAPSUS usability test recording.**

| Task | Time (s) | States | Trans. | Clicks | Single | Double |
|---|---|---|---|---|---|---|
| T01 | 25.12 | 4 | 3 | 8 | 8 | 0 |
| T02 | 21.15 | 5 | 4 | 5 | 5 | 0 |
| T03 | 21.50 | 6 | 5 | 9 | 7 | 1 |
| T04 | 4.26 | 1 | 0 | 2 | 2 | 0 |
| T05 | 12.36 | 4 | 3 | 6 | 6 | 0 |
| T06 | 21.96 | 12 | 11 | 14 | 14 | 0 |
| T07 | 24.22 | 25 | 24 | 18 | 4 | 7 |
| T08 | 33.51 | 26 | 25 | 19 | 7 | 6 |
| T09 | 30.79 | 20 | 19 | 17 | 8 | 4 |
| T10 | 22.27 | 14 | 13 | 10 | 2 | 4 |
| T11 | 13.57 | 3 | 2 | 4 | 4 | 0 |
| T12 | 20.42 | 6 | 5 | 4 | 4 | 0 |
| T13 | 20.33 | 12 | 11 | 4 | 4 | 0 |
| T14 | 29.01 | 13 | 12 | 9 | 9 | 0 |
| T15 | 66.63 | 30 | 29 | 33 | 26 | 2 |
| **Total** | **367.10** | **181** | **166** | **162** | **110** | **24** |

The variation across tasks is itself informative for the usability researcher. Tasks T01, T04, T05, and T11 produced short state sequences (1 to 4 states), reflecting straightforward navigational flows. In contrast, T07, T08, T09, and T15 produced 20 or more states, indicating complex interaction patterns with extensive form filling or repeated navigation. The double click activity is also revealing: T07, T08, T09, and T10 each registered four or more double click groups, which a researcher might investigate further as candidate evidence of user confusion or interface unresponsiveness.

Additionally, UxMiner processed the entire 494.4-second recording in 1023 seconds. Figure 3 breaks down this total in all pipeline phases. Two operations dominate the total cost: Track A's OCR step accounts for 56.8% of the execution time (580.7 s), and Track B's click detection accounts for 21.2% (216.5 s). Together with Track A's semantic enrichment via the local Ollama SLM (17.5%, 178.8 s), these three operations are responsible for 95.5% of the total processing time. In this way, the remaining phases together account for less than 5% of the total. This distribution suggests that future optimization effort should target OCR throughput and high-rate frame processing.

In summary, UxMiner produced both the machine-readable benchmark and the human-readable report without any manual intervention beyond the spoken task markers recorded during the

usability test session itself. Manually deriving equivalent metrics, such as task durations, state sequences, transitions, click counts, and double-click events for 15 tasks, would typically require several hours of frame-by-frame video review and tabulation by a trained researcher [8, 9]. Our tool reduces this analytics workload to a single automated run that completes in roughly twice the duration of the recording, while producing structured artifacts that are directly reusable across studies. Last but not least, we answer our research question (Section 1) by stating that these results indicate that automatically mining quantitative usability metrics from raw screen recordings is feasible, without manual tabulation, and at a practical computational cost.

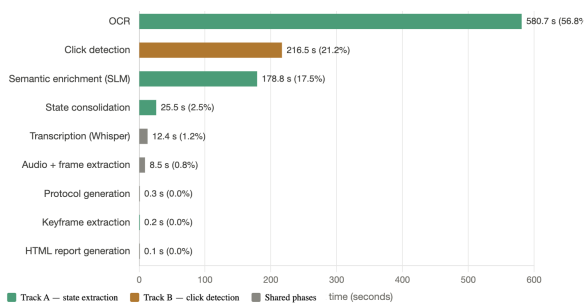


**Figure 3: Processing time per pipeline phase.**

## 4 Limitations and Final Remarks

**Limitations.** We group the limitations of the current UxMiner version into two thematic clusters: dependence on external tooling and protocol, and generalization to other contexts. Regarding the first cluster, UxMiner relies strictly on the Cursorlux Mouse Highlighter tool to render click marks. Without it, Track B cannot detect any pointer interactions, meaning it lacks a generic computer-vision capability. Additionally, the *Segmentation* phase requires the researcher to vocalize explicit task boundaries (e.g., "begin of task one"). This constraint prevents researchers from simply analyzing existing legacy usability recordings without adapting to the specific spoken script and highlighting software beforehand.

Regarding context generalization, several components rely on fixed parameters that may not transfer cleanly to other applications. Because the state consolidation step evaluates Jaccard similarity over OCR-extracted tokens, it assumes a text-heavy interface. Consequently, layout designs dominated by visual elements, such as image galleries, yield impoverished state graphs.

**Final Remarks.** This paper introduces UxMiner, an automated tool that extracts quantitative usability metrics directly from screen recordings of usability test sessions to mitigate a major post-test analytics bottleneck. By combining OCR, computer vision, and a local SLM, the tool utilizes a parallel two-track architecture to map observable user interface states, state transitions, and discrete click interaction events. A running example on the SmartAPSUS public-health platform demonstrates the practical feasibility of the tool. This automated approach reduces the cognitive load and the hours of manual frame-by-frame video review that usability researchers typically endure. At last, we plan that this approach could support comparisons between different users, expert benchmarks, and future interaction analyses.

## Artifact Availability

The artifacts of this work are available in our Online Appendix [18].

## Acknowledgements

We thank CNPq for its support and Decit/SECTICS/MS (Ministry of Health) for funding this research project (grant 445259/2023-0). We also acknowledge the use of Claude Opus 4.8 for proofreading and language refinement of this manuscript.

## References

[1] William Albert and Tom Tullis. 2022. *Measuring the user experience: Collecting, analyzing, and presenting UX metrics.* Morgan Kaufmann.
[2] Max Bain, Jaesung Huh, Tengda Han, and Andrew Zisserman. 2023. Whisperx: Time-accurate speech transcription of long-form audio. *arXiv preprint arXiv:2303.00747* (2023). doi:10.48550/arXiv.2303.00747
[3] Mateusz Barański, Jan Jasiński, Julitta Bartolewska, Stanisław Kacprzak, Marcin Witkowski, and Konrad Kowalczyk. 2025. Investigation of Whisper ASR Hallucinations Induced by Non-Speech Audio. In *IEEE International Conference on Acoustics, Speech and Signal Processing*. doi:10.1109/ICASSP49660.2025.10890105
[4] Anders Bruun and Jan Stage. 2015. New Approaches to Usability Evaluation in Software Development: Barefoot and Crowdsourcing. *Journal of Systems and Software* 105 (2015), 40–53. doi:10.1016/j.jss.2015.03.043
[5] John W. Castro, Ignacio Garnica, and Luis A. Rojas. 2022. Automated Tools for Usability Evaluation: A Systematic Mapping Study. In *Social Computing and Social Media: Design, User Experience and Impact.* doi:10.1007/978-3-031-05061-9_3
[6] Joshua Daniel. 2024. Cursorlux. https://tinyurl.com/56wh7mtk.
[7] Richard O. Duda and Peter E. Hart. 1972. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM* 15, 1 (1972). doi:10.1145/361237.361242
[8] Juan M. Ferreira et al. 2020. Impact of usability mechanisms: An experiment on efficiency, effectiveness and user satisfaction. *Information and Software Technology* 117 (2020). doi:10.1016/j.infsof.2019.106195
[9] Kasper Hornbæk. 2006. Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies* 64, 2 (2006), 79–102. doi:10.1016/j.ijhcs.2005.06.002
[10] Jessica L. Howe, Katharine T. Adams, A. Zachary Hettinger, and Raj M. Ratwani. 2018. Electronic Health Record Usability Issues and Potential Contribution to Patient Harm. *JAMA* 319, 12 (2018), 1276–1278. doi:10.1001/jama.2018.1171
[11] International Organization for Standardization. 2018. *Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts.* Standard ISO 9241-11:2018. International Organization for Standardization, Geneva, CH. https://www.iso.org/standard/63500.html
[12] Wolfgang Kluth, Karl-Heinz Krempels, and Christian Samsel. 2014. Automated Usability Testing for Mobile Applications. In *International Conference on Web Information Systems and Technologies.* doi:10.5220/0004985101490156
[13] Allison Koenecke et al. 2024. Careless Whisper: Speech-to-Text Hallucination Harms. In *ACM Conference on Fairness, Accountability, and Transparency.* doi:10.1145/3630106.3658996
[14] Jakob Nielsen and Kara Pernice. 2010. *Eyetracking Web Usability.* New Riders, Berkeley, CA.
[15] Jim Nilsson and Tomas Akenine-Möller. 2020. Understanding ssim. *arXiv preprint arXiv:2006.13846* (2020). doi:10.48550/arXiv.2006.13846
[16] Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. 2013. Using of Jaccard coefficient for keywords similarity. In *international multiconference of engineers and computer scientists*, Vol. 1. 380–384.
[17] Ollama. 2026. Ollama: Run Llama 3, Mistral, Gemma, and other large language models locally. https://ollama.com/.
[18] Online Appendix. 2026. nteraction Graphs for Automated Usability Analytics. https://doi.org/10.5281/zenodo.21264281.
[19] Thiago Pinheiro, Mágno Gomes, Igor Vanderlei, Dimas Nascimento, Thiago Fabrício, Rodrigo Andrade, and Daliton Silva. 2025. SmartAPSUS: Integrando Dados, Previsão e Otimização para Fortalecer a Gestão da APS no Brasil. In *Anais Estendidos do XL Simpósio Brasileiro de Bancos de Dados.* doi:10.5753/sbbd_estendido.2025.247659
[20] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, and Ilya Sutskever. 2023. Robust Speech Recognition via Large-Scale Weak Supervision. In *International Conference on Machine Learning.*
[21] Ronald Rivest. 1992. RFC1321: The MD5 message-digest algorithm. doi:10.17487/RFC1321
[22] Maximilian Speicher, Brian Palmieri, Jialun Jiang, and Michael Nebeling. 2024. Enhancing UX Evaluation Through Collaboration with Conversational AI Assistants: Effects of Proactive Dialogue and Timing. In *Conference on Human Factors in Computing Systems.* ACM, Honolulu, HI, USA, 1–17. doi:10.1145/3613904.3642168
[23] An Yang et al. 2025. Qwen2.5 Technical Report. *arXiv:2412.15115v2* (2025). doi:10.48550/arXiv.2412.15115