When Tracking Learns: Learning-Based Browser Fingerprinting Under Anti-Fingerprinting Defenses

Xu Lin
Washington State University
Pullman, WA, USA
xu.lin@wsu.edu

Abstract

Browser fingerprinting has often been framed through a deterministic lens: fixed probes are expected to yield outputs stable enough to be matched across sessions, and anti-fingerprinting defenses are commonly assessed by whether they disrupt such matching. We argue that this view is incomplete once tracking learns. A modern tracker can treat browser fingerprinting as multi-sample inference under variability, aggregating repeated observations and learning device-specific structure that persists despite randomized probes, system updates, and injected noise. We instantiate this perspective on HTML5 canvas as a representative case and present `LearnedFP`, a learning-based fingerprinting system that deliberately introduces controlled variability through server-seeded randomized renderings and enables device identification via pixel-level modeling. We evaluate `LearnedFP` on a crowdsourced longitudinal dataset of 3,896 devices, collected in two campaigns 116–139 days apart and hosted on different sites. `LearnedFP` improves temporal Top-1 identification by 2.8× over the state-of-the-art non-deterministic method when matched for collection time. In contrast, traditional deterministic matching loses 63% of returning devices. We further evaluate `LearnedFP` against anti-fingerprinting protections deployed by browsers (e.g., Safari, Firefox, Brave, and Tor) and popular anti-fingerprinting extensions. Bounded perturbations leave learnable residual structure, and counterintuitively, defenses introduce defense-specific artifacts that serve as fingerprinting signals. Our results suggest that defenses should be evaluated by learnability rather than output instability alone, and motivate more robust mitigation strategies.

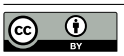
CCS Concepts

• **Security and privacy → Browser security**; *Web application security*.

Keywords

Browser Fingerprinting, Privacy, Web Tracking

ACM Reference Format:
Xu Lin. 2026. When Tracking Learns: Learning-Based Browser Fingerprinting Under Anti-Fingerprinting Defenses. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3846587



This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '26, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846587

1 Introduction

Online tracking has been pervasive across the web ecosystem for more than two decades [32]. As regulations, browser policies, and user awareness increasingly constrain stateful identifiers such as cookies, tracking has shifted toward stateless mechanisms that do not require persistent client-side storage [43]. Browser fingerprinting has therefore become a major stateless tracking technique [15, 30], enabling persistent identification across sessions, websites, and privacy modes, often without user awareness [48]. Mozilla has reported that, without anti-fingerprinting protections, more than 60% of users may appear uniquely identifiable to websites [40]. In response, modern browsers and privacy tools have deployed anti-fingerprinting defenses that aim to preserve web functionality while reducing cross-session linkability. Because many fingerprinting surfaces are also needed for legitimate applications, deployed defenses rarely block the underlying APIs outright. Instead, they mediate what scripts observe: perturbing rendered outputs, rounding or normalizing reported values, spoofing selected attributes, or gating specific extraction paths [9, 57]. These mechanisms are designed to disrupt *deterministic* matching, where a tracker expects the same probe to produce the same value across visits.

This paper argues that such defenses can provide an incomplete sense of protection once tracking *learns*. A modern tracker need not operate through exact matching. It can instead treat fingerprinting as inference over a distribution of observations: collecting multiple samples, tolerating variability, and learning device-specific structure that persists beneath randomized probes, injected noise, software updates, and routine configuration changes. In this setting, output instability is no longer protective; it becomes part of the learning process. High-dimensional outputs, such as canvas pixel buffers, audio waveforms, and GPU readbacks, can retain rich residual signal even after bounded perturbation, and a learning-based adversary can aggregate repeated measurements and model the variation introduced by these defenses rather than be defeated by it. Moreover, anti-fingerprinting defenses are difficult to apply coherently across the browser stack. In practice, defenses are piecemeal: one API path may be perturbed, another sanitized, and a third left unchanged. The resulting inconsistencies between raw buffers, encoded exports, partial reads, full reads, and offscreen renderings can themselves become stable, browser- or extension-specific fingerprinting features.

These observations motivate a shift in how anti-fingerprinting defenses are evaluated. The relevant question is not whether outputs change, but whether the residual signal remains learnable under realistic adversarial budgets, and whether defenses remain coherent under adaptive, cross-path probing. To examine this, we

instantiate a learning-based fingerprinting system, LearnedFP, using HTML5 <canvas> as a representative high-dimensional surface. Canvas is ubiquitous, widely used by legitimate applications, and already targeted by many defenses, making it an ideal case study. Unlike traditional canvas fingerprinting, which renders a fixed image and hashes the exported result, LearnedFP deliberately introduces controlled variability: the server generates a freshly randomized rendering challenge for every sample, so a single visit yields many independently varying canvas observations; the tracker reads raw RGBA pixel buffers rather than encoded image hashes, and the backend learns device-specific structure from these pixel-level observations across multiple probes.

We evaluate LearnedFP along three axes: learnability under variability, robustness to deployed defenses, and generality beyond canvas. To test learnability under variability, we collect a large-scale crowdsourced longitudinal dataset of 3,896 unique devices across two campaigns separated by 116–139 days and deployed on two distinct sites. Randomized outputs remain strongly linkable under learning-based inference. Specifically, LearnedFP improves Top-1 temporal re-identification by 2.8× over DRAWNAPART, the state-of-the-art non-deterministic GPU fingerprinting baseline, when matched for collection time. By contrast, FingerprintJS [20], a widely used library that derives a visitor identifier by hashing browser attributes, loses nearly two-thirds of returning devices when those identifiers change across campaigns.

To assess effectiveness against deployed defenses, we evaluate LearnedFP against major browsers and privacy tools, including Safari, Firefox, Brave, Samsung Internet, Tor, and popular canvas-protection extensions. We characterize how each defense modifies canvas outputs across raw buffers, encoded exports, repeated reads, partial and full reads, and offscreen rendering paths. Many defenses perturb only bounded portions of the signal, leave some channels untouched, or introduce stable inconsistencies across extraction paths. LearnedFP remains effective under most evaluated defenses, and, counterintuitively, actively defended devices can become more identifiable than undefended ones, because defense-specific perturbation patterns themselves act as stable fingerprints. Tor provides the strongest protection by requiring user permission for extraction, but this can disrupt applications that rely on canvas readback, and make Tor users detectable as a cohort.

To assess whether learning-based fingerprinting generalizes across browser surfaces, we instantiate the same workflow on Web Audio, where the response is a numeric waveform rather than a rendered image. Randomized audio outputs likewise expose learnable cross-session structure. The risk is not tied to pixels, fonts, or any single API; it can arise whenever web content can issue controlled challenges and repeatedly read back rich, implementation-dependent responses.

Our work makes the following contributions:

- We reframe browser fingerprinting as multi-sample inference under variability, where each observation is a sample from a device-specific response distribution rather than a value to be matched. We instantiate this view as LearnedFP, a learning-based system using server-seeded randomized renderings, raw RGBA pixel-level modeling, and multi-key fusion.
- Through a longitudinal evaluation of 3,896 devices over a nearly four-month cross-site gap, LearnedFP substantially outperforms the state-of-the-art non-deterministic method and remains effective in scenarios where deterministic fingerprinting fails systematically.
- We characterize deployed defenses across major browsers and extensions and show that, counterintuitively, defenses can make devices more identifiable, with defense-specific artifacts themselves serving as stable fingerprinting signals.
- We extend the analysis to Web Audio, showing the threat applies more broadly to browser surfaces exposing high-fidelity readback that can be sampled repeatedly.
- We derive capability-focused mitigation principles and implement a prototype defense that limits high-fidelity extraction, enforces cross-path coherence, canonicalizes invisible channels, and constrains repeated silent sampling.

We have disclosed our findings to affected browser vendors. Our results suggest that durable privacy protection requires limiting what scripts can extract from the browser, rather than only destabilizing what they observe. We release our code to support future work.[1]

2 Background

Browser fingerprinting enables websites to track users by collecting signals that reflect the browser, the underlying platform, and its rendering stack. Unlike cookie-based tracking, it does not require persistent client-side storage, making it particularly attractive when cookies are restricted or partitioned. Driven by advertising incentives, fingerprinting is now widely deployed. Its prevalence on the top 10k websites grew from 0.4% in 2013 [43] to 25% by 2021 [22], with recent measurements showing large-scale deployment on authentication pages [49].

In response, browser vendors have deployed anti-fingerprinting defenses [41, 58], and prior work has proposed techniques for detecting and blocking fingerprinting scripts [2, 42]. Despite these efforts, fingerprinting remains prevalent. The W3C acknowledges that fully eliminating fingerprinting against a determined adversary is implausible [44]. The core difficulty is that many fingerprinting techniques rely on the same APIs and rendering paths that support legitimate web functionality. Aggressive blocking risks breaking page layouts or disabling interactive components; randomization-based defenses may introduce inconsistencies that impair usability. As a result, deployed defenses typically aim to *disrupt* rather than *eliminate* fingerprinting signals. They perturb rendered outputs, round or normalize reported values, or spoof selected attributes, while still returning plausible responses.

The HTML5 <canvas> element is one of the most extensively studied fingerprinting surfaces [29]. Canvas allows scripts to draw graphics and extract the result via APIs such as `getImageData()`, `toDataURL()`, and `toBlob()`. Subtle differences in the rendered output arise from the browser engine, graphics stack, font and text rendering pipeline, and color management, making canvas one of the higher-entropy browser attributes used for fingerprinting [29, 34]. A conventional canvas fingerprinting workflow renders a fixed image, exports the resulting bitmap, and derives a stable identifier

[1] https://github.com/xlin29/learned-fp

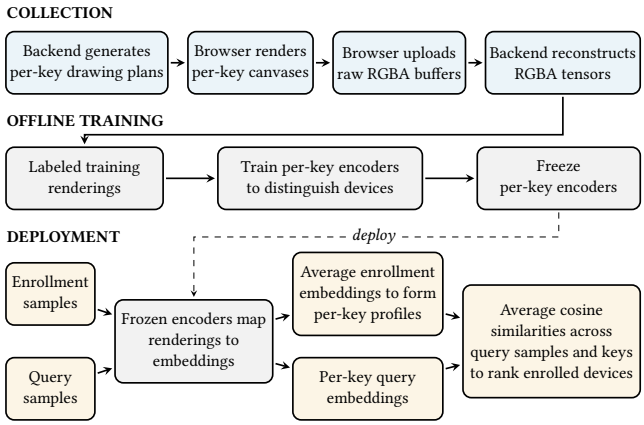


Figure 1: The LearnedFP pipeline. The collection row shows how enrollment and query samples are collected.

from the output [39]. Because canvas is also heavily used for benign functionality, such as games, image editing, and interactive graphics, browsers cannot easily disable it without affecting usability, making it a common target for anti-fingerprinting defenses. We use canvas as our primary case study because it provides a concrete, widely deployed example of a broader pattern: browser surfaces that allow web content to issue controlled probes and read back rich, device-specific responses.

3 Threat Model

We consider a privacy-invasive web service, such as an advertising network, third-party tracker, or analytics provider that seeks to track devices across the web. The adversary aims to identify any device that visits a site where its code is deployed, including devices it has never observed before, and to re-identify it when it returns, even after browser updates, configuration changes, or anti-fingerprinting protections. The fingerprinting payload is delivered as ordinary web content. It may execute as first-party code on a tracker-controlled site, as a third-party script or iframe, or through a colluding content provider. The adversary cannot escape the browser sandbox or same-origin policy; it observes only outputs that its own scripts can read back through standard APIs. Anti-fingerprinting defenses are within scope: privacy-focused browsers, private-browsing modes, and extensions.

Unlike adversaries that rely on exact matching of deterministic fingerprints, our adversary aggregates evidence across multiple samples on the server side. Such infrastructure is readily available to commercial trackers and advertising networks.

4 LearnedFP Design

This section presents LearnedFP, a learning-based browser fingerprinting system for HTML5 <canvas> that treats each output as one sample from a device-specific response distribution.

The central design choice in LearnedFP is to make variability part of the fingerprinting process. The tracker varies probe parameters across observations and relies on persistent device-specific structure in the resulting outputs. A single output is not expected to be stable; instead, trackability emerges from aggregating multiple observations and learning which pixel-level patterns remain predictive under controlled variation.

We use the following terminology. A *key* is a rendering family, such as an emoji grid, a randomized text layout, a moiré line field, or a shadow-compositing probe. LearnedFP uses nine keys. A *sample* is one complete rendering bundle generated from a fresh server seed, containing one canvas rendering for each key. The tracker's budget is determined by the number of samples collected during *enrollment* (when a device is first profiled) or *query* (when a return visit is linked against existing profiles). Figure 1 summarizes how these samples move through collection, offline training, and deployment.

4.1 Canvas Outputs and Rendering Keys

LearnedFP operates on raw canvas pixel buffers rather than on encoded exports. For each rendering, the client extracts the Uint8ClampedArray returned by getImageData(), where each pixel is represented by four 8-bit channels $(R, G, B, A) \in [0, 255]^4$. Each output is therefore a three-dimensional tensor indexed by width, height, and channel, preserving the spatial and channel-level structure that hash-based fingerprints discard.

The nine keys (Figure 2) exercise complementary parts of the rendering stack. Text and emoji keys (the four Emoji* keys, GlyphSet, and RandChar) stress Unicode processing, font fallback, glyph rasterization, and emoji composition. Geometric keys (GradStep, MoireLine, ShadowBlur) stress antialiasing, compositing, shadow kernels, gradient interpolation, and floating-point rounding. Figure 2 reports the number of possible configurations for each key, i.e., how many distinct probes the tracker can generate. This reflects probe diversity rather than fingerprinting entropy.

We initially considered a broader feature set, including probes used by existing fingerprinting libraries, mathematical curves, geometric primitives, gradients, and WebGL-based 3D renderings. Empirical testing led us to the current nine-key design, which provides strong discriminative performance while remaining lightweight. We omit 3D rendering because 2D canvas probes achieve comparable discriminative power with lower overhead and broader browser compatibility.

4.2 Fingerprinting Workflow

Step 1: Server-side probe generation. At the beginning of each visit, the backend generates a fresh session nonce. For each key, it derives a drawing plan by mapping pseudorandom values generated from the nonce into the key's parameter ranges. For example, an EmojiFace rendering selects 36 emojis from a pool of 569 candidates, with the font size, x/y offset, and rotation angle of each emoji sampled independently. This gives the tracker control over probe variation.

Step 2: Client-side rendering. The client component executes as ordinary web content. Given the drawing plans from the server, it renders each key on a small canvas of fixed resolution (e.g., 100×100) and extracts the raw RGBA buffer using getImageData(). Rendering can occur off-screen or on hidden canvases.

Step 3: Serialization and upload. The client serializes each raw RGBA buffer as bytes, optionally compresses and batches multiple samples, and uploads the result through standard HTTP endpoints.

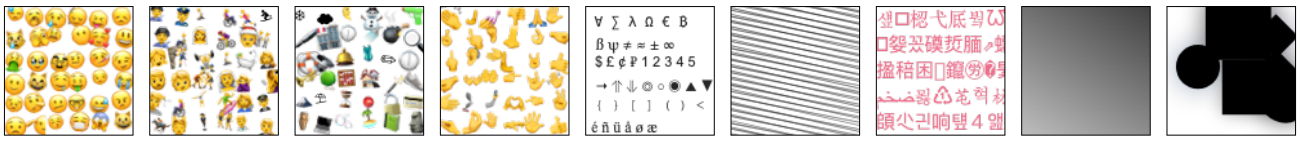


Figure 2: Rendering keys used by `LearnedFP`. Each key shows an example output, with the configuration-space size in parentheses.

Collection cost depends on the number of samples and keys, the canvas resolution, and the batching policy.

Step 4: Backend ingestion. Upon receipt, the backend decompresses the data and reconstructs each rendering as a raw RGBA tensor. `LearnedFP` operates on these raw buffers throughout. During collection, the four Emoji keys and GlyphSet carry a session-specific marker in the RGB channels of fully transparent pixels to detect stale or malformed uploads; the RGB values of pixels with $A = 0$ are zeroed before feature extraction and never enter the model. These five keys bypass `toBlob()` and are uploaded as raw bytes; the remaining four use lossless PNG.

4.3 Learning-Based Cross-Visit Linkage

`LearnedFP` treats cross-visit linkage as a retrieval problem under variability. The backend maps each rendering to an embedding, constructs profiles from enrollment samples, and ranks enrolled devices by similarity for each query.

Per-key encoders. For each rendering key, the backend trains a CNN encoder that maps an RGBA tensor (4×100×100) to a 256-d embedding. Training treats each device as a class and minimizes cross-entropy. At deployment, the encoders are frozen and their embeddings are used for linkage. Appendix C gives the full implementation.

Profile ranking. For each device, the backend constructs one profile for each rendering key by averaging the embeddings of its enrollment samples. On a return visit, the same frozen encoders map the latest observations to embeddings for each key. The backend computes cosine similarities between these query embeddings and the corresponding profiles, averages the scores across query samples and the nine keys, and returns the highest-scoring device as the predicted identity.

Robustness through fusion. Modeling each key separately improves robustness because different keys capture different parts of the rendering stack. If a system update or defense affects some keys more than others, the less affected keys still contribute to the final score.

5 Identification Effectiveness

This section evaluates how effectively `LearnedFP` identifies devices under realistic tracking scenarios despite the intentional output variability built into its design. The evaluation has four parts. First, we compare `LearnedFP` head-to-head against DRAWNAPART [25], a state-of-the-art GPU-timing fingerprinting baseline, under single-visit enrollment and matched collection-time budgets. Second, we evaluate `LearnedFP` at the full deployment scale. Third, we contrast

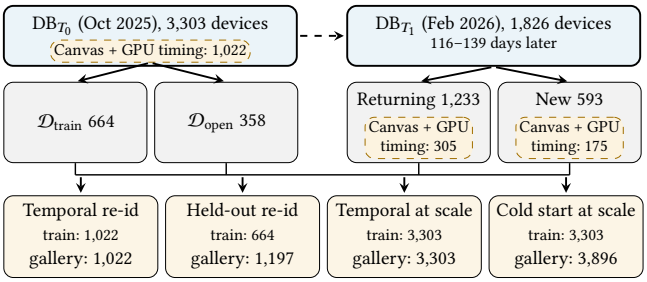


Figure 3: Dataset structure and evaluation scenarios. Dashed insets mark the devices carrying both `LearnedFP` and DRAWNAPART data.

`LearnedFP` with FingerprintJS [20] to show how deterministic fingerprinting behaves across the same temporal gap. Fourth, we evaluate FP-STALKER [61] with DRAWNAPART and with `LearnedFP`.

5.1 Dataset and Protocol

Campaigns and units. We recruited participants through the crowdsourcing platform Prolific [45] and conducted two collection campaigns separated by 116–139 days. The initial campaign DB$_{T_0}$ (Oct 5–19, 2025) provides single-visit enrollment data. The follow-up campaign DB$_{T_1}$ (Feb 12–21, 2026) provides data from returning and new devices, with each device contributing up to four visits. To approximate cross-site tracking, DB$_{T_1}$ was hosted on a different site rather than under the same first-party domain as DB$_{T_0}$.

A *visit* is one user encounter with the collection site. During both campaigns, we collected three types of measurements: `LearnedFP` samples, DRAWNAPART traces (Chromium-based browsers only), and FingerprintJS (v4.6.2) attributes [20] as a deterministic baseline. Figure 3 summarizes the dataset and evaluation settings. Temporal re-identification enrolls in DB$_{T_0}$ and queries in DB$_{T_1}$; the held-out setting follows the same timeline but evaluates devices excluded from encoder training. Cold-start tracking uses DB$_{T_1}$ for both enrollment and query.

Participant recruitment. We recruited participants from Prolific's Standard sample with no demographic, geographic, or device restrictions. The task involved a passive website visit, and participants were compensated at $12 per hour. For the follow-up campaign, we recontacted prior participants using their PIDs and recruited additional participants. After excluding one participant who later withdrew from the study, the analyzed cohort contains 3,896 PIDs. Prolific demographic data show participants from 50 countries; the largest groups were from South Africa (25.5%), the

United Kingdom (20.1%), and the United States (12.7%). The median age was 34 years. Among the 3,868 participants with sex data, 50.1% reported female, 49.7% male, and 0.2% preferred not to say.

Participants and devices. A PID identifies a participant rather than a device, and the same participant may use multiple devices. For each PID, the device used at initial enrollment serves as the reference device; subsequent visits are retained only if they are consistent with that same device, while visits from a different or replacement device are excluded. Thus, each PID is associated with one enrolled device, without assuming that the participant uses only that device.

Same-device labels. Because DB_{T_0} and DB_{T_1} were deployed on different sites, no first-party cookie links records across campaigns. A matching PID therefore identifies only a candidate return, which we retain only if the two records pass a metadata-consistency check. The check computes a weighted agreement score over metadata fields. The weights were guided by prior longitudinal studies [33, 61], with greater weight given to hardware- and system-dependent fields. Record pairs scoring at least 0.80 are retained. The FingerprintJS canvas attribute is excluded from the score, so the canvas signal evaluated by `LearnedFP` cannot influence the labeling decision. Neither the PID nor the consistency score is provided to any evaluated method as an input or matching feature.

Of the 2,043 PIDs observed in both campaigns, the check excludes 810 (39.6%), leaving 1,233 returning devices. The rejected pairs are dominated by substantial system changes: 94.2% report different operating-system platforms or differ on at least two hardware descriptors, while no pair is rejected for settings changes alone. Rejected DB_{T_1} records are dropped from the query set rather than relabeled as new devices, while their DB_{T_0} counterparts remain in the enrollment gallery; new devices are limited to PIDs not observed in DB_{T_0}. Using cookie-linked records within DB_{T_1}, the 0.80 threshold accepts all 17,172 known same-device pairs and rejects 99.74% of 6.4 million clear negative pairs, defined as pairs from different participants that differ in operating-system platform or on at least two hardware descriptors.

Data splits. The comparison against DRAWNAPART uses the 1,022 devices with both canvas samples and GPU-timing traces. Following the split protocol used by DRAWNAPART [25], we partition these devices by identity into $\mathcal{D}_{\text{train}}$ (65%) and $\mathcal{D}_{\text{open}}$ (35%). Within the training population used by each experiment, we split DB_{T_0} outputs 80/20 for training and model selection. No DB_{T_1} data is used to train an encoder or select its hyperparameters; DB_{T_1} outputs enter only as gallery entries and queries. Normalization statistics are computed on the training devices alone and applied unchanged at inference time.

Metrics. We report Top-n accuracy, compared with the uniform random-guess baseline n/G, where G is the gallery size. Lift is accuracy divided by this baseline, which allows comparison across gallery sizes.

5.2 Compared Systems and Cost

Table 1 summarizes the comparison between DRAWNAPART and `LearnedFP` across fingerprinting modality, browser coverage, variability, and enrollment cost. We next describe each method's implementation and the matched evaluation setup.

Table 1: Comparison of DRAWNAPART and `LearnedFP`. Enrollment times are client-side medians, excluding upload.

Aspect	DRAWNAPART	`LearnedFP`
Measurement modality	GPU timing	Pixel rendering
Browser coverage	Chromium-only	Cross-browser
Variability	Hardware-induced	Server-controlled
Enrollment data	28 traces	100 samples
Enrollment time	42 s	10 s
Enrollment upload	423 KB	23.1 MB

State-of-the-art method. DRAWNAPART [25] identifies devices from GPU timing side channels. Each timing trace is mapped by a CNN to an embedding for nearest-neighbor matching. We built the baseline from the authors' released code [26]. Trace collection uses the routine from their browser extension, which we execute directly in the page. The encoder is their Keras CNN, trained on our DB_{T_0} data following their reported procedure: semi-hard triplet fine-tuning with batch size 1,024 for 30 epochs, and checkpoint selection by best validation 1-NN accuracy. Three things are necessarily ours, because their release omits them: the trained weights, the fitted scaler, and the number of epochs for the classifier stage. Because our evaluation pipeline is implemented in PyTorch, we ported the trained Keras model, reproducing its embeddings to within $2{\times}10^{-7}$.

Single-visit enrollment. To model a realistic tracker that must build an enrollment profile from one encounter, we enroll each device from a single DB_{T_0} visit: DRAWNAPART collects 28 traces (four 7-trace blocks), while `LearnedFP` collects 100 samples. The trained encoders remain frozen throughout evaluation.

Cost. We time both methods on visits in which both collections were completed: 100 `LearnedFP` samples and 28 DRAWNAPART traces. All reported times are medians. `LearnedFP` takes 10 s to render, while DRAWNAPART takes 42 s to measure. Neither method blocks the page: both run offscreen and upload in the background. Because the two collections run in the same visit and contend for the same GPU, these timing measurements are conservative upper bounds for either method deployed on its own. Upload volumes are bytes received by the server.

5.3 Head-to-Head Results

Because a GPU trace and a canvas rendering differ in kind, we compare the methods within a matched collection window rather than by unit count. A DRAWNAPART block contains seven sequentially collected traces and is uploaded as one unit. For visualization, intermediate trace and sample budgets are distributed over the measured block duration; the full-block endpoint corresponds to the observed server-side completion time. For each device, we count how many `LearnedFP` samples the server has stored by the time that device's block arrives and report the median over the devices evaluated in each figure. Client-side rendering does not bound this count; upload does. The endpoint comparison therefore matches collection time while accounting for `LearnedFP`'s larger upload volume (Table 1).

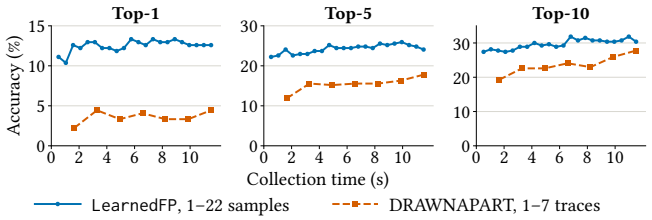


Figure 4: Temporal re-identification against a 1,022-device gallery; the base rates are 0.098%, 0.49%, **and** 0.98%.

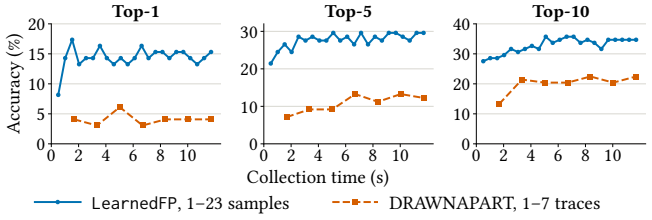


Figure 5: Held-out re-identification against a 1,197-device gallery; the base rates are 0.084%, 0.42%, **and** 0.84%.

Temporal re-identification. This experiment models tracking of enrolled devices over time: a tracker maintains an enrollment gallery and ranks each return query against the full gallery. We evaluate LearnedFP and DRAWNAPART on the same 1,022-device enrollment gallery, with returning queries drawn from DB_{T_1}. Among the 305 returning devices available to both methods, every point uses the same 270 devices that provide at least 22 LearnedFP samples. On these devices, one DRAWNAPART block of seven traces completes in 11.5 s, and 22 is the median number of LearnedFP samples received within the same window.

Figure 4 compares the two methods within the matched collection window. At Top-1, LearnedFP reaches 12.59% versus 4.44% for DRAWNAPART; at Top-5, 24.07% versus 17.78%; and at Top-10, 30.37% versus 27.78%. The gap narrows as the cutoff widens, but LearnedFP retains a larger advantage at the top rank. Because a tracker must ultimately commit to one identity, Top-1 is the most operationally relevant metric. The methods also differ in how much evidence they need. LearnedFP's first sample arrives at 0.4 s and already yields 11.11% Top-1, more than twice what DRAWNAPART reaches with its full seven-trace block.

Held-out re-identification. This experiment models the deployment case in which a tracker cannot retrain for every device it encounters. The encoders are trained once on $\mathcal{D}_{\text{train}}$ and then applied unchanged to devices outside that set. Of the 105 returning devices in $\mathcal{D}_{\text{open}}$ available to both methods, every point uses the same 98 that provide at least 23 LearnedFP samples. They are ranked against a 1,197-device gallery: the 1,022 enrolled devices plus 175 devices first seen in DB_{T_1}, also available to both methods, added as distractors so that the candidate pool is not restricted to devices known to return. On these devices, one DRAWNAPART block completes in 11.7 s, and 23 is the median number of LearnedFP samples received within the same window.

In Figure 5, LearnedFP reaches 15.31% versus 4.08% for DRAWNAPART at Top-1; at Top-5, 29.59% versus 12.24%; and at Top-10, 34.69% versus 22.45%. LearnedFP identifies devices excluded from encoder training at 183× the base rate, so its advantage does not depend on memorizing the evaluated devices. Unlike Figure 4, this figure uses encoders trained on $\mathcal{D}_{\text{train}}$, the 664-device subset that excludes every device evaluated here. Under this condition, the Top-10 convergence seen there does not appear. The absolute accuracies are not directly comparable across the two figures because they use different galleries and device sets. LearnedFP's first sample arrives at 0.4 s and already yields 8.16% Top-1, twice what DRAWNAPART reaches with its full block. Randomized canvas outputs are therefore not mere noise. They retain device-specific structure that can be recovered for identification.

Stricter labels. We raise the metadata-consistency threshold from 0.80 to 1.00, retaining a return only when all scored fields match across campaigns. This leaves 152 returns; 133 have both a DRAWNAPART block and at least 22 LearnedFP samples and are included in the comparison. Browser version is not scored, and 124 of the 133 report a browser update between campaigns. Within the same 11.5 s window, LearnedFP rises from 12.59%, 24.07%, and 30.37% under our primary labels to 17.29%, 31.58%, and 36.84% at Top-1, Top-5, and Top-10, while DRAWNAPART rises from 4.44%, 17.78%, and 27.78% to 5.26%, 20.30%, and 32.33%. The stricter rule selects an easier subset, so both methods improve, but the comparative conclusion is unchanged, with LearnedFP ahead of DRAWNAPART at every cutoff.

Baseline validation. We validate our DRAWNAPART baseline against the closest comparable setting in the original paper: one-shot matching within a single collection period, where DRAWNAPART reaches 5.44%, 14.10%, and 19.95% at Top-1, Top-5, and Top-10 across intervals from four hours to 52 days. In Figure 4, by contrast, a single seven-trace block must match an enrollment made 116–139 days earlier, more than twice their longest interval. Our baseline, built from their released code, reaches 4.44%, 17.78%, and 27.78%, one percentage point lower at Top-1 and higher at Top-5 and Top-10. Their setting is harder in one respect, a lower base rate (0.05% versus our 0.098%). Ours is harder in two others: their network is trained on devices with more than 196 traces each, while our dataset provides 28 per device; and 22% of our devices are mobile, which we identify at about half the desktop rate, whereas their traces come from desktop Chromium browsers.

Insight 1: Variability is not noise; it is a learnable tracking signal.

5.4 Scalability

We next test whether LearnedFP scales to the full population: 3,303 devices in DB_{T_0}, plus 1,233 returning and 593 new devices in DB_{T_1}. For this experiment we do not tune hyperparameters per key. We use one fixed configuration for all nine encoders; on this population, its Top-1 accuracy differs from per-key Bayesian search by 0.45 percentage points (Appendix C). Using a fixed configuration avoids repeated per-key tuning at this scale, and the small accuracy gap indicates that LearnedFP is robust to hyperparameter choice.

Table 2 reports LearnedFP performance at scale. LearnedFP remains effective at every setting: 10–13% Top-1 on temporal re-identification against a 3,303-device gallery, and 15–24% Top-1 on cold-start tracking against a 3,896-device gallery. All settings exceed

Table 2: Evaluation of LearnedFP on all available devices. k denotes query budget. Random Top-1 baselines: 0.030% (gallery 3,303) for temporal re-identification; 0.026% (gallery 3,896) for cold-start.

Scenario	k	#Devices	Top-1 (%)	Top-5 (%)	Top-10 (%)
Temporal re-identification	1	1,233	10.22	19.95	26.28
	5	1,219	11.57	22.48	28.96
	100	426	13.38	28.40	34.51
Cold-start	1	593	15.35	30.19	34.57
	5	463	20.30	34.99	40.39
	20	113	23.89	48.67	53.10

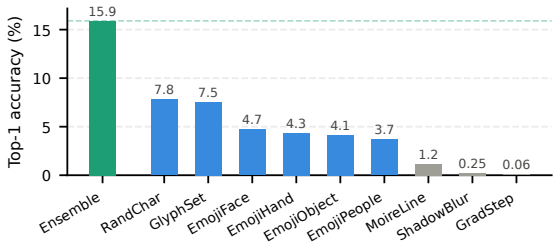


Figure 6: Per-key Top-1 accuracy. The dashed line marks the 9-key ensemble (random baseline 0.030%).

random by hundreds of times (338–931× Top-1 lift), confirming that the learned signal persists at deployment scale.

Cold-start accuracy is higher despite the larger gallery. The main difference between the two settings is temporal drift: cold-start enrollment and query occur within the same campaign, so browser version, OS, fonts, and GPU driver remain largely unchanged, whereas temporal re-identification queries the same device roughly four months later, allowing browser and driver changes to accumulate. The gap therefore reflects the added difficulty of longitudinal drift rather than a difference in encoder capability.

Per-key contribution. To isolate what each key contributes, we score the nine encoders separately within DB_{T_0}, where training and test samples come from the same campaign. Figure 6 shows a three-tier hierarchy. Text and glyph keys (`RandChar`, `GlyphSet`) provide the strongest individual signal at 7.5–7.8% Top-1, reflecting cross-platform differences in font rasterization and Unicode rendering. Emoji keys form a middle tier (3.7–4.7%), and geometric keys fall below 1.2%. Despite this heterogeneity, the 9-key ensemble reaches 15.90% Top-1, more than double the strongest individual key: weaker keys are not redundant but contribute complementary signal recovered through fusion.

5.5 Comparison with FingerprintJS

We evaluate how FingerprintJS [20], a deterministic fingerprinting method, handles the same tracking setup. FingerprintJS collects 41 attributes and derives a `visitorId` by hashing them together. Our analysis reveals three failure modes of deterministic fingerprinting in deployment.

Enrollment collisions. Even at enrollment, 7.18% of devices (237 of 3,303) share a `visitorId` with at least one other enrolled device; the largest collision group contains 21 devices. Because exact matching is binary, devices cannot be distinguished while they share a `visitorId`.

Temporal instability. Among 1,233 returning devices, 776 (62.94%) produce a different `visitorId`, causing FingerprintJS to misclassify them as new. This brittleness follows from the design of deterministic fingerprinting: any system-level change such as a browser update, OS upgrade, display reconfiguration, or font change can alter the `visitorId`. The underlying change is often subtle. Among the 496 returning devices with complete FingerprintJS canvas data in both campaigns, 34.9% exhibit pixel changes, with a median change affecting only 6% of pixels. Such subtle changes are catastrophic for hash matching, yet remain learnable for `LearnedFP`.

Cold-start false positives. Among the 593 cold-start devices, 14.17% produce a `visitorId` that collides with at least one other identity: 9.95% match an enrolled DB_{T_0} device (false-positive identification), and 8.60% match another cold-start device, with some devices falling into both categories. `LearnedFP`'s continuous embedding is not subject to these exact-match failure modes. Instead of an identifier to match, `LearnedFP` ranks each query against the full enrollment gallery and absorbs system-level changes through learned representations. At $k = 1$ on the 3,303-device gallery, `LearnedFP`'s Top-50 reaches 43.96%, already exceeding FingerprintJS's recovery rate; Top-100 reaches 52.15%, narrowing tracking to 3% of the gallery for over half of returning devices. With more samples, Top-50 grows to 52.35% at $k = 100$. Perfect identification is not required: a tracker can filter the remaining candidates with context such as site history, network location, or coarse device metadata. Two design properties enable this robustness: deliberate per-sample probe randomization encourages the model to learn device-level rendering structure rather than memorize a fixed output, and multi-key fusion absorbs changes confined to part of the rendering stack.

Insight 2: Deterministic fingerprinting fails systematically; learning-based identification persists.

5.6 Integration with FP-STALKER

Deterministic matching by exact equality fails once an attribute changes. A linker instead models how attributes evolve and reconnects a device across the change. FP-STALKER [61] is the state of the art in this setting, with a rule-based variant and a stronger hybrid that combines rules with a random forest. DRAWNAPART augments the hybrid with a GPU-timing embedding. We evaluate the hybrid with DRAWNAPART and with `LearnedFP`. The prior evaluation of FP-STALKER with DRAWNAPART used collection periods of 2–7 days. Our campaigns are 116–139 days apart, so each return reflects roughly four months of accumulated drift rather than a series of small changes.

Protocol. We evaluate a 1,022-device enrollment gallery and 305 returning devices for which both DRAWNAPART traces and `LearnedFP` samples are available. FP-STALKER either links a query to an enrolled device or assigns a new identifier, so we report the correct-link rate and false-link rate (FL). Following DRAWNAPART, we call the hybrid without an embedding nude FPS. We

Table 3: FP-STALKER with and without each embedding under two integration designs. Cells report mean correct-link rate ± standard deviation / mean held-out FL (%) across eight forests. The Nude column is calibrated separately for each integration design.

Canvas treatment	Nude	+ DRAWNAPART	+ LearnedFP
Short-circuit integration, published λ			
gate + feature	39.5 ± 2.4 / 0.37	42.6 ± 2.1 / 1.1	39.9 ± 2.5 / 0.9
feature only	41.3 ± 1.7 / 0.29	44.8 ± 3.9 / 0.8	47.2 ± 3.8 / 1.0
hash removed	43.9 ± 2.6 / 0.29	48.5 ± 3.3 / 0.9	51.1 ± 3.7 / 1.1
Feature integration, λ selected			
gate + feature	39.0 ± 2.6 / 0.77	38.9 ± 2.4 / 0.67	38.3 ± 1.7 / 0.93
feature only	44.8 ± 2.6 / 0.45	46.8 ± 3.7 / 0.40	51.2 ± 2.2 / 0.08
hash removed	48.6 ± 2.1 / 0.70	49.6 ± 1.9 / 0.59	58.3 ± 2.5 / 0.31

train the random forest on cross-campaign pairs, using five-fold cross-validation by participant with all pairs involving held-out participants excluded. Thresholds are selected on training folds to target 1% FL, then applied to held-out folds, and the published diff = 0.10 is fixed throughout.

Because FP-STALKER was developed on data collected nearly a decade ago, applying it to current browser fingerprints requires several adaptations. We replace Flash-derived attributes with JavaScript counterparts where available; for example, the rule written for Flash-based font fingerprinting is applied to the corresponding JavaScript font probe. We omit eleven attributes with no counterpart in our collection, nine of which FingerprintJS does not collect. These omitted attributes do not enter the rule checks, the change counter, or the feature vector, so they can neither reject a candidate nor contribute identity information.

Canvas treatments. Because FP-STALKER already uses a canvas hash, LearnedFP measures the same surface, whereas DRAWNAPART contributes a separate GPU-timing signal. We therefore evaluate three treatments of the canvas hash. In the original FP-STALKER design, canvas-hash equality acts as a hard gate: a candidate whose hash differs is discarded before reaching the random forest. The hash is also a model feature. DRAWNAPART's released integration [26] removes the gate but keeps the hash as a feature and in the change counter, while our updated configuration removes it from the linker entirely.

Embedding integration. For each canvas treatment, we evaluate the DRAWNAPART and LearnedFP embeddings under two designs that use the same FP-STALKER hybrid and differ only in where the embedding is consulted. Under DRAWNAPART's released *short-circuit integration*, the top candidate is accepted when embedding similarity exceeds ϵ, bypassing the forest's acceptance threshold λ, the top-two margin diff, and the tie check. Under *feature integration*, cosine similarities are appended to the forest's feature vector and the forest is retrained.

Results. Table 3 reports both integration designs. Under short-circuit integration, both embeddings improve the correct-link rate in all three treatments, but at a higher FL than nude FPS. DRAWNAPART helps more while the canvas gate remains, whereas LearnedFP provides the larger gains once the gate is removed. Under feature integration, neither embedding helps while the hash still gates the candidate set. Once the gate is removed, however, LearnedFP improves the correct-link rate by up to 9.7 percentage points while lowering FL, and outperforms DRAWNAPART on both measures.

The pattern follows from what the gate admits: hash equality restricts the candidate set to devices whose canvas did not change, so LearnedFP's canvas evidence largely repeats a condition the gate has already enforced, whereas DRAWNAPART's GPU-timing embedding provides a separate signal that is independent of the canvas gate. The three treatments change how the canvas hash enters the linker, not the underlying data collected by the tracker. The canvas hash is discriminative but temporally unstable: it changes for 102 of the 305 returning devices, and on these devices hash equality provides essentially no linking signal (AUC 0.497), whereas LearnedFP's pixel-level representation reaches an AUC of 0.855. Because a changed hash can eliminate the correct candidate before the linker considers other evidence, removing the gate raises the correct-link rate even in the nude column.

6 Learning Under Anti-Fingerprinting Defenses

The previous section showed that randomized canvas outputs remain learnable. We now ask whether deployed anti-fingerprinting defenses change this conclusion, addressing two questions. First, how do deployed defenses modify canvas outputs? Second, does LearnedFP remain effective when the browser or an extension actively perturbs the canvas surface?

6.1 Experimental Setup

Defense settings. We evaluate popular browsers that explicitly implement privacy-preserving countermeasures against canvas fingerprinting. Our test configurations include Brave with its default protections, Firefox with Enhanced Tracking Protection (ETP) set to Strict, Samsung Internet with Smart Anti-Tracking (SAT) set to Strict, Safari in private browsing mode, and Tor Browser with its default configuration. All experiments use the latest available versions at the time of testing: Firefox 144, Brave 1.88.138, Samsung Internet 29.0.0.59, Safari 26.4, and Tor 15. We also test four browser extensions that specifically target canvas fingerprinting: Canvas Fingerprint Defender on Chrome (~50K users) [11], Canvas Blocker on Chrome (~90K users) [10], Fingerprint Spoofer on Chrome (~20K users) [19], and CanvasBlocker by kkapsner on Firefox (~39K users) [24].

Two-phase measurement. For each browser configuration, we run two complementary experiments. First, we disable LearnedFP's own server-side randomization and repeatedly render fixed probes. This isolates defense-induced changes from LearnedFP's intentional probe variation. We collect outputs across ten independent sessions and restart the browser between sessions so that per-session defense state is refreshed. Second, we re-enable server-seeded randomized probes and evaluate whether LearnedFP can still identify devices in the presence of the defense.

Identification evaluation dataset. We evaluate identification under defenses on a gallery of 653 entries: 593 participant devices from DB_{T_1} and 60 in-lab browser instances spanning Windows, macOS, Android, and iOS (Appendix D). An instance is one browser

configuration on one device. The in-lab instances cover twelve configurations: three standard configurations with no active canvas defense (Chrome, Firefox, Safari) and nine defended configurations, with five instances per configuration (15 default and 45 defended instances in total). We span multiple GPU vendors and architectures to avoid conflating defense artifacts with hardware-specific rendering differences. Each in-lab instance contributes 100 canvas samples; participant devices contribute up to 100 (median 45, mean 52). Samples are split 80/20 chronologically into enrollment and query sets. The `LearnedFP` encoders are trained on DB_{T_0}, which is disjoint from all devices in this evaluation, and remain frozen throughout.

6.2 Defense Characterization

Metrics. We characterize each defense along two axes, corresponding to the columns of Table 4. The first axis captures perturbation behavior. Δpx is the fraction of pixels that change across sessions; Max Δ is the largest per-channel deviation (0–255); and Alpha indicates whether the alpha channel remains unmodified. The second axis captures extraction-path consistency. Stable means repeated identical calls return the same bytes. Sub = Full means a `getImageData()` read of a subregion agrees with the same region of a full-canvas read. Px = PNG means the raw RGBA buffer agrees with the PNG-decoded `toDataURL()` output. URL = Blob means `toDataURL()` agrees with `toBlob()`. We additionally test `OffscreenCanvas` (OSC) behavior in §6.3.

These metrics capture not only whether a defense changes the output, but also whether it changes the output coherently. A small but stable perturbation may be learnable; a large perturbation may still leave some channels untouched; and an inconsistent perturbation may reveal the defense itself.

Table 4 shows that most defenses do not eliminate canvas signal. Instead, they modify it partially or inconsistently. Several defenses leave the alpha channel untouched, preserving a stable signal that a per-pixel RGBA encoder can still exploit. Others perturb different extraction paths differently, creating stable cross-path artifacts. These artifacts are especially problematic because they can reveal the presence of the defense even when they do not identify the underlying device.

6.3 Browser-Specific Findings

Brave. Brave's Farbling defense injects bounded per-pixel noise ($\Delta \leq 1$) on all nine keys, leaving the alpha channel intact. Although the noise is small, it is applied inconsistently across extraction APIs: Sub≠Full, Px≠PNG, and URL≠Blob on eight of nine keys, and repeated `toBlob()` calls can return different bytes for the same canvas. A tracker can identify Brave from these cross-path inconsistencies alone.

Firefox. Firefox ETP Strict applies per-session noise to all nine keys, bounded by $\Delta \leq 3$ on the fully opaque `GradStep` key but reaching $\Delta = 255$ on the others. Unlike Brave, the defense is largely API-coherent within a session: the only discrepancy is Px≠PNG, and this discrepancy also appears in Firefox without ETP Strict, making it partly browser-native rather than defense-specific.

Samsung Internet. Samsung Internet SAT Strict produces the sparsest perturbation we observed: two changed pixels per 10,000-pixel canvas for eight keys (four for `GradStep`), confined to the G and B channels with R and alpha untouched. The perturbation is unstable across repeated calls, and the Px≠PNG discrepancy appears under SAT Strict but not in the default setting. These signatures uniquely identify SAT Strict.

Safari. Safari Private is one of the few configurations that modifies the alpha channel, and only on keys containing semi-transparent pixels. The main canvas pipeline is otherwise coherent across all extraction paths (`getImageData`, `toDataURL`, `toBlob`, subregion and full reads), but OSC outputs diverge from direct `getImageData()` reads on all nine keys, with deviations reaching $\Delta = 255$. Neither artifact appears in non-private Safari, making Safari Private directly detectable.

Tor. Tor uses the most restrictive strategy. Rather than injecting bounded noise, it gates canvas extraction and requires explicit user permission before scripts can access pixel data through any extraction API. If a script attempts silent extraction before permission is granted, Tor returns a synthetic fallback output. The fallback tiles a small set of repeated pixel values with an 8-pixel period and is unstable across calls and extraction paths. This prevents stable device-level readback but can disrupt legitimate canvas functionality, such as image editors, color pickers, and applications that rely on pixel readback. Moreover, the fallback makes Tor itself detectable even when individual Tor users are not distinguishable from one another.

Browser extensions. The four extensions illustrate four distinct perturbation strategies: uniform pixel shift, sparse grid perturbation, draw-time noise, and API hijacking. Canvas Fingerprint Defender applies a near-uniform shift to ∼87% of pixels and modifies alpha. Canvas Blocker perturbs only a sparse 10×10 grid ($(0,0), (10,0), \ldots$), bounded by $\Delta \leq 10$ with alpha intact. The regular spatial pattern is itself a distinctive signature. CanvasBlocker by kkapsner injects draw-time noise reaching $\Delta = 255$ on ∼46% of pixels, yet repeated `getImageData()` calls return byte-identical results. Fingerprint Spoofer replaces `getContext()` and `toDataURL()` with a synthetic canvas, but it fails to override `toBlob()`. In our tests, `getImageData()` returns the same output for all nine keys, rather than plausibly perturbing each one.

Defense detectability. The very mechanisms designed to prevent canvas fingerprinting can themselves become fingerprinting signals. To make this concrete, we construct a minimal defense-detection probe that fills a 100×100 canvas with a semi-transparent color and reads the pixel buffer twice (Figure 7a). On an undefended browser, internal rounding during canvas extraction shifts the input slightly. Chrome and Safari return $e = (100, 149, 199, 128)$, whereas Firefox returns $(99, 149, 199, 128)$. We treat both as undefended baselines, and any further deviation indicates either an active defense or a distinctive browser behavior.

Figure 7 shows that a small number of defense-specific artifacts is sufficient to uniquely identify all nine tested defenses. Four produce a signature that no other defense shares. Where signatures overlap, one additional signal separates them: Canvas Fingerprint Defender and CanvasBlocker by kkapsner both produce a uniform shift, distinguished by whether it is stable across reads; Brave and Firefox ETP Strict both perturb RGB, distinguished by magnitude;

Table 4: Defense characterization and LearnedFP identification accuracy under defense ($N = 653$ gallery; random Top-1 baseline = 0.15%; metric definitions in §6.2; lift = accuracy / $(1/N)$).

Browser / Extension	Setting	Perturbation			Extraction consistency				Identification accuracy		
		Δpx	Max Δ	Alpha	Stable	Sub=Full	Px=PNG	URL=Blob	Top-1 (lift)	Top-3	Top-5
Browser defaults (no active canvas defense)											
Chrome	Standard	0%	0	✓	✓	✓	✓	✓	0% (0×)	100%	100%
Firefox	Standard	0%	0	✓	✓	✓	✗	✓	80% (522×)	100%	100%
Safari	Standard	0%	0	✓	✓	✓	✓	✓	60% (392×)	80%	80%
Browser-level defenses											
Brave	Default	≈ 5%	1	✓	✗	✗	✗	✗	60% (392×)	100%	100%
Firefox	ETP Strict	≈ 2%	255	✓	✓	✓	✗	✓	60% (392×)	100%	100%
Samsung Internet	SAT Strict	0.02–0.04%	3	✓	✗	✗	✗	✗	60% (392×)	80%	100%
Safari	Private	≈ 25%	255	✗	✓	✓	✓	✓	100% (653×)	100%	100%
Tor	Default	100%	255	✗	✗	✗	✗	✗	20% (131×)	40%	100%
Browser extensions											
Canvas FP Defender	Chrome	≈ 87%	255	✗	✗	✗	✗	✗	60% (392×)	100%	100%
Canvas Blocker	Chrome	≈ 0.5%	10	✓	✗	✗	✗	✗	100% (653×)	100%	100%
CanvasBlocker by kkapsner	Firefox	≈ 46%	255	✓	✓	✓	✗	✓	40% (261×)	100%	100%
Fingerprint Spoofer	Chrome	≈ 0%	255	✗	✗	✗	✗	✗	60% (392×)	100%	100%

```
ctx.fillStyle = 'rgba(100, 150, 200, 0.5)';
ctx.fillRect(0,0,100,100);
const px = () => ctx.getImageData(0,0,100,100).data;
const read1 = px(), read2 = px();  // repeated calls
```

(a) Defense detection probe.

Defense	Condition
Tor	period-8 tile
Fingerprint Spoofer	`ctx.canvas` ≠ canvas element
Canvas FP Defender	uniform shift; reads unstable
CanvasBlocker (kkapsner)	uniform shift, or broad OSC ≠ direct read; reads stable, α intact
Safari (Private)	α modified, or OSC ≠ direct read with `read1[0]` = 100
Canvas Blocker	deviations only on 10×10 grid; α intact
Samsung SAT Strict	G/B perturbed; R clean
Brave (Farbling)	per-pixel noise, max $\Delta \leq 2$
Firefox ETP Strict	max $\Delta > 2$, or OSC ≠ direct read with `read1[0]` = 99
No defense	uniform fill = baseline

(b) Classifier rules. Δ is the per-channel deviation from the undefended baseline. The OSC comparison uses a separate semi-transparent canvas with anti-aliased edges. The first matching rule wins.

Figure 7: Defense detection probe (a) and classifier rules (b).

and Safari Private and Firefox ETP Strict both diverge between OSC and direct reads, distinguished by the value read back from the uniform fill. No permission prompt, timing side channel, or user interaction is required; the probe is ordinary canvas code and completes in well under a second.

6.4 Identification Results

Effectiveness. As shown in Table 4, anti-fingerprinting defenses do not prevent LearnedFP from re-identifying defended instances. Among the 45 active-defense instances, LearnedFP achieves 62.2% Top-1, 91.1% Top-3, and 100% Top-5 accuracy. All nine active-defense groups reach 100% Top-5; seven reach 100% Top-3 (Samsung Internet at 80% and Tor at 40% are the exceptions). Since the random Top-1 baseline in the 653-device gallery is only 0.15%, even the weakest Top-1 result is 131× chance.

The high absolute accuracies on the in-lab instances reflect the lab evaluation setup. Three factors matter. First, each configuration contains only five instances. Second, lab instances span multiple operating systems and GPU vendors, providing more cross-instance diversity than a homogeneous user pool. This is visible in the standard rows: Firefox and Safari span multiple platforms and reach 80% and 60% Top-1, while Chrome Standard's Windows instances collide. Third, each lab instance contributed the full 100 samples, compared with a median of 45 across participant devices, yielding more stable enrollment profiles. Because all three factors apply equally to standard and active-defense configurations, the comparison between them remains valid.

Defense-induced identifiability. Counterintuitively, defended devices are often easier to identify than their undefended counterparts. The active-defense instances achieve higher Top-1 accuracy than both the 15 standard in-lab instances (46.7%) and the 593 participant devices (20.7%). Safari illustrates this effect: Safari Standard achieves 60% Top-1, while Safari Private reaches 100% Top-1 after introducing approximately 25% pixel perturbation. Canvas Blocker on Chrome reaches 100% Top-1, while Chrome Standard achieves 0%. However, Chrome Standard reaches 100% Top-3 for all five

Table 5: Web Audio identification with LearnedFP. V1 and V2 denote the initial and follow-up campaigns. Random = $1/N$, where N is the gallery size; ± is the standard deviation across cross-validation folds.

Setting	Enroll/Query	Random (%)	Top-1 (%)
Same visit (seen)	V1 80/20 split	0.48	11.85 ± 5.40
Same visit (unseen)	V1 30/20 samples	1.92	11.82 ± 2.78
Return visit	V1→V2 30/20 samples	0.38	8.27

instances, indicating that the Top-1 failures reflect ties with a small number of very similar Windows-Chrome entries. Firefox is an exception: ETP Strict reduces Top-1 from 80% (Firefox Standard) to 60%, and CanvasBlocker by kkapsner from 80% to 40%, although Top-3 remains 100% in both cases.

Defense-induced identifiability arises because current defenses introduce structured perturbation artifacts. Rather than enlarging the user's anonymity set, a defense can add a distinctive signal tied to the browser-defense combination on top of natural hardware and rendering variation. Tor is the strongest evaluated defense against device-level identification, reaching only 20% Top-1. However, all five Tor instances are recovered within Top-5, and the nearest neighbors of Tor queries are other Tor instances. This indicates cohort detection rather than device identification: a tracker can detect that the user runs Tor even if it cannot separate individual Tor users.

These results expose a fundamental tension in functionality-preserving defenses: bounded perturbations leave learnable pixel-level structure, untouched channels (e.g., the alpha channel) remain informative, stable perturbations become fingerprinting signals, and cross-path inconsistencies identify the defense itself. The learned encoder does not require outputs to be identical; it only requires the residual response distribution to remain distinguishable. Output instability alone is therefore not a sufficient evaluation criterion.

Insight 3: Anti-fingerprinting defenses can become fingerprinting signals.

7 Generalization Beyond Canvas

We use HTML5 `<canvas>` as the primary case study, but the risk extends beyond pixels and graphical rendering: any browser surface that supports controlled challenges and repeated high-dimensional readback may expose learnable signal.

7.1 Case Study: Web Audio

To test whether the same pattern appears beyond canvas, we instantiate the `LearnedFP` workflow on Web Audio. Unlike canvas, Web Audio produces a numeric waveform rather than an RGBA pixel buffer, exercising an independent audio processing pipeline. Our payload executes 12 Web Audio graphs in an `OfflineAudioContext`, each producing a `Float32Array` waveform; the graphs cover oscillators, filter chains, compression, waveshaping, reverb, panners, delay lines, and channel mixing. As in the canvas pipeline, graph parameters are server-seeded.

Using the same Prolific recruitment and compensation protocol as the canvas study, we separately recruited 262 participants; 29 PIDs also appear in the canvas cohort. We collected Web Audio measurements in an initial campaign and re-invited participants to the same site five days later for a follow-up campaign. A first-party cookie linked the two visits for 133 returning devices. Each device contributed 60 samples per campaign. We train per-probe encoders on initial-campaign data using 5-fold cross-validation by device identity and keep them frozen throughout. We evaluate two settings. First, we test whether independently sampled measurements from the same visit cluster together in the learned representation, for both seen identities (used to train the encoders) and unseen identities (held out from training). Second, we test whether a return visit can be linked back to its initial-campaign profile.

Results. Table 5 shows that Web Audio also exposes learnable device-specific structure. For the same-visit evaluation, seen identities use an 80/20 per-device split, while unseen identities use 30 enrollment samples and 20 query samples. For the return visit, we use 30 samples from the initial campaign for enrollment and 20 from the follow-up campaign for query. The gallery contains all 262 devices, including both seen and unseen identities. Same-visit identification reaches ∼11.8% Top-1 in both settings, 24.6× and 6.1× their respective random baselines. In the return-visit setting, Top-1 reaches 8.27% (21.7× random) and Top-5 reaches 15.79% (8.3× random). Even across visits and randomized audio probes, the learned representation retains cross-session linkage signal.

We report descriptive results for Brave and Chrome returnees. Brave returnees (N=11), whose Farbling state resets each session, reach 45.45% Top-1 (119× random) and 72.73% Top-5. Chrome returnees (N=100), with no active audio defense, reach only 2.00% Top-1 (5.2× random) and 7.00% Top-5. Although the Brave cohort is small, Brave returnees are *more* identifiable than Chrome returnees despite the defense, echoing the defense-induced identifiability observed for canvas.

7.2 Other Surfaces

Other browser APIs expose comparably rich, repeatedly sampleable responses, such as WebGL and WebGPU through GPU readback, or media and layout APIs through platform-specific processing, though we do not evaluate them here. A deployed tracker also need not depend on any single strong surface. The architecture used throughout this paper extends directly to multiple surfaces and can further fuse evidence across modalities. Surface-specific defenses are therefore incomplete: even if one API is sanitized, other surfaces may remain exposed, and durable defenses must be evaluated at the level of extraction capability rather than the output stability of any single surface.

Insight 4: The threat is the capability, not the canvas. Any API exposing high-fidelity readback is a candidate.

8 Discussion

8.1 Ethical Considerations

Our study was reviewed by our Institutional Review Board and determined to be Not Human Subjects Research. Before data collection, participants were informed of the study purpose and the

categories of data collected; participation was voluntary, and participants could withdraw or request deletion at any time. Measurement records were associated with random identifiers generated by our collection site. We retained pseudonymous participant identifiers provided by the crowdsourcing platform only to construct cross-campaign same-device labels; these identifiers were not used as model inputs. Because browser fingerprinting is dual-use, we do not release trained models, raw participant-level browser fingerprints, or a deployable tracking service. Data handling followed the protocol submitted to our IRB and the commitments made to participants.

8.2 Learning-Resistant Defenses

Our results suggest that defenses should be evaluated by the capabilities they leave to web content, not by output instability alone. Evaluation should measure learnability under realistic sampling budgets, and assess extraction-path consistency and defense-specific artifacts. These observations motivate three design principles. First, extraction capability control: defenses should constrain what scripts can extract, not merely perturb what individual API calls return. A defense can harden the readback layer by intercepting high-fidelity extraction interfaces such as `getImageData()`, `toDataURL()`, and `toBlob()` for canvas. The same approach applies to `readPixels()` for WebGL and `getChannelData()` for Web Audio, so that what the user sees can be separated from what scripts can read. Second, canonical snapshots across extraction paths: piecemeal API patching creates defense-specific signals when one path is perturbed, another sanitized, and a third left unchanged. A defense should route all script-visible readback and export paths through a single canonical snapshot, so partial and full readback, raw buffers and encoded exports, and offscreen round trips remain consistent up to format conversion; for canvas, enforcing RGB = 0 whenever $\alpha = 0$ removes script-readable information that is invisible in the rendered image. Third, bounded sampling budgets: because learning-based tracking depends on many rich and independent samples, a defense should budget extraction per frame and progressively downgrade readback as the budget is exhausted, with stricter defaults for third-party, offscreen, background, and private-browsing contexts.

Prototype. We implement these principles as a canvas defense extension; the same design could extend to other readback surfaces such as WebGL and Web Audio. Every covered readback path reads from a single canonical snapshot, and each frame has its own budget for high-fidelity reads. Nested frames are counted separately, and a frame's counter resets only when it navigates. The extension differs from existing defenses in both directions. Mainstream noise-injection defenses (e.g., Brave's Farbling) introduce variation across reads while leaving underlying device-specific structure learnable across repeated samples. Our extension instead limits readback fidelity as the budget is consumed: the first few reads return the canonical snapshot at its native eight bits per channel, further reads are quantized to four bits per color channel, and reads beyond the budget return a constant image, contributing zero additional bits of canvas information at the cost of being detectable as a defense. Tor blocks high-fidelity readback altogether, which is strong but breaks some legitimate canvas-dependent applications; our extension intercepts only the script-visible readback, so the canvas itself keeps painting natively, and a display-restore path returns the original pixels when a page reads them and writes them back unchanged. A page that instead builds its display from readback data, for example through `toDataURL()`, sees the sanitized bytes. Users can adjust both thresholds per tab based on their needs, allowing the defense to range from permissive to constant-output sanitization; the defaults are 3 and 20 reads, respectively.

8.3 Responsible Disclosure

We disclosed our findings to the affected browser vendors. Mozilla (Firefox), Brave, and Samsung have responded; Safari is continuing to review our report.

Firefox. Mozilla described our prototype as a well-executed example of one of several known canvas-randomization bypasses that it would expect to work against most browsers, and said that canvas randomization is not a defensible security boundary. It also said that, at the time of its response, it was not aware of real-world fingerprinters deploying algorithms to bypass canvas randomization, or detecting Firefox's randomization and acting on it. The bug remains open. After our report on Firefox 144, we re-measured Firefox 149 and observed a substantive change in the defense under ETP Strict. The per-session pixel noise previously applied to the canvas is no longer present; the defense now appears to act at the extraction interfaces (e.g., `toDataURL`). However, we observe a new signal: an `OffscreenCanvas` round trip using `drawImage()` diverges from direct `getImageData()` within a single page load. This binary signal distinguishes ETP Strict from undefended Firefox.

Brave. Brave stated that its anti-fingerprinting protections intentionally balance fingerprinting resistance against site compatibility. It acknowledged that clustering approaches can partially circumvent Farbling, adding that practical attacks require very low false-positive rates and must distinguish users among much larger populations. Brave also explained that the Farbling seed from which the perturbation is derived rotates whenever storage is cleared, so its lifetime is no longer than that of persistent identifiers available to the site. Our defense classifier uses the perturbation's bounded magnitude rather than its specific values, and therefore does not require the seed to persist. Brave noted ongoing efforts to improve its protections while preserving benign page functionality.

Samsung. Samsung did not classify the detectability of its canvas randomization defense as a vulnerability, noting that randomization can still provide privacy benefits even when the browser itself is identifiable. Samsung confirmed that this limitation is common across browsers that implement canvas anti-fingerprinting, and said it would consider our suggestions for future improvements.

Our results expose a gap in how anti-fingerprinting defenses are evaluated. Current defenses largely target deterministic matching, while multi-sample, learning-based attacks require a different threat model. Closing this gap requires both updated evaluation practices and capability-focused defense design. Our extension prototype provides a deployable interim option. An extension is itself detectable, placing its users in a distinct cohort. A default platform-level defense would avoid creating this extension-specific cohort and is therefore the natural next step.

8.4 Limitations

Our cross-campaign labels combine PIDs with a consistency check on browser and device metadata, designed to favor label precision over recall. The check excludes returns with substantial system changes, such as switching from Windows to Android. Two sources of label noise remain possible, and they bias absolute rates in opposite directions: excluding heavily drifted same-device returns can raise them, whereas retaining a replacement device whose metadata matches the original can lower them. DRAWNAPART, `LearnedFP`, and FingerprintJS are evaluated on the same labels, which keeps the comparison internally consistent. A stricter labeling test selects an easier subset and raises the rates for DRAWNAPART and `LearnedFP`, while leaving the comparative conclusion unchanged.

Our lab evaluation includes only five instances per defense configuration. It is designed to test whether defended browsers can be distinguished from their undefended counterparts, not to estimate the population-wide prevalence of these artifacts. Our Web Audio study is a cross-surface case study at smaller scale and shorter span than the canvas evaluation. Its purpose is to demonstrate that the learning-based approach generalizes beyond canvas, not to provide a full evaluation of `LearnedFP` for audio fingerprinting.

Once the readback budget is exhausted, our defense prototype returns a constant image, eliminating device-specific canvas information from subsequent reads. The constant output does not identify individual devices, but can reveal the extension's presence.

9 Related Work

Browser fingerprinting measurement. Browser fingerprinting was popularized by Eckersley's study on the uniqueness of browser and device configurations [15]. Subsequent work measured the deployment of stateless tracking in the wild and characterized the broader fingerprinting ecosystem, including cookieless tracking systems [43], third-party trackers [1], and large-scale tracking measurements [16]. Other studies examined how uniqueness varies across realistic user populations and over time [21, 29]. A complementary line of work detects fingerprinting scripts and behaviors using code analysis [2] and machine learning [22, 48]. Our focus is different from detection and prevalence measurement. We study what a modern tracker can infer when it can issue repeated, controlled browser probes.

Canvas and GPU fingerprinting. Canvas fingerprinting was introduced by Mowery and Shacham [39]. Follow-on work demonstrated that modern browsers expose many rendering behaviors that can be combined into distinctive fingerprints [12, 30], and rendering-based features have also been studied in authentication contexts [14, 28, 36, 47]. Beyond 2D canvas, DRAWNAPART [25] fingerprints devices through GPU timing variation, and recent work on WebGPU and GPU interfaces has shown that new graphics capabilities continue to create fingerprinting and side-channel opportunities [17, 27]. Rather than treating rendering behavior as a fixed identifier or passive observable, we actively control its variability.

Fingerprint instability. A central limitation of deterministic fingerprinting is that browser attributes change over time. Prior longitudinal studies have quantified this instability across user populations, showing that fingerprints can drift as browsers, devices, configurations, and execution environments change [5, 6, 33]. To address this drift, prior work has proposed linkage methods that relax exact matching, including rule-based heuristics and learning-based strategies for reconnecting fingerprints observed at different times [46, 61]. Unlike these approaches, which operate on existing conventional attributes and try to repair instability after it occurs, our work introduces a new fingerprint construction that deliberately incorporates variability into the measurement process.

Anti-fingerprinting defenses and their limitations. To mitigate fingerprinting, browsers increasingly deploy defenses [41, 58]. In parallel, researchers have proposed techniques for detecting and blocking fingerprinting scripts [7, 8, 42, 59]. Despite these efforts, fingerprinting remains prevalent. Modern fingerprinting techniques blur the boundary between benign web functionality and tracking, while newly introduced browser APIs and execution paths are repeatedly repurposed for privacy-invasive measurement [23, 51, 52]. These trends expose the limitations of defenses that rely on static assumptions about how fingerprinting is performed. Our work re-evaluates these defenses against a learning-based adversary.

Fingerprinting across surfaces and applications. Fingerprinting has expanded to surfaces beyond JavaScript canvas APIs. CSS-based and stylistic fingerprinting can bypass defenses focused on JavaScript outputs [31, 56, 60], browser extensions expose new fingerprintable resources and behaviors [50, 54, 55], font and sensor APIs leak device-specific signal [13, 18], and peripheral timing introduces additional fingerprintable variation [38]. Fingerprints are also used as risk signals [3, 4, 53, 62], but deterministic fingerprints can be replayed or spoofed, undermining authentication workflows [35, 37, 49]. These directions explore where fingerprinting occurs. We study how learning-based inference reshapes fingerprinting and its defenses across surfaces.

10 Conclusion

This paper argues that browser fingerprinting should be understood not only as deterministic matching over stable attributes, but as multi-sample inference over learnable response distributions. The defensive question therefore shifts from output stability to learnability under realistic sampling budgets. We demonstrate this through `LearnedFP`, a learning-based fingerprinting system built on HTML5 `<canvas>`. On a crowdsourced longitudinal dataset of 3,896 devices spanning four months, `LearnedFP` substantially outperforms the state-of-the-art non-deterministic baseline by inverting the role of variability: instead of treating variability as drift, `LearnedFP` turns it into a learning signal and remains effective despite the long-standing instability of conventional fingerprinting. Our defense evaluation reveals that most of the mechanisms we test leave residual signal learnable by the model and can introduce defense-specific artifacts that make defended devices more identifiable. The risk is not confined to canvas. Our Web Audio case study suggests that any browser interface that allows controlled challenges and repeated high-dimensional responses can become a learnable measurement surface. These findings call for a shift in how defenses are designed and evaluated. We develop a browser extension as an interim defense before native browser integration. We hope this work motivates a new generation of defenses that account for what trackers can learn, not only what they observe.

References

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 674–689.

[2] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 1129–1140.

[3] Furkan Alaca and Paul C Van Oorschot. 2016. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *Proceedings of the 32nd annual conference on computer security applications*. 289–301.

[4] Nampoina Andriamilanto and Tristan Allard. 2021. Brfast: a tool to select browser fingerprinting attributes for web authentication according to a usability-security trade-off. In *Companion Proceedings of the Web Conference 2021*. 701–704.

[5] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. 2021. "Guess Who?" Large-scale data-centric study of the adequacy of browser fingerprints for web authentication. In *Innovative Mobile and Internet Services in Ubiquitous Computing: Proceedings of the 14th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2020)*. Springer, 161–172.

[6] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. 2020. FPSelect: low-cost browser fingerprints for mitigating dictionary attacks against web authentication mechanisms. In *Proceedings of the 36th Annual Computer Security Applications Conference*. 627–642.

[7] Meenatchi Sundaram Muthu Selva Annamalai, Igor Bilogrevic, and Emiliano De Cristofaro. 2023. Fp-fed: privacy-preserving federated detection of browser fingerprinting. *arXiv preprint arXiv:2311.16940* (2023).

[8] Sarah Bird, Vikas Mishra, Steven Englehardt, Rob Willoughby, David Zeber, Walter Rudametkin, and Martin Lopatka. 2020. Actions speak louder than words: Semi-supervised learning for browser fingerprinting detection. *arXiv preprint arXiv:2003.04463* (2020).

[9] Brave Software. 2020. Brave Fingerprint Randomization. https://brave.com/privacy-updates/3-fingerprint-randomization/. Accessed: Feb. 8, 2026.

[10] Canvas Blocker 2026. Canvas Blocker - Fingerprint Protect. https://chromewebstore.google.com/detail/canvas-blocker-fingerprin/nomnklagbgmgghhjidfhnoelnjfndfpd. Chrome Web Store, accessed March 31, 2026.

[11] Canvas Fingerprint Defender 2026. Canvas Fingerprint Defender. https://chromewebstore.google.com/detail/canvas-fingerprint-defend/lanfdkkpgfjfdikkncbnojekcppdebfp. Chrome Web Store, accessed March 31, 2026.

[12] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-) browser fingerprinting via OS and hardware level features. In *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society.

[13] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The web's sixth sense: A study of scripts accessing smartphone sensors. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1515–1532.

[14] Nor Izyani Daud, Galoh Rashidah Haron, and Siti Suriyati Syd Othman. 2017. Adaptive authentication: Implementing random canvas fingerprinting as user attributes factor. In *2017 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. IEEE, 152–156.

[15] Peter Eckersley. 2010. How unique is your web browser?. In *Privacy Enhancing Technologies: 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings 10*. Springer, 1–18.

[16] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 1388–1401.

[17] Ethan Ferguson, Adam Wilson, and Hoda Naghibijouybari. 2024. Webgpu-spy: Finding fingerprints in the sandbox through gpu cache attacks. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. 158–171.

[18] David Fifield and Serge Egelman. 2015. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19*. Springer, 107–124.

[19] Fingerprint Spoofer 2026. Fingerprint Spoofer. https://chromewebstore.google.com/detail/fingerprint-spoofer/facgnnelgcipeopfbjcajpaibhhdjgcp. Chrome Web Store, accessed March 31, 2026.

[20] FingerprintJS. [n. d.]. FingerprintJS. https://github.com/fingerprintjs/fingerprintjs. Accessed: 2025-07-23.

[21] Alejandro Gomez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *Proceedings of the 2018 world wide web conference*.

[22] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2021. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1143–1161.

[23] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *Proceedings of the 27th Network and Distributed System Security Symposium (NDSS)*.

[24] kkapsner. 2026. CanvasBlocker. https://addons.mozilla.org/en-US/firefox/addon/canvasblocker/. Firefox Add-ons (Mozilla), accessed March 31, 2026.

[25] Tomer Laor, Naif Mehanna, Antonin Durey, Vitaly Dyadyuk, Pierre Laperdrix, Clémentine Maurice, Yossi Oren, Romain Rouvoy, Walter Rudametkin, and Yuval Yarom. 2022. DRAWNAPART: A Device Identification Technique based on Remote GPU Fingerprinting. In *Network and Distributed System Security Symposium*.

[26] Tomer Laor, Naif Mehanna, Antonin Durey, Vitaly Dyadyuk, Pierre Laperdrix, Clémentine Maurice, Yossi Oren, Romain Rouvoy, Walter Rudametkin, and Yuval Yarom. 2022. DrawnApart: A Device Identification Technique Based on Remote GPU Fingerprinting — Artifact and Code Repository. https://github.com/drawnapart/drawnapart. Accessed: 2025-10-24.

[27] Tomer Laor and Yossi Oren. 2025. Poster: LockedApart: Faster GPU Fingerprinting Through the Compute API. In *Proceedings of the Microarchitecture Security Conference*.

[28] Pierre Laperdrix, Gildas Avoine, Benoit Baudry, and Nick Nikiforakis. 2019. Morellian analysis for browsers: Making web authentication stronger with canvas fingerprinting. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16*. Springer, 43–66.

[29] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)* 14, 2 (2020), 1–33.

[30] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE.

[31] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. 2021. Fingerprinting in style: Detecting browser extensions via injected style sheets. In *30th USENIX Security Symposium (USENIX Security 21)*. 2507–2524.

[32] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. 2016. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security 16)*.

[33] Song Li and Yinzhi Cao. 2020. Who touched my browser fingerprint? a large-scale measurement study and classification of fingerprint dynamics. In *Proceedings of the ACM Internet Measurement Conference*. 370–385.

[34] Xu Lin, Frederico Araujo, Teryl Taylor, Jiyong Jang, and Jason Polakis. 2023. Fashion Faux Pas: Implicit Stylistic Fingerprints for Bypassing Browsers' Anti-Fingerprinting Defenses. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 987–1004.

[35] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. 2022. Phish in sheep's clothing: Exploring the authentication pitfalls of browser fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*. 1651–1668.

[36] Zengrui Liu and Nitesh Saxena. 2024. Dual Study of Canvas Fingerprinting Based Authentication: A Novel Spoofing Attack and the Countermeasure. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1354–1364.

[37] Zengrui Liu, Prakash Shrestha, and Nitesh Saxena. 2022. Gummy browsers: targeted browser spoofing against state-of-the-art fingerprinting techniques. In *International Conference on Applied Cryptography and Network Security*. Springer, 147–169.

[38] John V Monaco. 2022. Device fingerprinting with peripheral timestamps. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1018–1033.

[39] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP* (2012).

[40] Mozilla. [n. d.]. Fingerprinting Protections in Firefox. https://blog.mozilla.org/en/firefox/fingerprinting-protections/. Mozilla Blog, accessed January 2026.

[41] Mozilla. 2025. Firefox's Protection Against Fingerprinting. https://support.mozilla.org/en-US/kb/firefox-protection-against-fingerprinting. Accessed: Feb. 8, 2026.

[42] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. 2015. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*. 820–830.

[43] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2013. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy*. IEEE.

[44] Privacy Working Group. 2025. Mitigating Browser Fingerprinting in Web Specifications. https://www.w3.org/TR/fingerprinting-guidance/. W3C Working Group Note.

[45] Prolific. 2025. Prolific: Online participant recruitment and crowdsourcing platform. https://www.prolific.com.

[46] Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. 2020. Long-term observation on browser fingerprinting: Users' trackability and perspective. *Proceedings on Privacy Enhancing Technologies* (2020).
[47] Esteban Rivera, Lizzy Tengana, Jesús Solano, Christian López, Johana Flórez, and Martín Ochoa. 2022. Scalable and secure html5 canvas-based user authentication. In *International Conference on Applied Cryptography and Network Security*. Springer, 554–574.
[48] Valentino Rizzo, Stefano Traverso, and Marco Mellia. 2021. Unveiling web fingerprinting in the wild via code mining and machine learning. *Proceedings on Privacy Enhancing Technologies* (2021).
[49] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. 2024. The double edged sword: identifying authentication pages and their fingerprinting behavior. In *Proceedings of the ACM Web Conference 2024*. 1690–1701.
[50] Konstantinos Solomos, Panagiotis Ilia, Soroush Karami, Nick Nikiforakis, and Jason Polakis. 2022. The dangers of human touch: fingerprinting browser extensions through user actions. In *31st USENIX Security Symposium (USENIX Security 22)*. 717–733.
[51] Konstantinos Solomos, Panagiotis Ilia, Nick Nikiforakis, and Jason Polakis. 2022. Escaping the Confines of Time: Continuous Browser Extension Fingerprinting Through Ephemeral Modifications. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2675–2688.
[52] Konstantinos Solomos, Nick Nikiforakis, and Jason Polakis. 2024. Harnessing Multiplicity: Granular Browser Extension Fingerprinting through User Configurations. In *2024 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 161–174.
[53] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2015. Mobile device fingerprinting considered harmful for risk-based authentication. In *Proceedings of the eighth European workshop on system security*. 1–6.
[54] Oleksii Starov, Pierre Laperdrix, Alexandros Kapravelos, and Nick Nikiforakis. 2019. Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In *The World Wide Web Conference*. 3244–3250.
[55] Oleksii Starov and Nick Nikiforakis. 2017. Xhound: Quantifying the fingerprintability of browser extensions. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE.
[56] Naoki Takei, Takamichi Saito, Ko Takasu, and Tomotaka Yamada. 2015. Web browser fingerprinting using only cascading style sheets. In *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*. IEEE, 57–63.
[57] The Chromium Projects. 2026. User-Agent Reduction. https://www.chromium.org/updates/ua-reduction/. Accessed: Feb. 8, 2026.
[58] The Tor Project. 2019. Browser Fingerprinting: An Introduction and the Challenges Ahead. https://blog.torproject.org/browser-fingerprinting-introduction-and-challenges-ahead/. Accessed: Feb. 8, 2026.
[59] Christof Ferreira Torres, Hugo Jonker, and Sjouke Mauw. 2015. FP-Block: usable web privacy by controlling browser fingerprinting. In *Computer Security–ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II 20*. Springer, 3–19.
[60] Leon Trampert, Daniel Weber, Lukas Gerlach, Christian Rossow, and Michael Schwarz. 2025. Cascading Spy Sheets: Exploiting the Complexity of Modern CSS for Email and Browser Fingerprinting. In *Network and Distributed System Security Symposium (NDSS)*.
[61] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. Fp-stalker: Tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE.
[62] Shujiang Wu, Pengfei Sun, Yao Zhao, and Yinzhi Cao. 2023. Him of Many Faces: Characterizing Billion-scale Adversarial and Benign Browser Fingerprints on Commercial Websites. In *NDSS*.

A Open Science

We release the `LearnedFP` collection framework, training and evaluation pipeline, the code used to run and evaluate the DRAWNAPART and FingerprintJS baselines, the metadata-consistency check, defense-detection probe, lab `LearnedFP` samples, Web Audio experiment, and mitigation prototype at https://github.com/xlin29/learned-fp. We do not release participant-level crowdsourced fingerprints or derived representations such as features or embeddings. We also do not release models trained on those data, as doing so would lower the barrier to deploying a tracker. The released code and lab data nevertheless support end-to-end verification of the identification pipeline, defense detection, and mitigation prototype.

B Generative AI Usage

In accordance with the ACM Policy on Authorship, we disclose our use of Claude in preparing this paper. The tool was used for editorial assistance with writing, grammar, and spelling, and for coding assistance with our research code. All research ideas, the threat model, the `LearnedFP` system design, the defense characterization, the design of the mitigation prototype, and the experimental design are our own; the tool was not used to generate citations, dataset statistics, or any reported results. We are responsible for all content: AI-assisted code was manually reviewed, tested, and verified to produce the expected behavior; all reported numbers were generated by running this verified code on data we collected; and AI-assisted text was reviewed and revised before inclusion.

C `LearnedFP` Implementation

Every per-key encoder contains blocks of two `Conv2d–BatchNorm–ReLU` layers, doubling channel width at each block, followed by global average pooling; a linear layer and ReLU produce the embedding, which a dropout layer and linear classifier map to device logits. All encoders train for 30 epochs with AdamW, cosine annealing, and mixed precision, with the architecture and optimizer settings tuned per key by Bayesian optimization (79 trials of five epochs each). The full-scale evaluation of §5.4 instead uses one configuration for all nine keys: three blocks, base width 32, 3×3 kernels, dropout 0.2, learning rate 10^{-3}, weight decay 10^{-4}, batch size 256, label smoothing 0.05, and EMA decay 0.999. It reaches 15.90% Top-1 there against the searched configuration's 15.45%, and 12.18% against 12.55% on the temporal evaluation. For Figure 6, per-sample logits within DB_{T_0} are averaged in blocks of five; the ensemble additionally sums them across the keys. Cosine gallery ranking would give the ensemble 16.34% rather than 15.90%.

D Lab Devices

Table 6 lists the 13 physical devices underlying the in-lab instances.

Table 6: Lab devices used for the defense evaluation.

Platform	Devices
macOS 26.4	MacBook Pro (M4), MacBook Pro (M2 Max), MacBook Air (M1)
Windows 11	Lenovo LOQ 15IRX10 (i7-13650HX, NVIDIA dGPU), HP OMEN 30L (i7-10700K, RTX 3090), HP 14-dk1xxx (AMD Radeon iGPU)
Android	Galaxy A15, Galaxy Tab A11+, Galaxy A50 (Exynos 9610), Galaxy S21 FE 5G, Galaxy A56 5G
iOS	iPhone 17 (A19), iPhone Air (A19 Pro)