

COMPUTER-USE TRAINING DATA

Recording Human-Computer Interaction for Digital Human Training

From screenshot loops and post-hoc scrubbing to
accessibility-grounded targets, semantic actions and
capture-time privacy

AUTHOR

Neurologic AI

PUBLISHED

September 2026

COLLECTOR

DesktopTrace

ABSTRACT

A computer-use agent is trained on (*observation, action*) pairs: the screen as it looked before an act, and the act taken. Capturing the observation is easy, and it is not where datasets fail. The label is: *which* control was acted on, and *what* was actually typed. We evaluated four existing collectors (PC Agent, OpenAdapt, DuckTrack and AgentNetTool) and found that all four share one recipe and, with it,

four common weaknesses: the interaction target is whatever the platform's hit test happens to return, typed text is reconstructed from keystrokes, privacy is either absent or applied after recording, and capture is all-or-nothing across the whole machine. This paper describes **DesktopTrace**, the production collector we built instead, and the four changes that define it. **Application scoping runs at capture time**, so an application the operator did not select produces no record at all, a facility none of the four tools has. **A password entry is recorded as stars ("*****")**, never as characters; the operating system withholds those keystrokes from every recorder on the machine, and a verified session shows the entry present with `raw_events: 0` and the password absent from the output. **Targets are accessibility elements, with a bounded tree search** when the hit test returns a container: exact target naming rises from **13/21 to 21/21** in Chrome and **14/18 to 18/18** in Finder, and Chrome's median recorded box shrinks from **119 to 40 points**. Because names come from the accessibility tree rather than the pixels, **icons with no visible text are named** (a gear becomes *Settings*, an *×* becomes *Close*), which in our recordings covers **15 of 32 clicks**. **Text is reconciled** between the field's own value and the keystrokes, because each is wrong in cases the other is right. One real session filling a single search box falls from nine actions to two, and a record states each fact once: **1,764 down to ~1,000 bytes**. DesktopTrace ships for macOS and Windows; the mechanisms and measurements reported here are from the macOS build.

21/21

Exact target
naming in
Chrome, up from
13/21

40 pt

Chrome median
recorded box,
down from 119

15 of 32

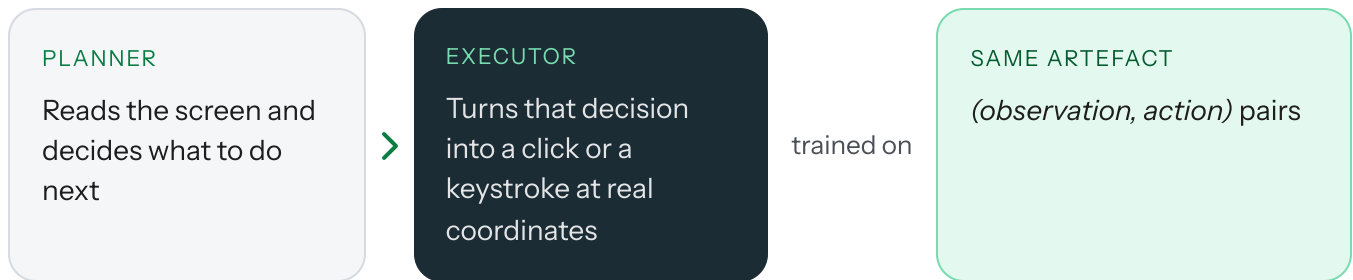
Clicks on icons
with no visible
text, now named

0

Raw events
recorded during a
password entry

Introduction

A Digital Human is two models and a contract between them. A planner reads the screen and decides what to do next; an executor turns that decision into a click or a keystroke at real coordinates. Both are trained on the same artefact: a sequence of (*observation*, *action*) pairs, where the observation is the screen immediately before the act and the action is a structured description of what the person did.



Collecting that artefact looks like a solved problem. Hook the keyboard and mouse, screenshot the display on a timer, pair each action with the most recent frame that predates it, write it all to disk. Four open tools do exactly this, and we ran all four before writing anything. The recipe works. What it produces is not, on its own, a usable training set.

The reason is that the expensive part of an interaction record is not the picture. It is the attribution: the claim that this click landed on the button called *Send*, that this burst of keystrokes put the text *Mumbai* into the field called *City*. That claim is what a planner learns to produce and what an executor is judged against. It is also the part that quietly goes wrong, and when it goes wrong it does not look like an error. A bounding box covering an entire browser tab strip looks exactly like a bounding box covering a button. A typed string missing its final character looks exactly like a string somebody chose to type.

A second problem is that a screen recorder is an unusually invasive instrument. It captures whatever the display is showing, which on a working machine includes other people's data: a customer's email, a colleague's message, a notification banner. For a company building training sets from real work, the question of what a collector declines to record is not a feature request. It decides whether the dataset can be used at all.

This paper describes what we measured in the existing tools, what we changed, and what the changes are worth in numbers. Section 2 states what an interaction record has to contain. Section 3 documents the four tools we evaluated and the gaps they share. Section 4

describes DesktopTrace's design decisions, each tied to the failure that motivated it. Sections 5 and 6 cover implementation and measurement. Section 8 states the limitations, including the ones we have not solved.

— 02

The problem

2.1 What an interaction record has to contain

For a pair to be usable in training, five things must be true of it, and each is a separate opportunity to be wrong.

1 The observation must predate the action.

A screenshot taken after a click shows the consequence, not the situation the decision was made in. Every tool in this study understands this and pairs backwards; it is the one point of universal agreement.

2 The action must be semantic, not raw.

A person filling in a form performs one act: type the city. The operating system reports it as ten key presses and ten releases. A dataset of raw events teaches a model to press keys; a dataset of acts teaches it to fill in forms. Where the boundary between acts is drawn, and by what rule, is a design decision with visible consequences downstream.

3 The target must be identified, not merely located.

A coordinate is worthless across window sizes, screen resolutions and application versions. What generalises is the control: its role, its name, its box. This is what the accessibility tree is for, and it is where the tools diverge most.

4 The text must be what was entered, not what was pressed.

One key is not one character. A backspace against an autocompleted selection deletes a run; an input method commits several characters at once; a dead key makes one character from two presses. A reconstruction from keystrokes is a guess that is usually right, and silently wrong in exactly the cases that matter.

5

The record must say how confident it is.

Some interactions cannot be attributed: the point lands on a canvas, or the application publishes nothing there. Those records are still worth keeping (the geometry and the screenshot are real), but a consumer must be able to exclude them before treating a label as ground truth. A dataset that cannot distinguish a confident label from a fallback has no floor on its noise.

2.2 The asymmetry that shapes the design

One property of interaction data dominates everything else about its economics and its risk, and it is a mismatch of scale.

The observation is the whole display: on the machine used throughout this paper, $3,420 \times 2,214$ pixels, about 170 KB compressed. The action concerns one control, and controls are small: the close button on a browser tab measures 28×28 points. The informative part of an observation is therefore on the order of 0.03% of its area, and the entire value of the record depends on correctly identifying which 0.03% it was.

OBSERVATION **$3,420 \times 2,214$**

pixels, the whole display,
about 170 KB compressed

ACTION **28×28**

points, the close button on a
browser tab

INFORMATIVE PART**~0.03%**

of the observation's area

Two consequences follow, and both are load-bearing for the rest of this paper.

Attribution deserves the engineering budget. Capturing frames is a solved problem that costs a few milliseconds. Naming the target is an unsolved problem that costs correctness, and it is where the measured differences between tools appear. Our own element lookup spends roughly 20 ms per action and we consider that well spent; the screenshot that accompanies it costs less.

The blast radius of a screenshot is the whole screen. Restricting which actions are recorded does not restrict what the frame shows. Any privacy design that operates on actions alone (which describes every filtering facility we found) leaves the images untouched. This is the single most under-estimated risk in tools of this kind, and Section 8 states plainly that we have not fully solved it either.

How interaction data is collected today

We ran and read four open collectors before designing ours: PC Agent's PC Tracker, OpenAdapt, DuckTrack and AgentNetTool. Each is described below on its own terms, with what it does well and where it falls short for our purpose. The claims here come from reading the source, not the documentation; where a number appears we state where it is set.

3.1 The recipe all four share

Despite different goals and codebases, the four converge on one architecture, and it is worth stating because the shared parts are also the shared limitations.

01

Raw input through `pynput`.

All four hook the keyboard and mouse with the same library and push every event onto a queue, never blocking the hook.

02

Screen capture decoupled from input.

Either a screenshot timer (PC Tracker at 4 fps, OpenAdapt at 10 fps) or a continuous video (DuckTrack and AgentNetTool drive OBS Studio at 30 fps over its WebSocket API).

03

Pair backwards.

An action is matched with the most recent capture that predates it. Universal, and correct.

04

Attribution as a side channel, if at all.

Where the element under the cursor is recorded, it is recorded by a single platform hit test and written to its own stream, separate from the action.

3.2 PC Agent / PC Tracker

The collector behind the PC Agent paper, and the closest in intent to what we needed: it targets exactly the planner/executor decomposition a Digital Human uses. Screenshots on a timer, raw events merged into semantic actions, accessibility lookup on macOS through `AXUIElementCopyElementAtPosition`.

WHY IT IS ATTRACTIVE

It is the only one of the four designed from the start around (*observation, action*) pairs for agent training rather than around replay or annotation. Its action vocabulary (click, drag, type, scroll, hotkey) is the right one, and its merging rules are sensible. The output is directly consumable.

WHERE IT FALLS SHORT

Three things, each of which we hit immediately in practice. First, the accessibility lookup is a single hit test with no ancestor walk and no fallback: when the platform returns a container (which, in browsers, is most of the time), that container is what the record says was clicked. Second, the element's name is taken as `AXTitle` or `AXDescription` or `AXValue`, so an unlabelled text field's name becomes its contents. That is both a data-quality problem (the same field has a different "name" in every record) and a privacy one (field contents leak into a field intended for labels). Third, the record carries no indication of where the name came from, so a consumer cannot tell a confident label from a coincidence. The foreground application is read from `NSWorkspace.frontmostApplication()`, an API we measured (§6.4) reporting a stale application for 33 seconds across five real app switches. Screen capture is 4 fps (`capture_interval=0.25`), which under fast interaction can leave an observation a quarter-second older than the decision it is supposed to explain.

3.3 OpenAdapt

The largest and most engineered of the four: a recording database, a processing pipeline, multi-process writers, and a replay system that can re-execute a recording against the live desktop. Screenshots at 10 fps, actions merged in a processing thread, everything written to SQLite by separate writer processes so producers never block on I/O.

WHY IT IS ATTRACTIVE

Its engineering is the most serious of the four, and its privacy story is the only one that exists at all. It integrates Microsoft Presidio, AWS Comprehend and Private AI as pluggable scrubbing providers, with support for redacting both text and image regions; we counted 713 references to scrubbing across the codebase. Its multi-process writer design is the right answer to the "never block the input hook" constraint and influenced our own.

WHERE IT FALLS SHORT

The scrubbing is off by default (`SCRUB_ENABLED: bool = False`) and lives under `legacy/` . More fundamentally, it is post-hoc: it is machine-learning PII detection applied to data already written to disk. That design has three consequences. The sensitive data existed in the clear, on that disk, before anything scrubbed it. A detector that misses is a leak, and named-entity recognition on UI text (labels, fragments, values out of context) is a harder problem than on prose. And it addresses *categories* of content (names, card numbers) rather than *sources* of content, which is the axis an operator actually reasons about: *do not record anything from my mail client*. OpenAdapt is also replay-oriented, so its action stream is closer to the input events than to the semantic acts a planner should be trained on.

3.4 DuckTrack

The simplest of the four: the core recorder is roughly 140 lines. It takes no screenshots at all: OBS Studio records one continuous 30 fps video under remote control, while every raw event goes to `events.jsonl` with no merging and no interpretation. Both streams share one `perf_counter` clock, so any event's timestamp converts to a video offset later.

WHY IT IS ATTRACTIVE

The clock design is genuinely elegant, and 30 fps video is a far better temporal record than any screenshot timer: nothing is missed between frames, and the cost is a single hardware-encoded stream rather than thousands of files. Deferring all interpretation to post-processing means a change of mind about action boundaries does not require re-recording.

WHERE IT FALLS SHORT

For our purpose it produces an input log, not a dataset. There is no accessibility capture, so no target is identified at all: an action is a coordinate. There is no merging, so the output is key presses rather than acts. There is no privacy facility of any kind. And the OBS dependency means the collector cannot be handed to a non-technical operator as a single application, which was a hard requirement for us.

3.5 AgentNetTool

A fork of DuckTrack grown into a full annotation platform: an Electron front end over a Python Flask backend, DuckTrack's OBS recorder almost line-for-line, plus accessibility capture, an offline action-reduction pipeline that turns raw events into semantic actions with

per-action video clips, and a review-and-upload workflow.

WHY IT IS ATTRACTIVE

It is the most complete system of the four and the one that has most clearly been used at scale for real crowdsourced collection. It captures three accessibility streams (the element under each click, the active window polled at 0.2 s, and optionally the full window tree at keyframes), and it is the only tool of the four that encrypts those files line by line on disk, a recognition that a11y capture is sensitive that the others do not make.

WHERE IT FALLS SHORT

The element capture is the same single hit test with the same container problem, and it runs in a background thread after mouse-down rather than synchronously before the UI changes, so a menu that opens on press can be resolved against a tree that has already moved. Semantic actions are produced offline from raw events, which means text is reconstructed from keystrokes and the field's own value is never consulted. Scoping is all-or-nothing: the video records the whole screen for the whole session, and the encryption protects the a11y files at rest, not the decision about what enters them. As a system it is built around a crowdsourcing pipeline with login and upload, which is a different deployment model from ours.

3.6 What none of them does

Four gaps are common to all four tools, and they are what DesktopTrace was built to close.

No application scoping.

We searched all four codebases for any facility to restrict capture to a chosen set of applications: zero occurrences. Every tool records the whole machine for the whole session. An operator who wants to record their work in a CRM must also record whatever else is on their screen and, in every case, a screenshot or video frame of the whole display regardless.

No capture-time privacy.

Only OpenAdapt has a privacy facility at all; it is disabled by default and runs after the data is on disk. None of the four has any special handling of password fields. Three of them record keystrokes with no filtering whatsoever.

No fallback when the hit test fails.

All three tools that capture elements use a single `AXUIElementCopyElementAtPosition` call and accept the result. None walks upward for context, none searches downward when a container comes back, and none records how the name was obtained.

No text reconciliation.

Typed text is reconstructed from key events everywhere. The field's own accessibility value (which the operating system will tell you, and which is right in exactly the cases keystroke reconstruction is wrong) is never read.

Capability	PC Tracker	OpenAdapt	DuckTrack	AgentNet	DesktopTrace
Observation precedes action	yes	yes	yes	yes	yes
Screen capture	4 fps	10 fps	30 fps video	30 fps video	10 fps
Semantic actions at capture	yes	partial	no	offline	yes
Element under cursor	hit test	no	no	hit test	hit test + search
Ancestor context	no	n/a	n/a	tree dump	bounded, named
Search when container returned	no	n/a	n/a	no	yes
Name provenance recorded	no	n/a	n/a	no	yes
Element name can be field contents	yes	n/a	n/a	no	no
Text from field value	no	no	no	no	yes
Text reconciled with keystrokes	no	no	no	no	yes
One action per field edit	no	no	no	no	yes
Application allow-list	no	no	no	no	yes
Password fields never recorded	no	no	no	no	yes
Login still visible in the trace	no	no	no	no	masked
Privacy applied at capture time	no	post-hoc	no	at rest	yes
Recorder excludes itself	no	no	no	no	yes
Settings recorded with the data	no	no	no	no	yes

Table 1. Capability comparison across the four collectors we evaluated and DesktopTrace. Entries were established by reading each tool's source rather than its documentation. "n/a" means the row does not apply because the tool captures no accessibility data. OpenAdapt's "post-hoc" privacy is its Presidio/Comprehend scrubbing pipeline, which is disabled by default; AgentNetTool's "at rest" is line-by-line encryption of the accessibility files, which protects storage rather than restricting what is captured.

04

DesktopTrace

DesktopTrace is a desktop application for macOS and Windows that records interaction as (*observation, action*) pairs for Digital Human training. Both versions write the same record format, so a dataset recorded on either platform is interchangeable. Where a mechanism below is specific to one operating system we say so; the measurements are from the macOS build. It keeps the parts of the shared recipe that are right (backwards pairing, non-blocking hooks, asynchronous persistence) and changes four things. Each subsection below states the failure that motivated the change before describing it.

4.1 Application scoping, enforced at capture

THE FAILURE

An operator recording work in a web application also records their mail, their messages, and the notification that arrives mid-session. The dataset then cannot be shared, or must be reviewed frame by frame before it can. Filtering afterwards does not help: the data already exists.

`allowed_apps` names the applications to record. Anything else produces **no record at all**: not a redacted action, not an action with the text withheld, and no screenshot. The check runs in the recorder before an action is constructed, so nothing from an excluded application is ever built, let alone written. Two weaker lists exist for cases where an allow-list is too strict: `ignore_apps` (record everything except these) and `redact_apps` (keep the act and its geometry, withhold the typed text).

`allowed_apps`

The applications to record. Anything else produces no record at all.

`ignore_apps`

Record everything except these.

`redact_apps`

Keep the act and its geometry, withhold the typed text.

Scoping is stated in application names ("Google Chrome", "Slack") rather than bundle identifiers, because the person configuring a collection run is usually not a developer. The operator picks them from a list of running applications in the control panel before pressing *Start*.

This facility has no counterpart in any of the four tools. It is also the control that does the most work in practice, because it is the one an operator can reason about before recording rather than after.

4.2 A password entry is recorded; a password is not

THE FAILURE

A trace in which the user clicks a field, does nothing, and then clicks *Sign in* is not a redacted trace; it is a wrong one. A planner trained on it learns that sign-in requires no typing. But recording the password is not an option, and "detect and remove it afterwards" is not either.

On macOS the operating system resolves the tension in an unexpected way. While a secure text field has focus, the system enables **Secure Event Input** and withholds key events from every event tap on the machine, including ours. A recorder therefore cannot capture a password typed into a proper password field even if it tries. We verified this directly: a session containing a real password entry reports `raw_events: 0` for that interval.

The consequence is that the interaction is invisible, not merely unrecorded. So DesktopTrace does not listen for those keys at all. A watcher detects the secure state through the Carbon `IsSecureEventInputEnabled` call (a cheap local gate) and then *watches the field* through the accessibility API, which reports that entry is happening without ever exposing what is being entered. When focus leaves, one action is emitted, with the text replaced by stars:

```
{ "type": "type", "text": "*****",  
  "redaction_reason": "secure_field",  
  "element": { "role": "AXSecureTextField", "label": "Password" } }
```

The dataset says a password was typed into the field labelled *Password*, and shows it as stars. It does not say what the password was, and no code path exists by which it could: the masked action never reads the field's contents, and is never permitted to carry text from

any source. This holds regardless of configuration: it is an invariant in the code, not a setting. DesktopTrace also never records its own window, matched on both process id and bundle identifier.

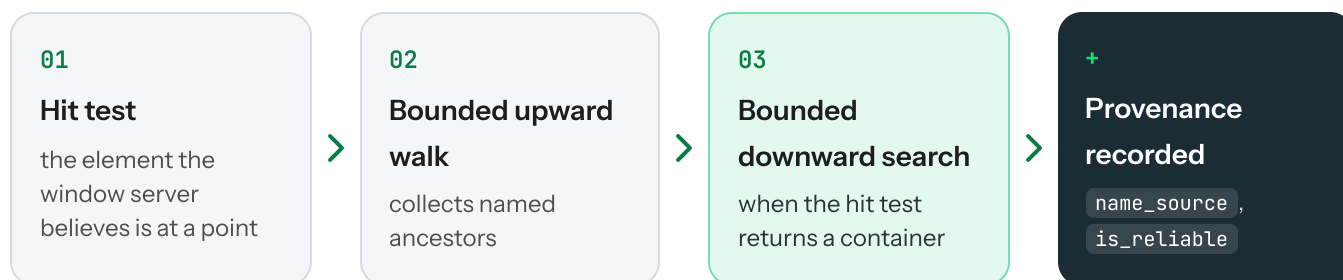
We note the honest limit: this protects *password fields*, which is what the operating system marks as secure. A username typed into an ordinary text field is an ordinary text field, and is recorded like any other. §8 returns to this.

4.3 Targets from the accessibility tree, with a search

THE FAILURE

`AXUIElementCopyElementAtPosition` (the platform hit test, used by every tool here that captures elements) returns the element the window server believes is at a point. Often that is the button. Often it is the container the button sits in, and in browser interfaces it is usually the container. The record then says a click landed on an unnamed group 800 points wide.

DesktopTrace treats the hit test as the first step of three.



A bounded upward walk collects named ancestors, which is what turns an unnamed cell into *a cell in the table Invoices in the window Billing*. Ancestors are filtered to informative ones (73% of raw ancestor entries are unnamed layout containers carrying no information) and the walk is capped. We measured the cap down from 8 to 4 with no loss of naming accuracy and 68% fewer ancestor bytes (§6.3).

A bounded downward search runs when the hit test returns a container. It walks the subtree looking for an interactive role whose box contains the point, capped at depth 14, 40 children per node and a 250-node budget. One detail matters and is easy to get wrong: the walk must **not** prune branches whose rectangle excludes the point. In Chrome a parent's rectangle does not enclose its children, so containment pruning visits four nodes and finds nothing, while the unpruned walk finds a tab's close button in 29 nodes and 2 to 4 ms.

Provenance is recorded. Every element carries a `name_source` saying where its name came from (`title` , `description` , `searched` , and so on) and an `is_reliable` flag. A consumer can filter to confidently-labelled actions before training, which is what gives the dataset a floor on its noise. An element's name is **never** taken from its value, so a field's contents cannot become its label.

The effect is measured in §6.1: exact target naming rises from 13/21 to 21/21 in Chrome and 14/18 to 18/18 in Finder, and Chrome's median recorded box narrows from 119 to 40 points.

4.4 Icons with no visible text

THE FAILURE

A large share of the controls people click carry no words at all: a gear for settings, an × for close, a + for a new tab, a ? for help, an arrow for back. A vision-language model reading the screenshot sees a glyph. It can *guess* that a gear means settings, but it cannot know: the same gear is *Preferences* in one application, *Options* in another and *Workspace settings* in a third, and an executor that must find the same control on a future screen needs its real name, not a description of its shape.

The accessibility tree has that name. Applications label icon-only controls for screen-reader users, and the label is published in the element's description even though nothing is drawn. DesktopTrace records it as the element's name, with `name_source: "description"` , so the record says *Settings* where the pixels say only *gear*.

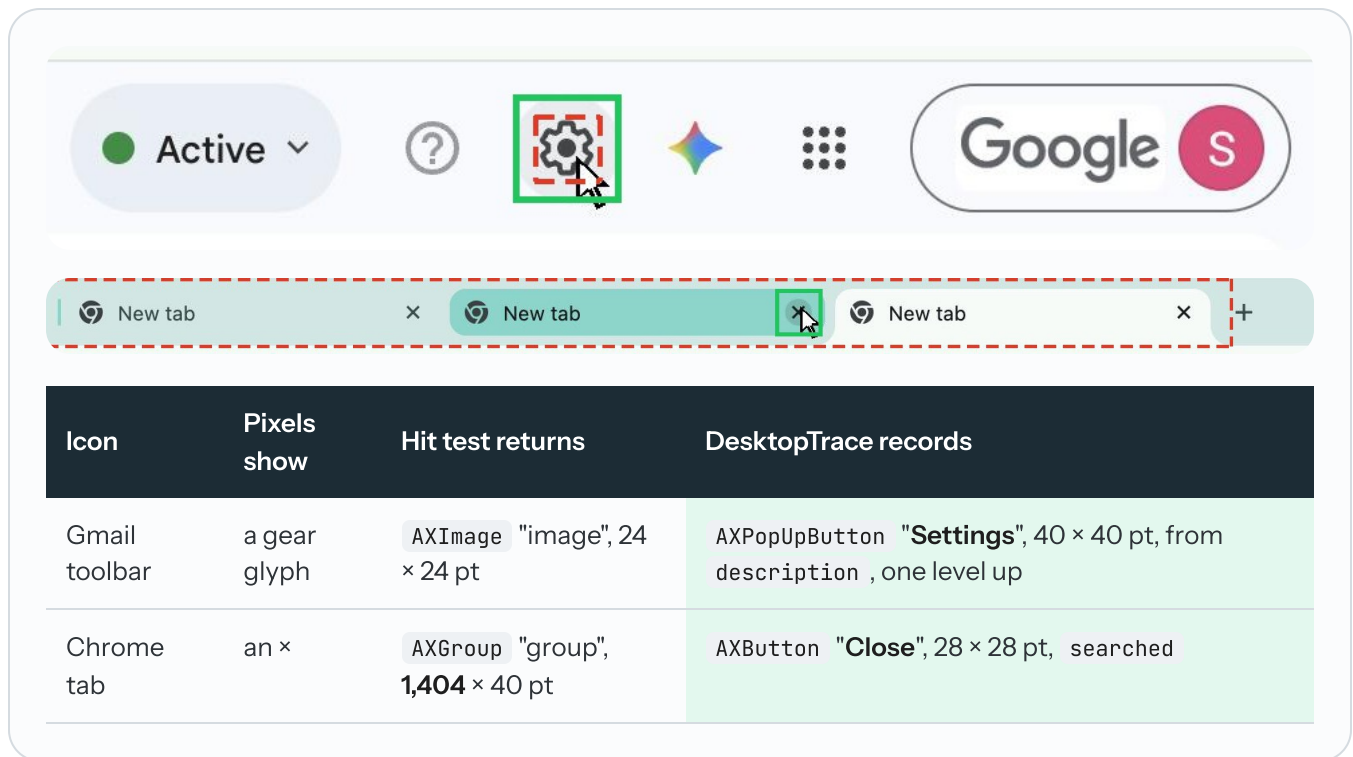


Figure 2. Two icon-only controls from real recordings, cropped from the full-display frames the tool saved. Dashed red: the element the platform hit test returned. Solid green: the element DesktopTrace recorded. Top, the hit test returns the gear glyph itself and the resolver climbs to the labelled button. Bottom, the hit test returns the entire tab strip (the box continues far to the left of the crop) and the bounded search finds the close button inside it. The cursor in both frames is drawn by the tool.

Two details make this work in practice, and both are visible in Figure 2. First, the hit test frequently returns *the glyph itself* (an `AXImage` named only by its role) rather than the button that contains it; the upward walk climbs one level to the named, interactive parent. Second, where the hit test returns a whole region, as it does over Chrome's tab strip, the bounded search of §4.3 finds the icon inside it. The name is never inferred from the image: when an application publishes no label, the record says the name came from the role alone and marks it unreliable, rather than inventing one.

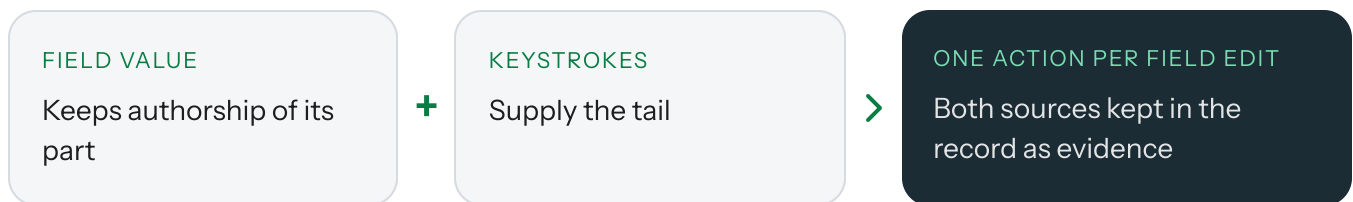
Across the three sessions recorded for this paper, **15 of 32 clicks** landed on an icon-only button: *New tab*, *Close*, *Settings*, *Support*, *Back to Sent Mail*, *Dismiss*. In every one the hit test alone returned either the bare glyph, an unnamed group, or nothing usable.

4.5 Text reconciled between the field and the keys

THE FAILURE

Reconstructing typed text from keystrokes is wrong whenever one key is not one character: a backspace against an autocompleted selection, an input method commit, a dead key. Reading it from the field's accessibility value is wrong too: Jotform reports an empty value throughout, and Chromium updates web-content values asynchronously, lagging by several characters.

Both failures share one shape: **the field's value is a prefix of what was typed**. That is the reconciliation rule. Where the field has a value, it keeps authorship of its part; the keystrokes supply the tail. Both sources are always kept in the record as evidence, so a consumer that disagrees with the rule can apply its own.



Edits to one field are coalesced into a single action: the typing, the corrections and the arrow keys inside a field become one act carrying the text it ended up with, emitted when focus leaves. In one real session, filling a single search box went from nine actions to two, and 21 raw events became one. Every path that ends an edit (a submit key, a click, a scroll, session close) must supply a fresh read of the field, because a path that forgets loses the last character silently, and a truncated string reads as something a person chose to type.

4.6 Records that state each fact once

An early record was 1,764 bytes, of which we measured roughly 25% as provably duplicated: the same geometry in points and in pixels, the raw element dictionary beside the parsed one, ancestors repeating the window title. Each duplicate is a chance for a consumer to read the wrong copy.

Geometry is now stored as **points only**. Pixels require a screenshot, and a dropped frame or `capture_frames: false` would leave a record with no geometry at all. Consumers that need pixels convert through one module that holds the Retina scale and the display origin; the two ways to get this wrong are both silent, so there is exactly one place that multiplies.

`actions.jsonl` is the only machine-readable output, deliberately: a second derived format means a new field has to be added twice and a consumer has to work out which file is authoritative. Records are now around 1,000 bytes.

RECORD SIZE

Early record		1,764 bytes
Now		~1,000 bytes

4.7 What ships with the data

Every session writes the settings it actually ran with into `session.json`, with counts of what was dropped and what was redacted. That file is the evidence of what a dataset does and does not contain, and it travels with the data. A `report.html` is written on stop: every action with its screenshot, the target's box drawn on it, and the full element detail beside it. Reviewing that page before sharing a recording is the step that reliably catches the notification banner or the second monitor.

The repository ships a data-protection policy stating what is captured, which settings reduce it, what is never captured, and what to do before and after a recording, with pointers to India's DPDP Act, the GDPR and the CCPA/CPRA. It is explicit that it is operational guidance and not legal advice.

— 05

Implementation

5.1 The pipeline

Figure 1 shows the path from a key press to a line in `actions.jsonl`. Three constraints shape it.

Nothing may block the input thread. The callback runs inside the operating system's event tap, and macOS *disables* a tap whose callback runs long, which silently truncates the recording rather than reporting an error. The budget is about a millisecond. Disk, image encoding and report generation all happen on workers behind bounded queues.

Observation is synchronous; persistence is not. The element under the cursor and the frame that precedes the action must be taken before the interface changes. A menu that opens on mouse-down has already moved by the time a background thread gets to it. Everything after that point can wait.

The scope gate comes first. Application scoping is checked before an action is constructed, not before it is written. An excluded application produces no object, so there is nothing to leak through a later stage.

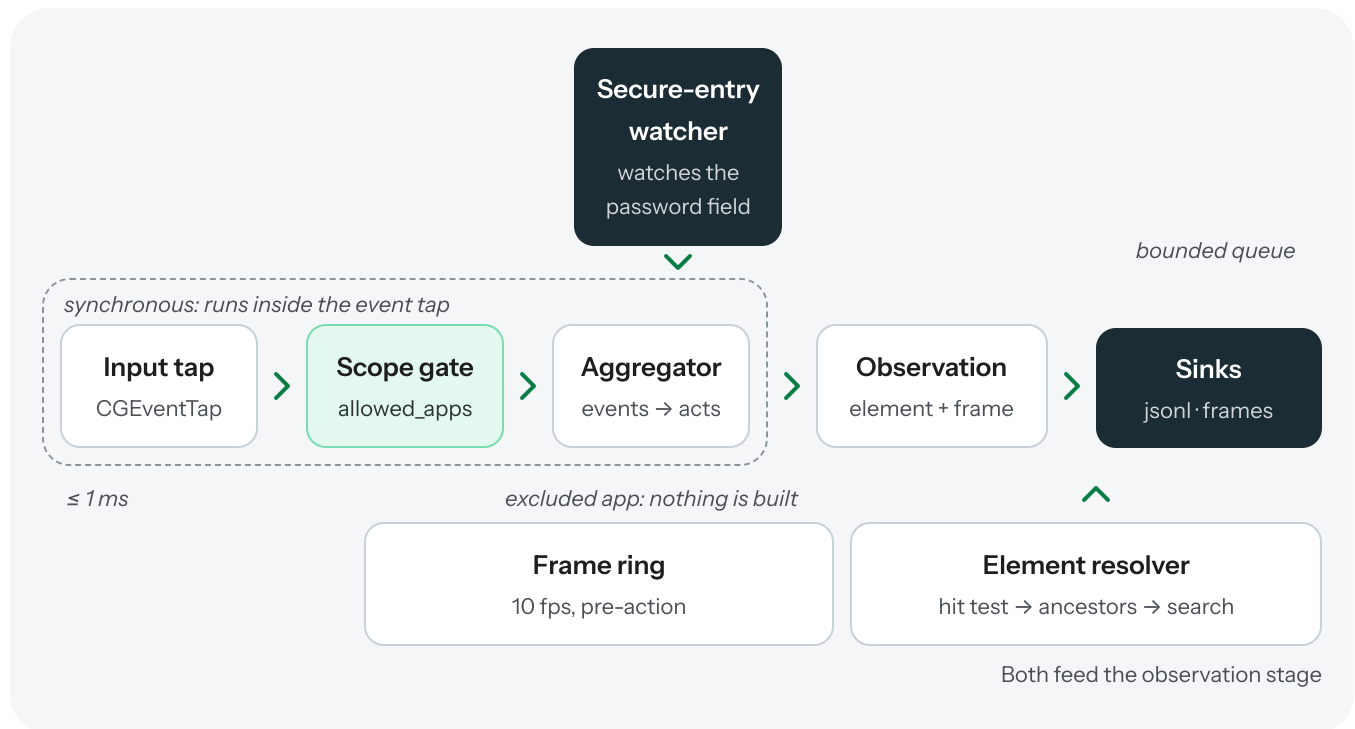


Figure 1. The capture pipeline. The dashed region runs inside the operating system's event tap and has a budget of roughly one millisecond; everything downstream of the observation stage is asynchronous and behind a bounded queue. The scope gate sits before the aggregator, so an action in an unlisted application is never constructed. The secure-entry watcher is a separate path because while a password field has focus the operating system delivers no key events to the tap at all.

5.2 Structure

The system is 12,937 lines of Python across seven packages. One architectural rule carries most of the weight: `core/` **imports no operating-system module**. Everything platform-specific sits behind six protocols in `platforms/base.py`. That is what makes the semantic layer testable with fabricated events on any machine: the whole suite runs with no permissions, no display and no real input, driving a complete recording against a backend made of fakes. There are 344 test functions, 374 tests with parametrisation, across 5,327 lines.

12,937

lines of Python across
seven packages

6

protocols in
`platforms/base.py`

344

test functions, 374
tests with
parametrisation

5,327

lines of tests

The seam is also what makes a second platform possible. The Windows version implements the same contract against the Windows accessibility and input APIs and writes the same record (the same role vocabulary, the same key names, the same geometry convention), so datasets from the two platforms can be mixed in one training set. The line and test counts above describe the macOS codebase.

5.3 Two front ends, one core

A command-line interface and a Tk control panel share every line of the session code and contain no capture logic. The panel exists because the operator is frequently not a developer: it is a single window with the output folder, the application selection, a golden-path toggle and one button. It minimises on start and, on macOS, shows a blinking red dot on its Dock tile, because a recorder that is easy to forget about is a privacy problem.

Two configuration stores exist and deliberately do not merge. One is written by whoever prepares a machine and read by the tool; the other is written by the tool and read back on next launch. Merging them would mean the application rewriting a hand-edited file: reordering it, dropping its comments, and racing an operator who has it open.

— 06

Measurements

All numbers below were measured with the macOS build on one machine (Apple silicon, 3,420 × 2,214 pixel display) against real applications, using an audit tool that compares the resolver's answer with the application's own accessibility tree. They are engineering measurements from a development machine, not a controlled study; §8, item 3 says what that means for how far they generalise.

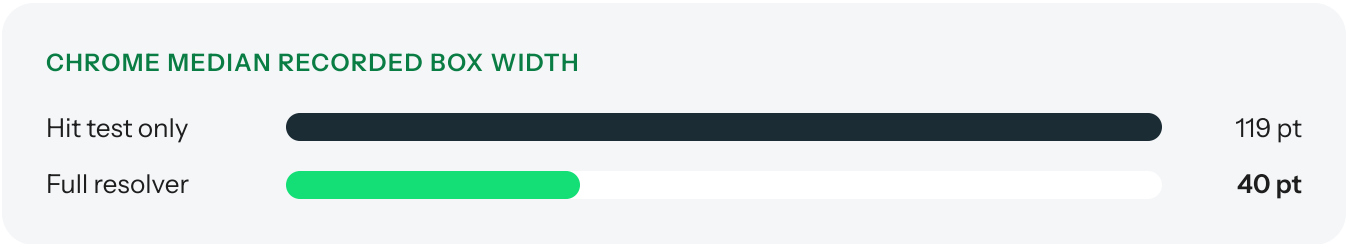
6.1 Target naming

The audit clicks a set of known controls in each application and asks whether the resolver named the control the person actually acted on. "Hit test only" is the single-call approach used by PC Tracker and AgentNetTool; "full resolver" adds the upward walk and the bounded search.

Application	Hit test only	Full resolver	Where the difference comes from
Google Chrome	13 / 21	21 / 21	tab strip and toolbar return one coarse container
Finder	14 / 18	18 / 18	sidebar rows and toolbar items
Slack	16 / 17	17 / 17	channel list rows
Notes	14 / 15	15 / 15	note list
TextEdit	12 / 17	17 / 17	format bar controls
Visual Studio Code	9 / 9	9 / 9	already precise; search not needed

Table 2. Exact target naming, by application. A result counts only when the resolver names the control the person acted on; naming its container counts as a miss. Visual Studio Code shows the case where the hit test alone is already correct and the extra work changes nothing, which is the expected outcome for a well-behaved native application.

The geometric effect is as important as the naming one, because the box is what an executor is trained to aim at. In Chrome the **median recorded box width falls from 119 points to 40**. A 119-point box spanning several tabs is not a wrong label that a model can learn around; it is a label that teaches the model to click in the wrong place.



6.2 The browser case

Chrome deserves its own note because it is where naive capture fails worst and where we spent the most time. Chrome builds two accessibility trees independently: the page, exposed as soon as anything asks for it, and its own interface (tab strip, toolbar), which is built only when Chrome is launched with `--force-renderer-accessibility` or detects a screen reader. We attempted to enable it from outside and could not: `AXManualAccessibility` reports the attribute unsupported, `AXEnhancedUserInterface` is not implemented, and neither reading nor hit-testing the tree triggers it.

Even with accessibility on, the hit test still returns a coarse group over the tab strip. The instructive finding is *why* the obvious fix fails: a downward walk that prunes branches whose rectangle does not contain the point visits **four nodes and finds nothing**, because in Chrome a parent's rectangle does not enclose its children. The unpruned walk finds the tab's close button in **29 nodes, in 2 to 4 ms**. This is recorded here because it is the kind of detail that is invisible in a passing test suite and costs a dataset its labels.

CONTAINMENT PRUNING

4 nodes

visited, finds nothing

UNPRUNED WALK

29 nodes

finds the tab's close button in 2 to 4 ms

6.3 Cost

Stage	p50	max	Note
Element lookup, idle application	0.7-1.7 ms	n/a	hit test plus batched attribute read
Element lookup, under load	20.2 ms	51 ms	full path including ancestors
of which ancestor walk	9.9 ms	n/a	49% of the total
Bounded search, when triggered	2-4 ms	20 ms	29 nodes in the audit; 20 ms in the recorded session of Figure 2

Table 3. Resolver latency. All of it is spent on the observation path, which is synchronous by design; the input tap itself stays within its millisecond budget because everything downstream is queued.

The ancestor walk was the obvious target: half the cost. Reducing `ancestor_limit` from 8 to 4 produced **no loss of naming accuracy** and **68% fewer ancestor bytes**, because 73% of raw ancestor entries are unnamed layout containers that carry no information at all. Filtering to informative ancestors, rather than truncating the list, is what makes a shorter limit free.

6.4 The foreground application

The standard API for "which application is in front", `NSWorkspace.frontmostApplication()`, is used by PC Tracker and by AgentNetTool's window poller. In one session we measured it reporting a stale application for **33 seconds across five real application switches**. Every action in that interval would have been attributed to the wrong application and, with an allow-list, could have been admitted or excluded wrongly.

The cause is that a poller on a timer lags the commonest cause of a foreground change: the user clicking the application they are switching to. DesktopTrace takes an action's application *from the resolved element* and reconciles it against the window server directly, correcting 52 of 65 actions in that session.

6.5 Aggregation and record size

Field coalescing takes one real session's search-box entry from **9 actions to 2** and **21 raw events to 1**. Record size fell from **1,764 bytes to about 1,000** by removing duplication: roughly 25% of the original record restated something it already said elsewhere.

9 → 2

actions for one search-box
entry

21 → 1

raw events to one action

1,764 → ~1,000

bytes per record

6.6 The password case, end to end

A session containing a real sign-in produced the masked action described in §4.2: `type "*****"` with `redaction_reason: "secure_field"`, in a session whose raw-event count for that interval was zero. The password does not appear anywhere in the output. The sign-in does.

One defect worth reporting, because it is the kind that matters: an early version of the watcher bypassed the application allow-list, and recorded a masked password entry in an application the operator had not selected. It was found by reviewing a real session's report.

The watcher now returns nothing for an excluded application, and the case is under test. We mention it because a privacy control is only as good as the paths that respect it, and the secure-entry watcher was a path we had forgotten.

— 07

Discussion

Attribution is where the value is, and it is not cheap.

The measurements above show every meaningful quality difference sitting in the element resolver, not in the capture loop. A collector that spends 20 ms per action naming its target produces a materially better dataset than one that spends 1 ms accepting whatever the hit test said. That trade is worth making because the cost is per action (a few hundred per session) while the error it prevents is in every record.

Privacy at capture time is a different thing from privacy afterwards, and only one of them is a guarantee.

A post-hoc scrubber can be evaluated, tuned and improved, and OpenAdapt's is a serious piece of work. But it operates on data that already exists, and its recall is an empirical property of a model. An allow-list checked before an action is constructed has no recall: the data was never made. When the question is whether a dataset can be shared, an argument of the second kind is the one that holds up.

Recording the shape of an interaction is compatible with withholding its content.

The password case is the clearest instance: the dataset knows that a password was entered into the field labelled *Password*, which is everything a planner needs and nothing an attacker wants. The general principle (record the act, withhold the payload) extends further than we have taken it, and we think it is the right default for a tool of this kind.

The operating system is sometimes an ally.

Secure Event Input means macOS actively prevents any recorder from capturing a password typed into a proper password field. That is worth knowing in both directions: it is a guarantee stronger than anything we could implement, and it is also why a naive recorder produces a trace in which sign-ins happen without typing.

08

Limitations

We list these plainly because a collector's limitations determine what its datasets can be used for.

01 A screenshot does not respect the allow-list within the frame. This is the most important limitation in the tool. Scoping decides whether a frame is captured, not what is visible in it. If another application is on screen when an allowed action occurs, it is in the image. The mitigations are operational (close everything irrelevant first, review `report.html` before sharing) and operational mitigations depend on people. Masking non-target windows within the frame is the most valuable thing we have not built.

02 Only password fields are protected. A username, a one-time code typed into an ordinary field, an account number in a form: these are ordinary text and are recorded. The protection follows what the operating system marks as secure, which is narrower than what is sensitive.

03 The numbers here are from one machine. Single operator, single display configuration, a handful of applications, and audits in the tens of interactions rather than the thousands. They are sound as engineering measurements and are not a controlled study. Applications we have not tested may behave differently, and Electron applications in particular vary widely in what they publish.

04 Measured on macOS. The Windows version writes the same record, but it resolves elements against a different accessibility API, and the numbers in §6 have not been re-measured there. Mechanisms described as macOS-specific (Secure Event Input in particular) have their own Windows counterparts, which this paper does not cover.

- 05 Chrome's own interface cannot be enabled from outside.** The search recovers the target in the cases we measured, but it is a workaround for a condition we cannot fix, and a Chrome change could alter it.
- 06 The reconciliation rule is empirical.** "The field's value is a prefix of what was typed" came from real recordings, and every branch of it was written against an observed failure. It is not derived from a specification, and an application that violates it will produce wrong text. Both sources stay in the record so that this is detectable.
- 07 Coalescing loses within-field ordering.** One action per field edit is the right granularity for a planner, but a study of how people correct themselves mid-field would need the raw stream, which is summarised rather than kept in full.
- 08 10 fps can still miss a transient.** A tooltip or a menu that appears and disappears between frames is not in any observation. Video capture, as DuckTrack and AgentNetTool use, is strictly better on this axis, at the cost of a heavier dependency.
- 09 Unsigned distribution.** The macOS application is currently distributed ad-hoc signed, so recipients must clear the quarantine attribute by hand. Developer ID signing and notarisation would remove that step; it is a packaging task, not a design one.
- 10 Comparisons reflect the versions we read.** All four tools are actively developed. Table 1 is accurate as of the versions in this study and should be re-checked before being relied on.
- 11 Nothing here is legal advice.** The data-protection policy shipped with the tool is operational guidance written by the people who built it. Screen and keystroke capture touches wiretapping and electronic-monitoring statutes in several jurisdictions, some requiring all-party consent and some carrying criminal penalties. Counsel decides that, not this document.
- 12 We have not measured downstream model quality.** Everything here is a property of the dataset: naming accuracy, box tightness, record size. That better labels produce a better planner is the premise of the work, not a result of it.

Conclusion

Four open collectors agree on how to record human-computer interaction, and the parts they agree on are right: pair the observation backwards, never block the input hook, write asynchronously. Where they fall short is the label (which control was acted on, what was actually typed) and what they decline to record.

DesktopTrace changes four things. It enforces application scoping at capture time, so an unlisted application produces nothing; no tool we evaluated has this. It records a password entry as stars and the password never at all, so a sign-in stays visible in the trace while its content does not exist in the data. It treats the platform hit test as a first step rather than an answer, adding a bounded upward walk and an unpruned downward search (which takes exact target naming from 13/21 to 21/21 in Chrome and narrows the median recorded box from 119 points to 40) and, because names come from the tree rather than the pixels, it names the icons that a model looking at the screen can only guess at. And it reconciles typed text between the field's own value and the keystrokes, because each is wrong exactly where the other is right.

CHANGE 1

Application scoping at capture time

CHANGE 2

A password entry recorded as stars, the password never

CHANGE 3

The hit test as a first step rather than an answer

CHANGE 4

Typed text reconciled between field value and keystrokes

The limitation we would most like to remove is the first one in §8: scoping controls whether a frame is taken, not what is visible inside it. Until that is solved, the strongest claim a collector of this kind can honestly make is that it records less than it could, which is still a claim none of the alternatives is currently in a position to make.

An allow-list checked before an action is constructed has no recall: the data was never made.

References

1. PC Agent / PC Tracker: a human-computer interaction collector and agent framework for training computer-use models. Source examined: `tracker/` , macOS accessibility path.
2. OpenAdapt: open-source process automation by demonstration. Source examined: `openadapt_capture/` (`recorder.py` , `events.py` , `processing.py`) and the `legacy/` scrubbing providers.
3. DuckTrack: multimodal computer interaction data collection via OBS Studio. Source examined: `ducktrack/recorder.py` , `ducktrack/obs_client.py` .
4. AgentNetTool: the annotation tool behind the AgentNet dataset, XLang Lab. Source examined: `api/core/recorder.py` , `api/core/action_reduction/` .
5. Microsoft Presidio: data protection and de-identification SDK (used by OpenAdapt's scrubbing providers).
6. Apple Inc., Accessibility API reference: `AXUIElement` , `AXUIElementCopyElementAtPosition` , `AXUIElementCopyMultipleAttributeValues` .
7. Apple Inc., Quartz Event Services: `CGEventTap` , and Carbon `IsSecureEventInputEnabled` .
8. Digital Personal Data Protection Act, 2023 (India).
9. Regulation (EU) 2016/679, General Data Protection Regulation.
10. California Consumer Privacy Act, as amended by the California Privacy Rights Act.

A

Appendix A: a sample record

This is the complete record DesktopTrace wrote for the Settings click in Figure 2, reproduced from `actions.jsonl` . The only change is the window title, which contained an email address and has been shortened for publication. Geometry is in points; the `frame`

block carries the display scale and origin needed to convert to the screenshot's pixels.

A session is one folder:

```
session-20260820-225503/
├─ actions.jsonl      one JSON record per action (below)
├─ session.json       settings the session ran with, counts dropped and
redacted
├─ config.json        the configuration file in force
├─ report.html        every action with its frame and target box, for review
├─ frames/
│   └─ action-000002-20260820-225506-539.jpg
│   └─ ...
│   └─ action-000007-20260820-225518-942.jpg ← the frame for this record
```

The record. Reading it top to bottom: `element` is what was acted on, `raw_element` is what the hit test returned (the gear image) and `resolution` says the resolver climbed one level to reach a named, interactive element and considers the result reliable. `timings_ms` shows the whole lookup took under 5 ms, and `frame.age_ms` that the screenshot was taken 11 ms before the click.

```
{
  "sequence": 7,
  "observation": {"title": "Inbox - [address removed] - Mail - Google Chrome", "tab":
null},
  "action": {
    "type": "click", "description": "click at (1428, 161)",
    "timestamp": "2026-08-20T17:25:18.953915+00:00", "point": {"x": 1427.8, "y":
161.3},
    "button": "left", "event_count": 2
  },
  "element": {
    "role": "AXPopUpButton", "subrole": null, "name": "Settings", "name_source":
"description",
    "bbox_xyxy": [1404.0, 138.0, 1444.0, 178.0], "role_description": "pop-up button",
    "enabled": true, "focused": false, "selected": false, "dom_class": "FH"
  },
  "app": {"name": "Google Chrome", "bundle_id": "com.google.Chrome", "pid": 33838},
  "resolution": {"rule": "interactive+named", "levels_up": 1, "reliable": true},
  "context": "Settings",
  "raw_element": {
    "role": "AXImage", "subrole": null, "name": "image", "name_source":
"role_description",
```

```

    "bbox_xyxy": [1412.0, 146.0, 1436.0, 170.0]
  },
  "ancestors": [
    {
      "role": "AXPopUpButton", "subrole": null, "name": "Settings", "name_source":
"description",
      "bbox_xyxy": [1404.0, 138.0, 1444.0, 178.0]
    }
  ],
  "timings_ms": {"hit_test": 1.36, "attributes": 1.158, "ancestors": 2.442, "total":
4.959},
  "frame": {
    "path": "frames/action-000007-20260820-225518-942.jpg",
    "captured_at": "2026-08-20T17:25:18.942694+00:00", "age_ms": 11.2, "display_id":
1,
    "display_origin": {"x": 0.0, "y": 0.0}, "scale": 2.0,
    "image_size_px": {"width": 3420, "height": 2214}
  }
}

```

Neurologic AI · DesktopTrace · September 2026. Measurements in this paper were taken on a single development machine and are reported as engineering results, not as a controlled study. Nothing here is legal advice.