On the Infeasibility of Population-Scale Browser Fingerprint Matching for Discord Ban-Evasion Detection: A Post-Mortem

Cian Kelly
Atria, a fiscally sponsored project of The Hack Foundation, a 501(c)(3) non-profit
cian@atriasafety.org

August 2026

Abstract

We report on an approximately two-week engineering and measurement effort to build a browser and device fingerprinting system, internally named Ninhydrin, intended to detect Discord ban-evasion ("alting") by matching a rejoining account's device fingerprint against the fingerprints of every other member or every previously banned member of a server. The system as specified requires a one-to-many (1:N) comparison at population scale: a new join is compared against N stored fingerprints, and the visitor is flagged if any single comparison clears a similarity threshold. We collected four waves of real-device ground-truth data using AWS Device Farm (up to 153 distinct physical devices across iOS and Android), built and calibrated a weighted per-field similarity scorer, integrated a third-party fingerprinting library (FingerprintJS) as an additional signal, and iteratively tuned scoring weights and decision thresholds against measured true-positive and false-positive rates. Individually, several of these interventions produced real, measured, reproducible improvements at the level of a single pairwise comparison. However, we show that the same measured per-pair error rates, when applied across the number of comparisons a 1:N deployment requires at realistic server sizes (thousands of members), compound multiplicatively and drive the probability that at least one innocent comparison exceeds the decision threshold toward unity, particularly for popular device models. We further show that every remediation we attempted, raising the decision threshold, gating on an auxiliary exact-match signal, requiring network-address corroboration, and adopting a population-size-adjusted (Bonferroni-style) threshold, either fails outright or only narrows the scope of the failure without resolving it, and that our own statistical power to certify any of these interventions safe is fundamentally limited by the small number of real same-model device pairs (order of tens) available from real-device testing campaigns of the size we could afford. We conclude that population-scale 1:N fingerprint matching against a full server membership is not achievable with the signal set, data volume, and engineering resources available to this project, and we document this conclusion, together with the underlying measurements, so that the reasoning is not lost and does not need to be independently rediscovered. A postscript, added several days after the conclusion below was reached, documents and independently measures a follow-on implementation attempt that narrows the comparison gallery from full membership to guild ban history and replaces the fixed global threshold with a per-guild, per-device-class calibrated decision rule. Measured directly against this project's real device corpus, that attempt is a genuine improvement in engineering integrity, it converts an unbounded, silently-catastrophic failure mode into a bounded, honestly-reported one, but it does not recover the underlying entropy shortfall this paper documents: on the real corpus subset with the least available entropy, the calibrated rule's own true-positive rate collapses to single digits alongside its false-positive rate, and at a simulated population of ten thousand candidates the same rule abstains on effectively

all traffic from the hardest real device class. We read this as confirming, not refuting, the central conclusion.

Contents

1 Introduction **4**

2 System Architecture **5**
2.1 Intended product flow 5
2.2 Fingerprint comparison 5
2.3 Decision threshold 6

3 Data Collection **7**

4 What Worked: Measured, Real Improvements **7**
4.1 Integrating a third-party whole-fingerprint hash 8
4.2 Absolute score caps for well-evidenced failure modes 9
4.3 Vendoring over reinventing 9

5 The Core Mechanistic Finding: Correlated Entropy Suppression on iOS Safari **9**

6 What Did Not Work: Rejected Interventions **11**
6.1 Proportional reweighting of iOS per-unit signals 11
6.2 Automated per-field weight optimization 11
6.3 CPU-timing ("silicon lottery") fingerprinting 12
6.4 Clock-oscillator ("Clock Around the Clock") fingerprinting 12
6.5 WebGPU `timestamp-query` execution-unit timing 13
6.6 Signals that actively backfired in production scoring 14
6.7 Two provisional mechanisms, flagged rather than fixed 15
6.8 Manual halving of model-class weight, as a diagnostic, not a fix 16

7 Threshold Sensitivity Analysis **16**

8 The Population-Scale 1:N False-Positive Problem **18**
8.1 The compounding mechanism 18
8.2 Real deployment behavior is not independent, and the general case is better than the naive estimate 18
8.3 Why the general case does not describe a real deployment at scale 19
8.4 Manufacturer and platform concentration 20
8.5 Restating the negative result 21

9 Attempted Remediations and Why Each Failed **21**
9.1 Raising the decision threshold alone 21
9.2 Gating on an auxiliary exact-match signal before scoring 21
9.3 Requiring network-address corroboration 22
9.4 Population-size-adjusted (Bonferroni-style) threshold 23
9.5 Additional Device Farm testing budget 24

10 Statistical Power: Why the Data Volume Available Cannot Resolve This **24**

11 Product and Architectural Constraints **25**

12 Conclusion **26**

13 Postscript (2026-08-11): A Follow-On Implementation Attempt **27**
13.1 What changed 27
13.2 How we measured it 28
13.3 Results: the pairwise scorer is unchanged 29
13.4 Results: real 1:N search behavior on both corpora 29
13.5 Results: population of ten thousand 31
13.6 Interpretation 32
13.7 The broader signal search, for context 34

1 Introduction

Discord communities that experience repeated ban evasion (a banned user creating a new account, sometimes called "alting," and rejoining) have limited tools for detecting that the new account belongs to the same physical device and person as a previously banned one. Discord's own account identifiers reset on account creation, and IP addresses are unreliable given widespread use of mobile networks, VPNs, and shared connections. A device or browser fingerprint, a composite of characteristics exposed by the browser and operating system such as installed fonts, GPU renderer strings, canvas and audio rendering output, screen geometry, and hardware concurrency, has been proposed in the literature and in commercial anti-fraud products as a way to re-identify a device across account boundaries without relying on cookies or IP address alone.

This paper documents the design, implementation, real-device measurement, and eventual abandonment of such a system, built under the project name Atria (product name Ninhydrin), over a period of roughly two weeks in August 2026. The system's intended architecture, as specified in the project's internal planning document, was straightforward: a joining or reverifying Discord user is directed through a consent screen to a fingerprint collection page, the resulting fingerprint is compared against every fingerprint already on file for that guild (Discord's term for a server), and if the best-scoring comparison exceeds a configured similarity threshold, the new account is flagged for moderator review or, in a stricter configuration, automatically banned.

We refer to this intended full-membership comparison as the *general matcher* throughout this paper, to distinguish it from a narrower, already-implemented component called `FastCheck`, which compares a new submission only against accounts banned in the same guild within the previous thirty minutes. `FastCheck` was designed as a fast, high-precision first pass intended to sit in front of the general matcher once built; it was never intended to be the whole system, and its real production candidate-set size (typically zero to five candidates, occasionally more during an active raid) is orders of magnitude smaller than the candidate-set size the general matcher would face against a server with thousands of members. The distinction matters throughout this paper: many of the false-positive-rate figures the project measured and considered acceptable were measured, implicitly or explicitly, in a regime where the number of comparisons per decision (N) was small. The central finding of this paper is that those same figures are not acceptable, and in most realistic configurations are catastrophic, once N grows to the size of a real server's membership, and that no remediation available to the project within its resource constraints closes this gap.

The paper is organized as follows. Section 2 describes the system architecture, the fingerprint collection and comparison mechanism, and the decision layer that turns a comparison into a moderation action. Section 3 describes the real-device data collection campaigns that grounded every measurement in this paper. Section 4 documents the interventions that produced genuine, measured improvements at the single-comparison level. Section 5 presents the core mechanistic finding that explains why those improvements have a hard ceiling on iOS Safari specifically. Section 6 documents interventions that were tried and rejected, with the measurements that led to rejection. Section 7 presents the full threshold sensitivity analysis. Section 8 is the paper's central negative result: the population-scale 1:N false-positive analysis, including the mathematical mechanism by which small per-pair error rates compound into near-certain false flags at real server sizes. Section 9 documents every proposed remediation to the Section 8 problem and the specific, measured or reasoned, cause of each remediation's failure. Section 10 formalizes the statistical power argument that recurs throughout Section 9: the project's real same-model sample size is not merely insufficient today, it is insufficient by roughly two orders of magnitude, and closing that gap was not achievable within the project's remaining resources. Section 11 discusses the product and architectural decisions that further narrowed the space of viable remediations. Section 12 concludes. Section 13

is a postscript, added several days after Section 12 was written, documenting and independently measuring a follow-on implementation attempt against this project's real device corpus, together with a short account of the broader search for new signals that underlies this paper's repeated claim that one was not found.

We want to be explicit about the nature of this paper. It is not a paper proposing a new fingerprinting technique, nor a paper claiming a system works. It is a negative result and a post-mortem, written because the underlying measurements are real, the reasoning that led to the project's cancellation is non-obvious and took considerable effort to derive correctly, and because we believe the specific failure mode documented here, per-pair error rates that look acceptable in isolation but compound catastrophically under 1:N deployment at scale, is likely to recur in any project that attempts to reuse browser fingerprinting for population-scale identity matching rather than narrow, time-windowed, or low-N verification. We would rather document why this does not work than let the same investigation be repeated from scratch.

2 System Architecture

2.1 Intended product flow

The verification flow, as specified, directs a Discord user through a `/verify` command to a single-use, time-limited token, which resolves to a web page presenting a consent and disclosure screen. On acceptance, the user completes a Roblox account link via OAuth2/PKCE (the product's target audience skews toward Roblox-playing communities), and the browser then runs a fingerprint collection script consisting of approximately forty individual collectors, organized into three data-sensitivity tiers plus a persistent-identity tier and an automation-detection tier. The collected payload is submitted to a server endpoint, hashed and stored, and compared against the guild's existing stored submissions.

The decision layer, internally called `internal/verdict`, was specified to fold a raw comparison score into one of three outcomes: a match on a cryptographically strong persistent identifier or genuine per-unit corroboration counts as CONFIRMED; agreement limited to model-class signals (the GPU renderer string, codec support, hardware benchmarks that are identical across every unit of a given phone model) caps at AMBIGUOUS and never auto-actions; and no match at all is CLEAN. CLEAN submissions are auto-approved with no moderator involvement. AMBIGUOUS, CONFIRMED, and any submission separately flagged by a guild's configured network-address or hardened-browser policy are written to a review queue, and a Discord bot posts an embed with a device summary and match breakdown, together with Approve and Deny buttons, to a moderator-configured channel. The product's default configuration takes no automated action beyond routing to this human review queue; automated banning or kicking is an explicit opt-in per guild, and enabling automated banning was specified to force the strictest available sensitivity setting regardless of the guild's own configured sensitivity, on the reasoning that an irreversible automated action requires a materially higher confidence bar than flagging something for a human to look at.

2.2 Fingerprint comparison

The comparison function, `fingerprint.Compare` in the project's Go implementation, is a weighted average over roughly one hundred and ten individual field-level checks. Each check compares one field (for example, GPU renderer string, screen width, a WebGL-computed hash, a specific canvas rendering hash, an audio-context rendering hash, a WASM benchmark timing bucket, or

the presence of a specific font) between two submissions and contributes its assigned weight to a running score if the two values match exactly, or contributes nothing (while still counting toward the maximum possible score) if they do not. The final percentage is the ratio of achieved score to maximum possible score across every field present on both sides; a field absent on either side (because that collector was not yet implemented at the time of an older submission, or because a device did not support it) is excluded from both numerator and denominator rather than penalized.

Checks are additionally organized into thirteen named groups (for example `gpu_modelclass`, `device_hw_modelclass`, `codec_modelclass`, `perunit_hash`, `perunit_wasm_bench`, `perunit_gpu_bench`, `perunit_timing_misc`, and `thirdparty_hash`), which allowed the project to apply per-group weight multipliers rather than only per-field weights, and to run automated weight optimization at the group level (twelve dimensions) as opposed to the full field level (roughly one hundred and ten dimensions); the distinction between these two levels of optimization, and why the coarser one is the right level of freedom for the amount of real data available, is discussed in Section 6.

Two mechanisms outside the weighted average deserve specific mention because they recur throughout this paper.

First, a persistent storage identifier: a 256-bit random token, issued server-side and written to a cookie, to `localStorage`, and to `IndexedDB` on first visit, and read back on subsequent visits. If both sides of a comparison report an identical, server-issued persistent identifier, the comparison short-circuits to a 100 percent match regardless of every other field, on the reasoning that a match on an unguessable, server-minted token is qualitatively different evidence from a match on any device characteristic: there is no legitimate way for two different physical devices to share one, barring an astronomically improbable collision in the token's own random-generation process. The same mechanism is, by the same reasoning, trivially defeated by any user who clears their browser's storage before creating a second account. This tension, a signal that is provably safe against innocent collision but essentially worthless against a motivated adversary, recurs as a structural theme in Section 9.

Second, an asymmetric mismatch penalty and an absolute score cap, both added after real-device measurement showed that the ordinary weighted-average treatment of certain fields was insufficient. These are discussed in Section 4, since both were genuine, measured improvements over the naive weighted average.

2.3 Decision threshold

The comparison score is a real number in $[0, 100]$. A community-configured sensitivity setting, exposed as a binary Low/High choice mapping to two specific pre-measured threshold values (86 and 83 respectively, not a raw numeric input, for reasons discussed in Section 7), determines whether a given comparison counts as a match. The threshold, named `SensitivityThreshold` in the implementation, is shared identically between `FastCheck` and the general matcher; there is deliberately no separate, hidden threshold for either component.

Critically for the analysis in this paper, neither `FastCheck` nor the general matcher as specified performs an early exit or requires any margin between the best-scoring candidate and the second-best. Every candidate in the (in `FastCheck`'s case, time-windowed; in the general matcher's case, full-membership) comparison set is scored, and the visitor is flagged if the single best-scoring comparison clears the threshold. Because there is no margin requirement, "the best of N candidates clears the threshold" and "at least one of N candidates clears the threshold" are mathematically identical conditions: the maximum of a set of values exceeds a threshold if and only if at least one element of the set does. This equivalence is the starting point for the population-scale analysis in

Section 8.

3 Data Collection

Every quantitative claim in this paper is grounded in real physical device measurements, not synthetic or simulated device fingerprints, collected across four campaigns using AWS Device Farm, a service that provides remote, billed-by-the-minute access to physical mobile devices for automated testing.

Waves 1 through 3 (August 8 to 9, 2026) collected fingerprints from a planned roster of approximately 153 physical devices spanning a range of iPhone and Android models, tagged with a synthetic guild identifier (999000111) reserved for this purpose and never a real Discord guild. Not every planned device produced usable data: a macOS Tahoe-hosted iOS test runner silently dropped sub-17.x iOS devices, and a MIUI install restriction blocked one Xiaomi device model entirely, so the realized dataset is somewhat smaller than the planned roster and unevenly distributed across models, with most model clusters represented by two to four distinct physical units.

Wave 4 (August 10, 2026) revisited the same roster of physical devices under a new synthetic guild identifier (999000112), this time with three additional experimental probes enabled (a CPU-timing-based fingerprint, a clock-oscillator-based fingerprint, and a WebGPU execution-timing probe, all discussed in Section 6) and with two third-party fingerprinting libraries, FingerprintJS and ThumbmarkJS, added to the collection payload for direct comparison against the project's own weighted scoring. This wave targeted 112 device ARNs (Amazon Resource Names identifying specific physical devices in the Device Farm inventory), reused directly from an earlier attempt that had been interrupted mid-run by a loss of network connectivity on the host machine (which was, at the time, tethered to a mobile hotspot whose Cloudflare tunnel dropped), and completed successfully on retry: 224 submissions across 105 distinct physical devices, with 327 of 336 scheduled test-suite invocations passing (the remaining nine failures were a mix of Device Farm images with broken or missing Chrome installations and two device-specific iOS anomalies unrelated to the fingerprinting logic itself).

We emphasize two limitations of this dataset that recur throughout the paper's later sections. First, the entire real-device corpus available to this project, summed across all four waves, contains on the order of 120 to 230 distinct physical devices, depending on how wave 1 through 3 devices reused in wave 4 are counted, and correspondingly a small number, order tens, of genuinely independent same-model, different-physical-unit pairs for any given phone model. Second, because every device in this corpus was rented from a single AWS Device Farm physical lab, the corpus cannot speak to any question that depends on realistic diversity of real-world network location, carrier, or household; every device in the corpus is, from a networking perspective, effectively co-located. This second limitation becomes directly relevant in Section 9.5.

4 What Worked: Measured, Real Improvements

Before presenting the paper's central negative result, we document three interventions that produced genuine, reproducible, real-device-measured improvements over the naive weighted-average baseline. We include this section for two reasons: to be fair to the parts of the investigation that succeeded on their own terms, and because understanding why these interventions worked is a necessary prerequisite for understanding why the remaining interventions attempted in Section 9 did not.

4.1 Integrating a third-party whole-fingerprint hash

FingerprintJS is an open-source library that computes a single composite identifier, called a visitor ID, from dozens of underlying browser signals (canvas rendering, audio rendering, WebGL parameters, installed fonts, math library precision characteristics, and more), combined via a hashing procedure the project's own scoring logic does not control or fully audit. Measured on the wave 4 campaign, comparing FingerprintJS's visitor ID directly against the project's own weighted score, the visitor ID agreed on 99.1 percent of genuine same-device pairs (n = 109) and only 4.5 percent of same-model-different-unit pairs (n = 22), a separation of 94.6 percentage points between the two conditions, the widest separation of any single signal measured in this project. The next best single field, a canvas hash computed against complex Unicode script rendering, separated the two conditions by 57.7 percentage points.

We integrated the visitor ID into the weighted score as an additional, heavily weighted check (weight 50 out of a maximum possible score on the order of several hundred, deliberately not made dominant, both because the underlying algorithm is a third-party black box whose behavior could change on a version bump, and because its resistance to a motivated adversary attempting to spoof it is unknown, unlike the project's own signals, whose spoofing difficulty could at least be reasoned about directly). We further added an asymmetric flat penalty (50 percentage points, applied after the weighted average rather than folded into it, and floored at zero) when the visitor ID is present on both sides of a comparison but disagrees, on the reasoning established by the same measurement: a visitor ID mismatch is nearly as informative, in the opposite direction, as a visitor ID match, given that it disagreed on only 0.9 percent of genuine same-device pairs.

This single change was, in isolation, an unambiguous, no-tradeoff improvement on the wave 4 dataset: true-positive rate at the project's threshold rose from 95.5 to 98.2 percent, cross-population false-positive rate fell from 1.48 to 0.29 percent, and the same-model, different-unit false-positive rate, the specific case the threshold exists to survive, fell from 11.5 to 7.7 percent. All three metrics moved in the correct direction simultaneously, which is unusual; most interventions attempted in this project traded one metric against another.

We understand the mechanism to be structural rather than incidental. Every check in the project's own weighted average gives partial credit: two devices can agree on some fields and disagree on others, and the final score is a blend. FingerprintJS's visitor ID instead hashes dozens of raw signals together into one combined value, so a match requires near-total agreement across all of them simultaneously; there is no partial credit for two different iPhones of the same model that happen to report an identical GPU string and core count, the way the project's own model-class fields freely give. This is precisely why the visitor ID separates the same-model, different-unit case so much more sharply than any individual field the project computes itself.

The wave 4 payload also collected a hash from a second open-source library, ThumbmarkJS, purely for comparison against FingerprintJS and the project's own score, never wired into production scoring. We re-ran this comparison directly against the real wave 4 corpus rather than relying on the original one-off measurement, and confirm ThumbmarkJS performs comparably to FingerprintJS on this dataset: 98.2 percent exact-hash agreement on same-device pairs (n = 110), against 0.1 percent on all cross-device pairs (n = 5,253) and 4.3 percent on the same-model, different-unit case (n = 23), a separation profile close to FingerprintJS's own (99.1, 1.0, and 4.5 percent respectively on the same corpus). We record this as a real, measured, low-effort opportunity the project did not act on before its cancellation, integrating ThumbmarkJS the same way FingerprintJS was integrated in Section 4.1 would very likely have produced a similar improvement, not as a criticism of the original integration choice, which had to pick one library to validate and commit to first.

4.2 Absolute score caps for well-evidenced failure modes

A second successful pattern was the use of an absolute cap or penalty on the final score, triggered by a specific, independently well-evidenced failure condition, rather than a general reweighting of the fields involved. Concretely: when both sides of a comparison are iOS devices and their reported screen width and height genuinely disagree, the final score is capped at 40 percent regardless of how high the underlying weighted average would otherwise compute, on the reasoning that two different physical screen panels is close to unambiguous evidence of two different physical devices, and no accumulation of coincidental agreement on other fields should be allowed to override that fact.

This cap reduced wave 4 cross-population false-positive rate from 0.73 to 0.29 percent and the wave 1 through 3 dataset's cross-population false-positive rate from 2.01 to 1.53 percent, in both cases at zero measured cost to true-positive rate. The FingerprintJS mismatch penalty described above follows the identical pattern: both work because they key on a piece of evidence that is independently strong enough to override a coincidentally high blended average, rather than attempting to make the blended average itself behave better through a different weight distribution.

4.3 Vendoring over reinventing

We note, as a general engineering observation rather than a novel technical finding, that vendoring a well-chosen third-party library outperformed attempting to reconstruct its underlying mechanism from first principles. FingerprintJS's whole-fingerprint hashing approach outperforms the project's own per-field weighted average on the single hardest measured case (same-model, different-unit iOS pairs) by a wide margin, and integrating it required one additional entry in the checks table and one penalty block, not new architecture. We record this because it is easy, when a custom-built system underperforms, to assume the fix requires more custom engineering; in this instance the fix was adopting an existing, independently engineered artifact instead.

5 The Core Mechanistic Finding: Correlated Entropy Suppression on iOS Safari

The single most important empirical finding of this project, and the one that ultimately bounds every remediation attempted in Section 9, is that the project's remaining false positives are not spread evenly across many weakly informative signals. They are concentrated in one specific, well-characterized population (different physical iPhones of the same model, compared under Safari), and within that population, the failure is not confined to one signal; it appears simultaneously across nearly every signal the system has.

The clearest single illustration of this predates the campaign-wide blame report below: an early real-device A/B test between two distinct physical Samsung Galaxy S25 units, same model, different owners in principle, matched on every model-class field the scorer checks (GPU renderer, CPU core count, screen geometry, codec support, and the rest of the fixed-by-spec-sheet block) and scored 85.6 percent overall under the pre-recalibration weights in place at the time, almost entirely on the strength of fields that carry zero same-device evidence and only ever prove same-model. This one pair is what motivated splitting model-class fields out from per-unit-capable ones in the first place, and every subsequent finding in this section is that same mechanism measured at the scale of a full campaign rather than one pair.

We measured this directly using a blame report that, for each of the 40 false positives remaining in the wave 4 dataset at the project's chosen threshold, recorded which individual fields were the

ones that happened to agree. A canvas rendering hash appeared as an agreeing field in 39 of the 40 false positives. A canvas hash computed specifically against complex Unicode script rendering appeared in 38. An audio rendering hash appeared in 38. FingerprintJS's own visitor ID, despite the strong separation reported in Section 4.1, still appeared as an agreeing field in 37. These are not four independent weak signals that happen to add up; they are the same handful of physical iPhones agreeing on nearly everything simultaneously, and 35 of the 40 false positives (87.5 percent) were comparisons between two different iOS devices.

We re-ran this same blame report while preparing this postscript, against the current codebase (including the iOS screen-geometry cap, the FingerprintJS integration, and the postscript's own `dig()/isIOS` fixes described in Section 13.3), to check whether the pattern was an artifact of the mid-pipeline snapshot the original 40-false-positive report was taken from. It is not. Wave 4's residual cross-device false-positive count under the fully-patched scorer is 16, down from the 40 analyzed above, and the same four fields still dominate the smaller residual set in nearly the same proportions: canvas rendering hash appears in 15 of the 16, complex-script canvas hash in 14, audio rendering hash in 14, and FingerprintJS's visitor ID, despite being individually the widest-margin discriminator this project measured, still appears in 13. Shrinking the false-positive count by a factor of 2.5 did not change which signals are responsible for what remains; it confirms the same handful of correlated failures are what is left once everything else has been fixed, which is the structural-ceiling reading the rest of this section argues for, not a byproduct of measuring an intermediate state.

We attribute this to Apple Safari's anti-fingerprinting hardening operating across multiple signal families at once, far more aggressively than the hardening present in Chrome or a generic WebView. Concretely: Safari applies noise or quantization to canvas rendering output; caps the `hardwareConcurrency` value reported to JavaScript at 4 regardless of the device's actual physical core count; reports a generic "Apple GPU" string regardless of the specific chip generation in use; and restricts font enumeration relative to other browsers. Each of these hardening measures independently suppresses entropy that a corresponding signal depends on, and because they are applied simultaneously by the same browser engine to the same device, the signals that depend on them fail together, not independently. A system that scores several of these signals and blends them, hoping that independent partial agreement is unlikely by chance, is defeated by the fact that the agreement is not independent.

One signal was found to survive this suppression nearly intact: reported screen width and height. Real device geometry is not something Safari can normalize away without visibly breaking every website's responsive layout, so it remains stable across repeated visits from the same physical device and reliably differs across models that have genuinely different physical screens. This is the empirical basis for the absolute cap described in Section 4.2, and it is, as far as this project was able to determine, the strongest single surviving signal for distinguishing different iPhone models under Safari.

However, screen geometry has its own structural ceiling, and this ceiling is not fully closed by anything built during this project. Apple frequently reuses the same physical screen panel across sibling and successive iPhone models: as one concrete example found in the project's own device roster, the iPhone 14 Pro, 15, 15 Pro, and 16 all report an identical 393 by 852 point screen, and the iPhone 16 Pro, 17, and 17 Pro all report an identical 402 by 874 point screen. When two different iPhone models happen to share an identical screen panel, the screen-geometry cap has no discriminating information left to key on, and the comparison falls back to the same degenerate, correlated-agreement collision problem described above. Thirteen of the sixteen false positives remaining in the wave 4 dataset after every intervention described in Section 4 was applied are exactly this case: two different iPhone models that happen to share a screen panel.

We regard this as a genuine, currently unclosed structural ceiling in the available signal set, not a tuning problem. Closing it further would require either a genuinely new iOS-available signal that Safari's hardening does not suppress, which this project searched for (Sections 6.3 through 6.5) and did not find, or accepting a materially higher false-positive rate specifically for comparisons between iPhone models that share a screen panel.

6 What Did Not Work: Rejected Interventions

This section documents four interventions that were designed, implemented, measured against real device data, and rejected, together with the specific measurement that led to rejection in each case. We include the negative results in the same level of detail as the positive ones in Section 4, on the view that a rejected hypothesis with a documented, specific reason for rejection is exactly as valuable to a future engineer as an accepted one, and considerably more valuable than an undocumented one that gets silently re-attempted.

6.1 Proportional reweighting of iOS per-unit signals

The first hypothesis tested, given the iOS same-model collision problem described in Section 5, was that the fields responsible were simply overweighted relative to the fields that discriminate well, and that halving the combined weight of the `perunit_hash` and `thirdparty_hash` field groups specifically for iOS-to-iOS comparisons would rebalance the blended score away from the problem.

This did not work. The confusion matrix did not move at all: the same 40 of 40 false positives that existed before the reweighting still existed after it, and the cross-population mean false-positive rate nudged slightly worse rather than better. The mechanism, once understood, is straightforward: halving a field group's weight changes how much that group influences the final blended score, but it does not change that group's own underlying match rate. On iOS specifically, the model-class fields (GPU renderer string, core count, platform string) that were not being reweighted turn out to agree at an equally high rate as the per-unit fields being reweighted, because Safari reports the same generic values for these fields regardless of the specific physical unit. Shrinking the per-unit group's influence on the blend simply handed proportionally more influence to an equally compromised group. The general lesson we take from this: a proportional reweighting can only help when it changes which field group dominates the blended score; it cannot help when every field group available is, to a similar degree, compromised by the same underlying suppression mechanism.

6.2 Automated per-field weight optimization

The second hypothesis was that the twelve-to-thirteen coarse field-group weights described in Section 2.2 were too coarse an instrument, and that automatically tuning all roughly 110 individual field weights via gradient-based optimization, with leave-one-cluster-out cross-validation and L2 regularization to control overfitting, would find a better operating point than manual group-level tuning had.

We built this optimizer, ran it against the real campaign data, and evaluated it on held-out clusters not used for fitting. The result: true-positive rate genuinely improved under cross-validation, from 94.6 percent (standard deviation 15.1 across folds) to 99.1 percent (standard deviation 4.8). False-positive rate, however, was identical before and after optimization, to one decimal place, both in-sample and on held-out data (17.1 percent and 17.3 percent respectively in both conditions). With roughly 110 free parameters available to fit against only 250 to 470 real hard-case pairs (the same-model, different-unit and cross-cluster comparisons that constitute the difficult part of this

classification problem), the optimizer had enough degrees of freedom to find weight assignments that better fit the true-positive side of the problem, which has comparatively more data to fit against, but not enough real data to discover any genuine improvement on the false-positive side that the coarser twelve-group search had not already found. We conclude that group-level reweighting, with roughly a dozen free parameters, is the appropriate level of freedom for the volume of real data this project was able to collect, and that moving to field-level optimization without a substantially larger volume of real same-model campaign data would not be expected to help, and did not help when tried.

One design choice in the optimizer is worth recording on its own, independent of the overfitting finding above: it refuses to reduce the DRM and security-attestation field group below a fixed floor, regardless of what the objective function says. The reason is structural rather than a tuning safeguard. Every pair in the campaign dataset is a genuinely honest device; nobody in it was actually attempting to spoof a hardware DRM attestation or a platform authenticator. A field like Widevine hardware-backed decryption support earns its weight from being expensive to fake, not from separating two honest phones well, and those are different axes that an optimizer fit purely against honest same-device/cross-device separation cannot see: on this dataset, a hard-to-forge field that two honest strangers with the same phone model both happen to report identically looks exactly like dead weight, and an unconstrained optimizer would zero it out. We regard this as a general lesson about applying purely statistical weight optimization to a signal set that has to survive adversarial as well as honest traffic, not specific to this one field group.

6.3 CPU-timing ("silicon lottery") fingerprinting

A candidate novel signal, sometimes described in prior work as exploiting per-unit manufacturing variance in dynamic voltage and frequency scaling behavior under sustained JavaScript computational load (colloquially, a "silicon lottery" or CPU-Print approach), was implemented as an experimental probe and measured on the wave 4 campaign. We found that only 7.5 percent of the variance in this probe's output across the dataset was attributable to genuine per-physical-unit variation beyond simply knowing the device's model; 55.8 percent of the total variance was explained by model identity alone, meaning the probe is mostly a noisy, computationally expensive way of re-deriving information the GPU renderer string and core count already provide for free. Concretely, the mean measured delta between two different physical units of the same model (0.127) was statistically indistinguishable from the mean delta between repeated measurements of the same physical unit (0.116). This signal was not wired into scoring. Given its measured lack of discriminating power and its roughly two-second collection cost, we consider it unlikely to be worth revisiting without a substantially different measurement methodology, though the raw data remains collected in case it proves useful for an unrelated purpose later.

6.4 Clock-oscillator ("Clock Around the Clock") fingerprinting

A second candidate signal, based on prior published work exploiting fine-grained timing differences in a device's clock oscillator, was implemented and measured at the full 105-device scale of the wave 4 campaign. This signal was found to be completely dead on modern real hardware: every single pair measured, same-device, cross-device, and cross-cluster alike, produced an exact-zero timing delta, with no exceptions across the entire dataset. We attribute this to modern browsers' Spectre-era coarsening of `performance.now()` timer resolution, introduced as a side-channel mitigation, having fully eliminated the sub-millisecond timing precision this technique depends on. We recommend against collecting this signal at all in any future work, not merely against scoring it, since it

consumed on the order of 500 trials of runtime per submission for zero measured information content.

6.5 WebGPU `timestamp-query` execution-unit timing

A third candidate signal, targeting the same open problem as Sections 6.3 and 6.4 (a single-shot signal capable of separating two physically identical units of one phone model) but aimed at manufacturing variance in GPU execution units rather than CPU frequency scaling or clock oscillators, was also built and included in the wave 4 collection payload: a WebGPU compute shader dispatched forty times per submission, timed via the `timestamp-query` feature where the browser and adapter grant it, rather than the coarser CPU-side wall-clock timing the project's ordinary GPU compute benchmark falls back to. Unlike the two signals discussed above, this one was collected on every wave 4 submission but never analyzed at the time; the project's comparison tooling built out for FingerprintJS, ThumbmarkJS, CPU-timing, and clock-oscillator timing (Section 4.1, Sections 6.3 and 6.4) has no equivalent code path for it, and it went unexamined until this postscript's own re-reading of the collection code surfaced the gap. The measurement below was therefore run now, against the same real wave 4 corpus, rather than at the time of original data collection, and we flag that explicitly since it changes when the finding was made, not what the data itself says.

The first result is about availability rather than discrimination. `timestamp-query` requires WebGPU itself, then the specific feature flag on top of it, and 96 of wave 4's 224 real submissions (42.9 percent, split 50 Android and 46 iOS) produced no usable measurement at all: 94 of those never reached the point of requesting the feature because `navigator.gpu` itself was absent, and 2 reached a granted WebGPU adapter that specifically declined the `timestamp-query` feature. Independent of anything the timing itself shows, a signal unavailable on 43 percent of real devices in this project's own campaign is a real deployment limitation on its own.

Where a real measurement was available, same-device pairs (n = 64) showed a median relative difference of 0.03 percent between visits, consistent with a stable, highly repeatable measurement. Cross-device pairs generally (n = 1,830) differed by a median of 82.7 percent, an enormous gap, but one we read as showing this measurement is dominated by raw GPU compute throughput (a property of which GPU model a device has, not of the specific physical unit) rather than any manufacturing-variance signal: a decomposition of variance in first-visit `medianNs` by device model alone explains 54.7 percent of total cross-device variance, closely matching CPU-Print's own 55.8 percent finding in Section 6.3, and for the same underlying reason, a fixed compute workload simply takes a different absolute amount of time on different GPU classes, which is a model-class fact expressed as a continuous number rather than a unit-level one.

The specific question this signal exists to answer, whether two different physical units of one model separate from each other, is the one case where this measurement is genuinely inconclusive rather than clearly positive or negative. The same-model, different-unit pairs (n = 11, the same order-of-magnitude sample size limitation Section 3 and Section 10 already document for this project's entire real-device corpus) showed a median relative difference of 11.1 percent, closer to the same-device figure than to the general cross-device figure and, on its face, a more promising gap than either CPU-Print (Section 6.3, where same-model and same-device deltas were statistically indistinguishable) or CryptoFP (Section 6.4, completely dead) produced. We do not trust this reading. The mean relative difference for the same same-model pairs is 60.6 percent, more than five times the median, which is the signature of a small sample dominated by one or two outlier pairs rather than a clean, consistently reproducible separation; eleven pairs is far below the sample size Section 10 shows is needed to distinguish real signal from sampling noise for any claim this paper makes, and we regard drawing a conclusion from it, in either direction, as exactly the kind

of overreach Section 10 argues against. We record this as an open question this project's real data cannot currently answer, rather than as a third rejected candidate alongside Sections 6.3 and 6.4: a repeat of this measurement with a same-model sample at least in the low hundreds, using a relative or normalized timing metric analogous to CPU-Print's own `throttleRatio` rather than the raw absolute `medianNs` used here, would be the concrete next step, not a conclusion either way from what exists today.

6.6 Signals that actively backfired in production scoring

Sections 6.3 through 6.5 document three candidate signals evaluated as standalone research questions and never wired into scoring at all. A related but distinct category exists inside the roughly 110 fields that are shipped in production: individual per-unit-capable fields, included in the weighted score on the strength of a single early real-device pair, that a full campaign later showed separate same-device from same-model, different-unit pairs in the wrong direction, matching more often across two different physical units of one model than across two visits of the one device. We document these separately from Sections 6.3 through 6.5 because they were never rejected outright; each was demoted to a floor weight (1, the minimum a check in this system can carry) rather than removed, on the reasoning that a field measured wrong-signed is more useful at negligible weight, still visible to a moderator in the matched/mismatched diagnostic list, than deleted outright.

The clearest case is the worker parallel-scaling saturation point, a probe that spawns concurrent workers under load and measures where throughput stops scaling, intended to catch a spoofed `hardwareConcurrency` value and reveal heterogeneous core topology. It was a confirmed differentiator on the original single-pair test that motivated including it. The 2026-08-08 campaign (38 same-device pairs, 15 same-model pairs) overturned that, at -23.3 points of separation, the single worst-performing field measured that campaign; the independent wave 4 re-run confirmed the same direction again on a different device set, at -12.6 points. Two independent campaigns agreeing a field actively hurts separation is a different and stronger finding than one campaign showing weak or no separation, and is why this field, alone among the demoted signals, was floored rather than merely down-weighted. We read the mechanism as thermal and background-load noise swamping the signal within a single device's own two visits more than it swamps the signal between two different physical chips, though this project did not run the controlled experiment that would confirm that explanation directly.

A second case, the integer kernel of the project's own WASM CPU benchmark, showed the same direction on the same two campaigns independently (-21.1 points on the 2026-08-08 data, -6.3 points on wave 4), while the benchmark's other three kernels (float, memory, branch) separated normally or close to neutrally. We regard this as informative in the same way Section 6.3's CPU-Print rejection is: a signal that sounds like it should carry real per-unit manufacturing variance can instead be dominated by session-level noise (thermal state, background load, JIT warmup) that swamps whatever silicon-level variance exists underneath it, and confirming that requires running the same measurement on independent real-device data twice, not reasoning about the mechanism once.

A third, smaller-sample case is worth flagging for a different reason: the WebGPU scene-render hash, one of the highest-weighted per-unit-capable fields, disagreed with itself across two visits of the identical device roughly three times out of four in the 2026-08-08 campaign, a same-device match rate of only 26.1 percent despite showing real separation (26.1 points) from the same-model comparison. A field this unstable against its own repeat visits cannot be trusted at high weight regardless of how well it separates in aggregate, since a coin flip that happens to correlate with model identity is not the same thing as a reliable per-unit signal, and it was demoted for that

reason specifically, independent of the worker-scaling and WASM-integer findings above.

6.7 Two provisional mechanisms, flagged rather than fixed

Two further mechanisms in the shipped scorer are explicitly marked provisional in their own code comments, in the sense that the direction of the fix is measured but the magnitude is not independently validated the way the mobile per-unit weights are, and we record both here because neither made it into this paper's earlier sections despite being real, dated findings from the same project history.

The first is `desktopModelClassBoost`, a multiplier applied to hardware-parts-list fields (GPU, CPU, RAM, panel, discrete peripherals) specifically when both submissions being compared look like desktop machines, on the reasoning that a self-assembled desktop's exact parts combination is closer to a per-unit identifier than a model identifier the way a mass-produced phone's specification is, since nobody else necessarily owns that exact combination of graphics card, core count, memory size, and display. The only labelled validation available is a single real hardware pair: an M2 Mac compared across three visits against a GTX 980 machine. Restricting the boost to hardware-parts-list fields rather than the full model-class block widened separation from 51.8 to 52.9 points on that one pair (the same-machine score rose 90.5 to 92.4, the different-machine score rose 38.7 to 39.5), a real but modest effect, and one entirely dependent on a sample size of one. The mechanism's own comment states plainly that it does not know how to handle the population that would break it worst, a fleet of identical corporate desktops, the same collapsed-entropy problem this paper documents for mass-produced phones, in desktop form, which nothing in `isDesktop` can currently distinguish from a genuinely unique self-built machine.

The second is `internal/rarity`, a corpus-measured signal-rarity multiplier built to weight a matching field by how much it actually narrows the field within the real submission population, rather than by a hand-assigned constant, using the same mathematics as inverse document frequency. It is implemented, actively counts every real submission into its statistics tables, and computes a shadow score logged alongside the live weighted score for comparison, but nothing in its own package enables it to affect a real verdict; that is a deliberate, separate, later decision the package's own top-of-file comment says should be made against labelled pairs, not against vibes. Reading the implementation directly surfaces a specific reason that later decision has not been made: the multiplier normalizes a value's surprisal by $\log_2$(number of distinct values observed for that field), which is the correct baseline only if the field's real distribution is close to uniform across those distinct values. It is not, for exactly the fields this paper's Section 5 already identifies as collapsed: a browser-build cluster where one value covers the overwhelming majority of the real corpus, with a long tail of rare values contributing almost nothing to the true entropy, still normalizes by the count of distinct values seen rather than by the cluster's actual empirical entropy, which overstates how typical the dominant value's rarity multiplier should be and understates how much a rare value in a high-diversity cluster should count. We did not independently re-measure the magnitude of this effect against real corpus data, since the package has never been enabled and doing so would require exactly the labelled-pairs validation its own comment says is the correct gate; we record the mechanism because it is a concrete, code-verified reason a corpus-scale rarity signal, which in principle could help exactly the collapsed-entropy problem Section 5 documents, remains unsafe to enable as written.

6.8 Manual halving of model-class weight, as a diagnostic, not a fix

Separately from the automated attempts above, we ran one purely diagnostic experiment: manually halving the weight of the model-class field group specifically, without any attempt to compensate elsewhere. This does reduce the same-model false-positive rate on the wave 1 through 3 dataset, from 17.1 to 8.6 percent, but does so purely by unconditionally trading away true-positive rate, which fell from 91.7 to 81.6 percent, with no attempt to preserve balance between the two. This moves the operating point along the same underlying receiver operating characteristic curve rather than producing a better curve, and we record it explicitly as a diagnostic that helped confirm the mechanism described in Section 6.1, not as a change that was ever shipped or considered viable on its own terms.

7 Threshold Sensitivity Analysis

The project's decision threshold, `SensitivityThreshold`, was recalibrated on 2026-08-08 alongside the per-unit weight changes described above, using 38 genuine same-device pairs (measured on a second visit, with the persistent identifier described in Section 2.2 deliberately stripped so as not to trivially inflate the result) and 15 same-model, different-unit pairs. Under the recalibrated weights, same-model pairs topped out at 82.9 percent and same-device pairs bottomed out at 77.9 percent, with all but five of the 38 same-device pairs scoring 83 percent or above. A threshold of 83 sits exactly at this natural gap: 0 percent false-positive rate on same-model, different-unit pairs under this specific 38/15-pair calibration set, down from 26.7 percent at a threshold of 78 under the same new weights, while still correctly recognizing 86.8 percent of genuine repeat visits.

We subsequently ran a full sweep of the threshold from 0 to 100 against the complete wave 4 dataset (105 devices, 224 submissions), reproduced in full in Table 1. Three features of this sweep are load-bearing for the rest of this paper.

Threshold	TPR (%)	FPR, all pairs (%)	FPR, same-model pairs (%)
0	100.0	100.00	100.00
10	100.0	13.68	76.92
20	100.0	7.95	26.92
30	99.1	6.67	19.23
40	99.1	5.33	19.23
50	99.1	3.83	19.23
55	99.1	2.45	19.23
60	99.1	1.30	19.23
65	99.1	0.92	19.23
70	99.1	0.66	19.23
72	99.1	0.57	15.38
74	99.1	0.46	11.54
76	99.1	0.37	7.69
78	99.1	0.31	7.69
80	99.1	0.31	7.69
81	99.1	0.31	7.69
82	99.1	0.31	7.69
83	98.2	0.29	7.69
84	97.3	0.27	7.69
85	95.5	0.22	7.69
86	92.9	0.15	0.00
87	92.9	0.15	0.00

Threshold	TPR (%)	FPR, all pairs (%)	FPR, same-model pairs (%)
88	89.3	0.15	0.00
89	83.0	0.15	0.00
90	77.7	0.13	0.00
92	54.5	0.05	0.00
94	34.8	0.04	0.00
96	15.2	0.02	0.00
98	9.8	0.00	0.00
100	1.8	0.00	0.00

Table 1: Full receiver operating characteristic sweep, wave 4 dataset (105 real devices, 224 submissions), measured directly against the production comparison function. TPR is true-positive rate, FPR is false-positive rate. The same-model column has a denominator of 26 real same-model, different-unit pairs.

First, thresholds 78 through 85 are entirely identical on same-model false-positive rate, all sitting at 7.69 percent (2 of 26 pairs). Moving the threshold anywhere within this band costs true-positive rate for zero benefit on the specific hard case the threshold exists to address. The two residual same-model false positives in this band are the same two pairs throughout the range; they are, not coincidentally, the same pairs on which FingerprintJS's own visitor ID collided (Section 4.1), and they score in a narrow band around 85 to 86 percent, not spread across the range.

Second, there is a genuine cliff at 86: same-model false-positive rate drops from 7.69 to 0.00 percent, eliminating both residual same-model false positives, but at a real, non-trivial cost: true-positive rate falls from 98.2 to 92.9 percent, meaning approximately six additional genuine same-device pairs, out of the roughly 109 measured in this dataset, are newly missed. The project's decided default sensitivity setting moved to this threshold on the reasoning that same-model, different-unit collisions route to human review rather than automatic banning by default, so the true-positive cost is "moderators see six fewer flagged pairs," not "an evader walks free with certainty," and eliminating the entirety of a well-characterized false-positive class was judged a clean, evidence-backed trade worth making.

Third, and this is the finding most directly relevant to a request, considered and rejected during this project's final week, to simply raise the threshold much further as a way of addressing the population-scale problem described in Section 8: true-positive rate collapses well before same-model false-positive rate reaches any additional useful floor below the 86 value. At a threshold of 98, true-positive rate is 9.8 percent: over 90 percent of genuine repeat visits by an actual returning device now score below the threshold and are missed entirely. This is not a stricter version of the same product; it is a device-matching system that essentially never recognizes a genuine returning device. The mechanism is that genuine same-device pairs do not reliably score anywhere near 98 percent in the first place, owing to ordinary measurement noise (the 0.9 percent same-device mismatch rate on FingerprintJS's own visitor ID, each occurrence of which costs a flat 50-percentage-point penalty, plus minor day-to-day drift from browser updates and environmental differences), so raising the threshold past approximately 86 to 90 trades true-positive rate for essentially no further same-model false-positive benefit, since that benefit was already exhausted by 86.

We additionally note, and return to this point formally in Section 10, that the "0.00 percent" same-model false-positive rate reported for every threshold from 86 through 98 in Table 1 is computed over the same 26 real same-model pairs throughout. Zero observed false positives out of 26 trials does not establish that the true underlying rate is actually zero, or even close to zero; it establishes only an upper bound on the true rate, and that upper bound, as we show formally in

Section 10, is far too loose to certify safety at the scale Section 8 requires.

8 The Population-Scale 1:N False-Positive Problem

This section presents the paper's central negative result. Every measurement in Sections 4 through 7 concerns the behavior of a single pairwise comparison. The general matcher, as specified in Section 2.1, does not perform a single pairwise comparison; it performs N of them, one against every existing submission in the guild, and flags the visitor if the single best-scoring comparison among those N clears the threshold. This section shows that this change, from evaluating one comparison to evaluating the maximum of N comparisons, is not a cosmetic difference. It fundamentally changes what false-positive rate is acceptable, and it changes it far more severely than intuition suggests.

8.1 The compounding mechanism

If a visitor's true match status is negative (they are not the same device as anyone in the database) and each of the N comparisons the system performs against them were governed by an identical, independent per-pair false-positive probability p, then the probability that at least one of the N comparisons incorrectly clears the threshold is

$$P(\text{at least one false flag}) = 1 - (1 - p)^N.$$

This quantity grows toward 1 rapidly as N grows, even for small p. At $p = 0.003$ (0.3 percent, a plausible-looking single-pair rate by the standards of Table 1), $N = 10{,}000$ already gives $1 - (1 - 0.003)^{10000} \approx 1.0$: a visitor compared against ten thousand accounts at a 0.3 percent per-pair false-positive rate is, under the independence assumption, almost certain to be flagged by at least one of them, even though no single comparison looked dangerous in isolation.

8.2 Real deployment behavior is not independent, and the general case is better than the naive estimate

The independence assumption above is not realistic, and we built a dedicated simulation, described here in full because it directly determines whether the naive estimate above is representative, to measure how the real, non-independent comparison function actually behaves as N grows. The simulation called the project's real `fingerprint.Compare` function unmodified against real device data (never synthetic fingerprints), at the project's real production threshold, with a bootstrap procedure (1,200 resamples per N value, split into 40 batches, percentile confidence intervals from the batch-level flagged fractions) to estimate both a mean flagged rate and a confidence interval at each population size N.

For the representative, current-production condition (wave 4 data only, so that FingerprintJS is present on every comparison, since it was entirely absent from earlier waves), sampling the comparison database uniformly at random (the "general" scenario, meant to model a generic, non-adversarially-composed set of other server members), the measured false-positive rate was substantially better than the naive independence estimate at every population size tested, reproduced in Table 2.

N	Mean FPR (%)	90% interval	Mean triggers	Naive estimate (%)
10	2.8	[0.0, 10.0]	0.03	17.5
20	4.8	[0.0, 10.0]	0.06	31.9

N	Mean FPR (%)	90% interval	Mean triggers	Naive estimate (%)
50	10.4	[3.3, 20.0]	0.13	61.7
100	15.8	[6.7, 23.3]	0.27	85.3

Table 2: General (uniformly random) scenario, wave-4-only data (representative of current production, since FingerprintJS is present on every comparison). FPR is false-positive rate; mean triggers is the mean number, per visitor, of the N comparisons that individually clear the threshold. Naive independence estimate uses $1-(1-0.019)^N$, where 1.9 percent is wave 4's original pre-fix pairwise cross-population false-positive rate.

This gap, 15.8 percent actual against an 85.3 percent naive estimate at $N = 100$, a factor of 5.4, is the reason the naive independence calculation in Section 8.1 is misleading on its own: real collisions concentrate almost entirely within same-model pairs rather than spreading independently across all pairs, and a uniformly random sample of N accounts is mostly composed of different device models than the visitor, which carry near-zero real collision risk under the project's own scoring function. This is the same underlying mechanism documented in Section 5: the failure is correlated and concentrated, not spread thin, and a correlated, concentrated failure mode produces a lower aggregate rate under uniform random sampling than an independent one would, precisely because most of the N comparisons are drawn from the safe part of the distribution.

8.3 Why the general case does not describe a real deployment at scale

The result in Table 2 was, at an earlier stage of this project, read as reassuring: real production semantics appeared to make the population-scale problem substantially less severe than a naive calculation would suggest. We want to be explicit that this reading was incomplete, and correcting it is itself part of this paper's negative result.

The uniformly-random sampling used to produce Table 2 draws from this project's entire available real-device pool, which contains at most 120 to 230 distinct devices (Section 3), deliberately curated across the campaign's design for cross-model diversity rather than to reflect a realistic population's device-model concentration. A real Discord server with thousands of members does not draw its device mix from a small, deliberately model-diverse research panel. It draws from the real retail smartphone market, which is not uniformly spread across many models; ownership concentrates heavily on a small number of currently or recently popular models within any given region and demographic. This means that, unlike this project's research panel, a real population of thousands of members will, for any individual visitor, very plausibly contain dozens to hundreds of other members who happen to own the exact same phone model as that visitor, purely as a consequence of ordinary market popularity, with no adversarial intent required from anyone.

This observation connects directly to the adversarial scenario this project's simulation also measured: instead of sampling the comparison database uniformly at random, a fixed fraction of the database is forced to share the visitor's exact device model, modeling exactly the situation a real, realistically model-concentrated population produces on its own. Table 3 reports these results for the wave-4-only, current-production-representative dataset.

N	10% same-model (%)	25% same-model (%)	50% same-model (%)
10	10.3	13.0	11.8
20	13.9	16.1	12.4

N	10% same-model (%)	25% same-model (%)	50% same-model (%)
50	23.5	19.3	15.6
100	27.3	24.4	22.2

Table 3: Adversarial (same-model-concentrated) scenario, wave-4-only data, mean false-positive rate at each population size and same-model fraction.

Even a modest 10 percent same-model concentration in the comparison database pushes false-positive rate at $N = 100$ to 27.3 percent, worse than the naive flat-rate estimate would predict for a database of that size, and dramatically worse than the 15.8 percent measured under uniform random sampling in Table 2. We regard this as the single most important reframing this project arrived at: the protection the general scenario appears to offer inverts almost immediately once the comparison database is not artificially model-diverse, and a real deployment's comparison database, being drawn from the real phone market rather than a curated research panel, should be expected to look considerably more like Table 3 than Table 2 once population size reaches the thousands a real server membership implies. We note one counterintuitive detail in Table 3 for completeness rather than smoothing it over: false-positive rate does not increase monotonically with same-model fraction (22.2 percent at 50 percent concentration versus 27.3 percent at 10 percent concentration, both at $N = 100$); we attribute this to small-sample noise in the same-model pool itself (typically only two to four real physical units per model cluster, so higher forced concentrations require more with-replacement reuse of the same one or two real device fingerprints, which does not add new independent risk beyond what a single copy already contributes) and consider the fine-grained ordering among adversarial fractions at this sample size unreliable, while the general direction, that adversarial concentration performs far worse than uniform random sampling, is solid.

8.4 Manufacturer and platform concentration

Breaking the general (uniform random) scenario down by manufacturer at $N = 100$ makes the platform-specific severity of this problem explicit. On the wave-4-only, current-production-representative dataset: Apple devices showed a mean false-positive rate of 47.3 percent (90 percent interval 30.0 to 60.0, n = 25 devices), Google Pixel devices 15.3 percent (6.7 to 23.3, n = 30), and Samsung devices 0.0 percent (n = 44). Apple is, by a wide margin, the dominant source of remaining population-scale risk, entirely consistent with the mechanistic finding in Section 5: Safari's cross-signal entropy suppression is an iOS-specific phenomenon, and it manifests here as the platform whose devices are, in aggregate, the hardest to tell apart even before any adversarial concentration is applied.

We separately measured the same manufacturer breakdown on the combined-wave dataset (waves 1 through 3 plus wave 4 together, necessary to reach larger N values but methodologically diluted, since roughly half of all comparisons in the combined dataset predate FingerprintJS's integration and therefore lose its benefit entirely) and found strikingly different figures: Apple 50.5 percent (n = 31), Pixel 81.5 percent (n = 38), Samsung 43.5 percent (n = 48). The order-of-magnitude difference in Pixel's figure between the two measurements (15.3 percent representative-of-production versus 81.5 percent with FingerprintJS diluted out) is itself a striking, if indirect, confirmation of how much of the system's real discriminating power rests on the single integration described in Section 4.1; removing it from roughly half the comparison pool does not degrade performance gently, it changes which manufacturer looks like the dominant risk entirely.

8.5 Restating the negative result

Combining the findings of this section: the number of comparisons a real deployment of the general matcher performs per decision is not the tens this project could measure directly, it is the full membership of a real Discord server, which for an active community is routinely in the thousands. At that scale, for any device model with meaningful market share, real membership composition should be expected to resemble the adversarial scenario in Table 3 rather than the uniform scenario in Table 2, not because of any attacker, but because ordinary retail market concentration produces exactly that effect on its own. Under that scenario, the measured false-positive rate already exceeds 20 percent at $N = 100$ and, by the compounding mechanism in Section 8.1, does not plateau; it continues rising toward saturation as the true number of same-model peers in a real population grows into the hundreds, which is entirely plausible for the top handful of phone models in a server with thousands of members. We regard this as a genuine, currently unresolved blocker to shipping the general matcher as specified, independent of any single threshold or weighting choice, for the reasons detailed in the next section.

9 Attempted Remediations and Why Each Failed

Having established the population-scale problem in Section 8, this project spent its final week attempting to design around it without abandoning the core requirement (comparison against a server's full membership, not a narrowed or time-windowed subset, since the product's purpose is specifically to catch evasion that a narrow recent-activity window would miss). This section documents each remediation attempted, in the order attempted, together with the specific reason each one failed.

9.1 Raising the decision threshold alone

Addressed fully in Section 7. Raising the threshold high enough to plausibly help at population scale (a threshold of 98 was tested directly) destroys true-positive rate first, before the same-model false-positive rate reaches any additional useful floor beyond what threshold 86 already achieves. This remediation was rejected on that basis alone, independent of the statistical-power concern raised in Section 10.

9.2 Gating on an auxiliary exact-match signal before scoring

The next hypothesis was a two-stage design: before running the full weighted comparison against a candidate, first check a cheap, high-precision exact-match signal, and only run the expensive fuzzy comparison, or only allow the fuzzy comparison to count toward a flag, for candidates that pass this first gate. This is a well-established general pattern in record-linkage and deduplication systems, sometimes called blocking, where a cheap key reduces an expensive quadratic comparison problem to a much smaller number of within-block comparisons.

We evaluated the two exact-match signals available in this project's own signal set as candidate gates. The first, FingerprintJS's visitor ID compared for exact string equality rather than folded into the weighted average, has a measured 4.5 percent collision rate on real same-model, different-unit pairs (Section 4.1). This is not a safe gate: applying the compounding formula from Section 8.1 with this measured rate, $1 - (1 - 0.045)^k$, reaches 50 percent at only $k \approx 15$ same-model peers and 99 percent at $k = 100$, figures easily reached by a single popular phone model in a server of a few thousand members. Moreover, gating on this signal does not add meaningful independent

protection beyond the weighted score itself: an exact visitor-ID collision already contributes most of what is needed to clear the ordinary weighted threshold (the 50-point match weight, combined with avoiding the 50-point mismatch penalty, is a 100-point swing on its own), so the two events, "visitor ID collides" and "weighted score clears threshold," are highly correlated rather than independent, and a two-stage design built on this pair collapses to approximately the same outcome as gating on nothing.

The second candidate, the persistent storage identifier described in Section 2.2, is by construction essentially collision-free between different real people (a 256-bit server-issued random token has no realistic collision probability at any population size this project could ever reach). This is a genuinely safe gate from a false-positive perspective. It is, however, trivially defeated by any user who clears their browser's cookies, `localStorage`, and `IndexedDB` before creating a second account, which is one of the most basic and commonly known steps in deliberate ban evasion. A gate that is safe against false positives but has near-zero recall against the specific adversarial population the feature exists to catch does not solve the problem; it avoids it by declining to detect most of the real cases.

We conclude that no exact-match signal currently collected by this project is simultaneously safe against coincidental collision at realistic population scale and resistant to deliberate evasion, and that this is not a signal-selection problem solvable by choosing a different field; it follows from the same structural finding in Section 5, that every signal this project measures is subject either to correlated collision on genuinely different same-model devices, or, in the one case that is not (the persistent identifier), to trivial removal by anyone motivated to avoid detection.

9.3 Requiring network-address corroboration

A third hypothesis, motivated by the observation that the failures documented in Sections 5 through 9.2 all share a common root cause (browser-API entropy suppression), was to require a second, mechanistically unrelated signal to corroborate a device match before it counts toward a flag at population scale. The clearest available candidate already present in the system's own data model, though never previously used for positive matching, was the requesting client's network address, stored on every submission specifically for a different purpose (a reduced-confidence flag when a submission originates from a VPN or hosting provider IP range). Network-address overlap between two unrelated real people is, in principle, a collision mechanism rooted in network topology, which has no obvious relationship to canvas rendering noise, GPU string genericization, or any of the browser-API hardening documented in Section 5, and combining two genuinely independent per-pair rates multiplicatively, rather than relying on either alone, is exactly the kind of decorrelation that could in principle rescue the compounding problem in Section 8.1.

This remediation could not be validated, and we now believe it may not even be a real decorrelation for this project's target population, for two separate reasons.

First, and decisively for measurement purposes, every device in this project's entire real-device corpus was rented from a single AWS Device Farm physical device lab (Section 3). Any network-address overlap measured from this data would measure only whether two device-farm rental sessions happened to route through the same facility's infrastructure, which is a certainty by construction and has no relationship whatsoever to whether two real, independent Discord members share network locality. No amount of additional Device Farm testing budget changes this; it is a wrong-population problem, not a small-sample problem, and it cannot be fixed by collecting more data of the same kind.

Second, even setting the measurement problem aside, we have a specific, concrete reason to suspect the true rate would be worse than hoped for exactly the population this product targets.

Carrier-grade network address translation, a widely deployed practice among mobile network operators, pools large numbers of unrelated mobile subscribers behind a small number of shared public network addresses. For a fingerprinting product whose entire premise is mobile-device identification, this means two total strangers on the same mobile carrier in the same region could plausibly share an observed network address or address range at a materially higher rate than two strangers on independent home broadband connections would, which is the opposite of the decorrelation this remediation requires. We were not able to measure this rate either, for the same reason given above, but we regard it as a serious enough concern, combined with the unmeasurability problem, that we do not consider this remediation viable without a real production deployment generating genuinely diverse network data, which raises its own separate question of whether it is responsible to deploy an unvalidated matching mechanism in order to gather the data needed to validate it.

9.4 Population-size-adjusted (Bonferroni-style) threshold

The fourth and most structurally sound remediation attempted treats the problem statistically rather than by searching for a new signal. Rather than a single global threshold applied uniformly regardless of how many comparisons a given decision involves, a decision-time threshold can be computed per visitor as a function of k, the number of same-device-model candidates present among that visitor's specific comparison set (a quantity already known at decision time, since the full N-way comparison has already been computed as part of the product's basic operation), chosen high enough that the compounded false-positive probability for that specific k, using the already-measured relationship between threshold and same-model false-positive rate from Table 1, stays under a fixed budget. This is a standard multiple-comparisons correction: the same statistical principle behind the Bonferroni correction, applied here per-visitor rather than globally, so that a visitor whose phone model is rare in the comparison population is judged by a permissive threshold appropriate to their low real risk, while a visitor whose phone model is common is automatically judged by a stricter threshold appropriate to their higher real risk, rather than forcing one global threshold to be simultaneously correct for both cases, which Section 7 shows is not possible.

This is, in our judgment, the only remediation attempted in this project that is honestly shaped like a real fix rather than a relocation of the underlying problem, and it requires no new signal collection, no schema change, and no new service: it is a lookup table applied at the single point in the existing code where a comparison score is checked against a threshold. We record it as a genuine, real, structurally correct partial mitigation.

It is, however, not a solution to the population-scale problem, for two separate reasons, one a hard mathematical limit and one a data limit.

The mathematical limit: for large k (a popular phone model in a large server), Table 1 shows there is no threshold value that simultaneously achieves a low compounded false-positive rate and a usable true-positive rate; the entire achievable menu is the same one presented in Section 7 (86 giving 92.9 percent true-positive rate at a per-pair rate that still does not survive compounding past roughly $k \approx 50$, up through 98 giving 9.8 percent true-positive rate). A per-visitor adaptive threshold routes each visitor to the correct point on this existing curve; it does not create a better curve, and for popular models at real server sizes, every point on the existing curve is either unsafe or useless. The honest output of this remediation is therefore that the general matcher becomes reliable for the substantial fraction of members whose device model is uncommon within their server, and correctly, transparently degrades, rather than silently floods the review queue, for whichever two or three device models are most popular in that server, which for most Western, Discord-using populations is very likely to include at least one current-generation iPhone.

The data limit: calibrating $k \rightarrow$ threshold precisely above 86 is blocked by the same small-sample

problem documented throughout this paper and formalized in Section 10. Table 1 shows identical 0.00 percent same-model false-positive rate from 86 through 98, which is a measurement-resolution artifact of only 26 real same-model pairs existing at all, not evidence that these threshold values are actually equivalent in true risk. The coarse version of this remediation (route rare models to 83 to 86, popular models to something stricter) is implementable today; a precisely calibrated version is not, without substantially more real same-model data than this project was able to collect.

9.5 Additional Device Farm testing budget

Finally, we evaluated whether the roughly 500 remaining AWS Device Farm device-minutes available to this project at the time of its cancellation, potentially directed at devices in a different Device Farm region or availability configuration, would meaningfully close the data-volume gap underlying Sections 9.3 and 9.4's data-limit findings.

We first note a factual correction to the premise: AWS Device Farm, unlike general-purpose compute services, operates from a single physical device laboratory rather than multiple independent, geographically distributed facilities with different underlying device inventories; requesting a different region or availability-zone configuration does not route a test session to different physical hardware or a different real-world network path, so this specific remediation does not address the network-diversity problem raised in Section 9.3 at all, independent of the budget question.

On the pure data-volume question: without a precise, recorded per-device-run duration figure from this project's own campaigns to extrapolate from, we can only reason at the level of an order-of-magnitude estimate. Treating each device run as consuming on the order of a few minutes including device allocation and teardown overhead, a budget of 500 device-minutes corresponds to roughly 80 to 100 device-runs at most, and realistically fewer once accounting for the possibility that Device Farm's inventory does not stock unlimited additional distinct physical units of any single popular model. Directed entirely at one or two specific models already identified as the dominant source of remaining risk (Section 8.4), rather than spread thin across many models as the earlier four campaigns were, this budget could plausibly move a single model's real same-model sample size from roughly four physical units (six pairwise comparisons) to perhaps fifteen to twenty units (on the order of one hundred to two hundred pairwise comparisons), a genuine and worthwhile tightening of the statistical confidence available for that one model. It falls, however, roughly two orders of magnitude short of the sample size Section 10 shows is actually required to certify a false-positive rate low enough to survive compounding at real population scale, and it can meaningfully improve calibration confidence for at most one or two models at this budget size, not resolve the underlying data-volume problem for the signal set as a whole.

10 Statistical Power: Why the Data Volume Available Cannot Resolve This

This section formalizes an argument that recurs, in less formal terms, throughout Section 9: this project's real same-model sample size is not merely smaller than would be ideal, it is smaller by roughly two orders of magnitude than what would be required to certify any threshold or signal safe for full-population deployment, and this gap was not closeable within the project's realistic remaining resources.

Consider a threshold or signal whose true same-model false-positive rate is some unknown value p, measured from n real same-model pairs with zero observed false positives (exactly the situation reported for every threshold from 86 through 98 in Table 1, with $n = 26$). The rule of three, a

standard approximation for the upper bound of a binomial proportion confidence interval when zero events are observed, gives a 95 percent confidence upper bound of approximately $3/n$ on the true rate. At $n = 26$, this upper bound is approximately 11.5 percent, meaning the observed "0.00 percent" at threshold 86 is fully consistent with a true same-model collision rate as high as roughly one in nine, not merely with a true rate near zero.

Section 8.1's compounding formula tells us what upper bound would actually be required. To keep the compounded probability of at least one false flag under a modest 5 percent budget at $k = 500$ same-model peers (a realistic figure for a single popular phone model in a server of several thousand members, well within normal Discord community sizes), the required per-pair rate solves $(1 - p)^{500} \geq 0.95$, giving $p \leq 0.0001$ (0.01 percent). Achieving a 95 percent confidence upper bound at that level, again via the rule of three, requires approximately $n = 3/0.0001 = 30{,}000$ real same-model pairs with zero observed false positives. Even relaxing the target considerably, to a 1 percent compounded budget at a more modest $k = 100$, still requires an upper bound near $p \leq 0.0001$ and a correspondingly large n; relaxing further to accept a merely inconvenient rather than catastrophic 20 to 30 percent compounded false-positive rate at $k = 100$ still requires p on the order of 0.2 to 0.3 percent, and correspondingly n on the order of 1,000 to 1,500 real same-model pairs, a figure this project's entire four-wave campaign history, across every model combined, fell roughly forty to sixty times short of.

We want to state this conclusion plainly, since it is the load-bearing statistical fact behind every remediation failure in Section 9: this project cannot currently determine, and could not have determined with any additional AWS Device Farm budget realistically available to it, whether any threshold, any gating signal, or any combination of signals is actually safe for full-population 1:N deployment. This is not a claim that the system is provably unsafe in the sense of a demonstrated high true error rate; it is a claim that the system's safety at the scale the product requires is, and given this project's resources would have remained, fundamentally unverifiable. We regard shipping a system whose core safety property cannot be verified, rather than one that has been verified and found wanting, as an unacceptable basis for a moderation tool whose false positives fall on real, innocent Discord users.

11 Product and Architectural Constraints

Several decisions made earlier in this project's design, independent of the measurements in Sections 4 through 10, further narrowed the space of remediations available once the population-scale problem became clear, and we record them here for completeness.

Cross-guild correlation, an early design option under which a device flagged in one Discord server could inform or pre-flag the same device in a different, unrelated server, was explicitly and permanently rejected midway through this project, on the stated reasoning that this shape of feature is mass-surveillance-like regardless of its technical merits. This foreclosed one obvious route to more comparison data per decision (pooling submissions across many guilds to build a larger, more statistically powered same-model reference set), since the project's matching logic is architecturally scoped to a single guild's own stored submissions from the start, not as a configurable option to be revisited later.

The product's default action on a match is routing to a human moderator review queue rather than automated banning, and this default meaningfully changes the practical consequence of the false-positive rates documented in this paper, from "an innocent user is banned" to "an innocent user's join generates a review-queue entry a moderator must clear." This is a real mitigating factor, and it reframes the operationally relevant question from Section 8's raw false-positive rate to a

review-queue-volume question: given a server's real join rate and the compounded per-visitor flag probability from Section 8, how many false review-queue entries does a moderator team actually have to process per day, and is that volume tolerable. We consider this the correct question for a future attempt at this problem to ask before any further engineering investment, and note that it does not require solving Section 8's underlying false-positive rate, only bounding it well enough that the resulting queue volume is survivable, a meaningfully lower bar than eliminating the false positives outright. We did not reach the point of answering this question with real numbers before this project's cancellation, since it requires real production join-rate data this project, being pre-launch, does not have.

We note, however, that an automated-ban configuration, which several communities are expected to want given that a review queue requires ongoing moderator attention a smaller community may not have, is specified to force the strictest available sensitivity setting. Given Section 7's finding that the strictest thresholds this project measured either still fail to survive population-scale compounding (86) or destroy true-positive rate outright (98), we do not believe any configuration of this product, as specified, was safe to expose an automated-ban option for at real server scale, even setting the review-queue mitigation aside entirely for that specific configuration.

12 Conclusion

We set out to build a system that could detect Discord ban evasion by comparing a rejoining account's browser fingerprint against every existing member of a server. Over four real-device measurement campaigns, we built a working, calibrated, real-data-validated pairwise comparison function; integrated a genuinely effective third-party signal; identified and fixed a specific, well-evidenced failure mode with an absolute score cap; and thoroughly investigated, measured, and correctly rejected several plausible-looking novel signals and reweighting strategies that did not survive real-device measurement.

None of this is sufficient. The product's core requirement, comparison against a server's full membership rather than a narrow or time-windowed candidate set, changes the relevant unit of analysis from a single pairwise comparison, which this project measured and tuned successfully, to the maximum of hundreds or thousands of such comparisons, which compounds even a small per-pair error rate toward near-certain false flags once population size and realistic device-model market concentration are accounted for. Every remediation we attempted, a stricter threshold, an auxiliary exact-match gate, network-address corroboration, and a population-size-adjusted threshold, either fails outright on its own terms or, in the one case that is structurally sound (Section 9.4), only correctly localizes the failure to popular device models rather than resolving it, and even that partial mitigation cannot be precisely calibrated with the real data volume available. Underlying nearly every one of these failures is a single statistical fact, developed formally in Section 10: certifying any threshold or signal safe at the scale this product requires needs on the order of one to several tens of thousands of real same-model device pairs, and this project's entire real-device measurement effort, across every campaign it ran, produced on the order of two to three hundred pairs at most and closer to two to three dozen for any individual device model.

We do not believe this gap was closeable within Atria's available resources, financial or in engineering time, and we do not believe it is closeable by further tuning of the signal set this project built. Closing it would require either a genuinely new, iOS-available entropy source that Safari's anti-fingerprinting hardening does not suppress across the multiple signal families documented in Section 5, which this project searched for directly and did not find, or a real-device data collection effort one to two orders of magnitude larger than what this project, or in our judgment most

projects of comparable resources, could realistically fund. Absent either of those, we conclude that population-scale 1:N browser fingerprint matching against a full Discord server membership, as specified for this product, is not achievable, and the project has been discontinued on that basis.

We publish this account so that the specific reasoning, and the specific real-device measurements underlying it, are not lost, and so that a future attempt at a similar problem, whether within this project's successor efforts or elsewhere, can start from where this investigation ended rather than repeating it.

13 Postscript (2026-08-11): A Follow-On Implementation Attempt

The conclusion above was reached and the project's cancellation decided before this section was written. Several days later, a project contributor, having read this paper, submitted a working implementation that responds directly to Section 9's discussion, and in particular to Section 9.4's population-size-adjusted threshold, the one remediation that section already characterized as "honestly shaped like a real fix rather than a relocation of the underlying problem." We record the attempt, and our own independent measurement of it against the same real device corpus described in Section 3, here. We did not write the code described in this section; we did review it line by line before running anything, build it, run this project's existing automated test suite against it, and then design and run every measurement reported below ourselves, against real data, rather than taking the submission's own reported numbers on faith.

13.1 What changed

The submission makes two changes, one architectural and one statistical, and states plainly that neither is a new signal: no new collector, no new probe, no schema change.

First, it narrows the comparison gallery. Section 9's discussion of every remediation in this paper preserved, as a stated constraint, "the product's core requirement, comparison against a server's full membership, not a narrowed or time-windowed subset." The submission argues, and we find the argument sound on reflection, that this constraint conflates two different narrowings. Removing `FastCheck`'s thirty-minute window is necessary, since an evader can simply wait a day before rejoining. Comparing against a server's entire membership is a different requirement, and not one the product's own stated purpose demands: a match against an ordinary member in good standing is not evidence of ban evasion at all, is not actionable under any configured moderation action described in Section 2.1, and is a more invasive claim to make about two uninvolved people than the product's own consent and disclosure framing anticipated. The submission's general matcher therefore compares a new join against every user ever banned in that guild who has a fingerprint on file, with no time window, rather than against the guild's full membership. This is strictly more coverage than `FastCheck`'s thirty minutes, and, for a guild whose membership is in the thousands, is typically one to two orders of magnitude fewer candidates than full-membership comparison, though not a small constant: a large or long-lived guild can accumulate a ban history numbering in the hundreds or more, so this narrows the typical case rather than capping the worst case, and it does not change anything Sections 5 through 10 of this paper establish about what happens to accuracy as that number grows.

Second, it replaces the fixed global threshold with a decision rule, implemented in a new package, that we confirmed by reading is a working, tested implementation of the Bonferroni-style, population-size-adjusted approach Section 9.4 already endorsed in principle. The rule requires, in order: that a comparison have enough overlapping signal to be judged at all (an evidence floor, since Section 4 and Section 6.2's own findings show a thin comparison can produce a misleadingly

high percentage from very little real agreement); that some of that agreement come from signal families capable of distinguishing two physical units of one model rather than model-class fields alone (a machine-readable form of exactly the distinction Section 5 draws only in prose); a score above a bar calibrated to this specific guild and this visitor's device class; a clear margin over the runner-up candidate; and that no second candidate independently clear the same bar, since Section 8.2's own simulation already measured that a mean of roughly two candidates, rising toward eight among visitors who ultimately triggered a false flag, clear the threshold together well before population size reaches the thousands this paper's central result concerns, and treated that as evidence of a generic device rather than as several pieces of corroborating evidence.

The calibration itself is the one piece of this submission we regard as a genuine methodological contribution independent of whether it resolves this paper's central problem, and it responds directly to Section 10. Section 10 shows that certifying a same-model pairwise false-positive rate low enough to survive compounding at real population scale requires on the order of thirty thousand real same-model pairs with zero observed collisions, a sample size roughly two orders of magnitude beyond anything this project's entire measurement campaign could produce. The submission sidesteps this specific objection, without resolving the underlying entropy problem, by changing what is being estimated. Rather than a per-pair collision rate estimated from labelled same-model pairs, which are scarce, it estimates a false-positive identification rate in the sense used in the biometric identification literature: the fraction of searches by visitors who are not represented in the gallery that nonetheless return some candidate above threshold. Every ordinary member of a guild, scored against that guild's own ban gallery, is exactly such a search, and unlike a labelled same-model pair, it requires no manual labelling, is generated continuously by ordinary production traffic, and a guild of even modest size produces the several hundred such searches Section 10 shows are needed to certify a false-positive rate on the order of one percent. This is a real answer to Section 10's specific data-volume objection to *calibration*. It is not an answer to Section 5's entropy problem, and the measurements below show why those are different questions.

The sample-size arithmetic behind that claim is the same rule of three Section 10 already uses, applied to the new quantity rather than a new method. Certifying, with zero observed events, that a false-positive identification rate is at or below some budget α requires approximately $n = 3/\alpha$ non-mated searches at 95 percent confidence. At $\alpha = 0.01$, the budget `identify.DefaultParams` actually ships with, that is $n \approx 300$; at the tighter $\alpha = 0.001$ Section 10 also discusses, $n \approx 3{,}000$. Section 10 shows 30,000 labelled same-model pairs are needed at the stricter budget this paper argues population scale actually requires (α on the order of 0.0001 against $k = 500$ same-model peers); the corresponding non-mated-search figure at that same α is $n \approx 30{,}000$ as well, since the rule of three scales identically regardless of what kind of event is being bounded. The difference is not in how much data the budget needs, it is in how expensive that data is to obtain: 30,000 labelled same-model pairs is a real-device campaign roughly sixty times larger than everything this project ran across four waves combined, while 30,000 non-mated searches is a few weeks of ordinary joins for a guild of a few hundred active members, generated automatically with no labelling step. That distinction, not a smaller sample-size requirement, is what the reframing actually buys.

13.2 How we measured it

We checked out the submission into an isolated git worktree, read the full diff against this project's main branch before running anything, and confirmed it introduced no destructive operations, no unreviewed network calls, and no secret handling. We then built it and ran this project's existing automated test suites, all of which passed unmodified. We did not stop there: this paper's own methodological standard, stated in Section 3, is that every quantitative claim rests on real phys-

ical device measurements rather than the submission's own reported figures, and we applied that standard here rather than exempting a late addition to the project from it.

We re-ran `cmd/analyze`, this project's original pairwise measurement tool, against the same two real-device corpora described in Section 3 (999000111, waves 1 through 3, and 999000112, wave 4), on the submission's own code, to confirm the underlying pairwise scorer's behavior had not silently regressed. We then ran the submission's own 1:N measurement tool, `cmd/simulaten`, in both of its two modes: against the real corpus directly (bounded to the roughly one hundred to two hundred real devices per guild that this project's data actually supports, exactly the caveat Section 3 raises about this dataset's size), and, separately, in Monte Carlo mode against a synthetic population generator that the tool calibrates against the real wave 4 operating point before reporting any 1:N result, extended out to a population of ten thousand candidates, an order of magnitude beyond what this paper's own Section 8 simulation could reach given the real device pool available at the time it was written. Finally, to understand a result described below that was not self-evident from the submission's own summary output, we wrote a small diagnostic program, built against the submission's own decision-rule package rather than a reimplementation of it, that records which specific gate of the five-part rule stopped each genuine repeat visit in our real corpus from being flagged.

13.3 Results: the pairwise scorer is unchanged

`cmd/analyze` against wave 4 reproduced Table 1 exactly: 98.2 percent true-positive rate, 0.29 percent cross-population false-positive rate, and 7.69 percent same-model, different-unit false-positive rate at threshold 83, confirming that the submission's changes to the pairwise scorer itself (a corrected treatment of timed-out and error-carrying probe results, discussed further below, and a strengthened `isIOS` detection that also reads the user agent rather than only `navigator.platform`) did not alter wave 4's own already-measured operating point. Run for the first time against the wave 1 through 3 corpus, which this paper's Table 1 never covered, the same tool reports a materially worse pairwise operating point: 91.7 percent true-positive rate, 1.76 percent cross-population false-positive rate, and 22.86 percent same-model, different-unit false-positive rate, roughly three times wave 4's same-model rate. We traced this directly to data completeness rather than to any change in the scoring logic: a direct query of the underlying submissions confirmed FingerprintJS's visitor ID, the single signal Section 4.1 identifies as this project's widest-margin discriminator, is present on zero of 434 wave 1 through 3 submissions and 218 of 224 wave 4 submissions, since wave 1 through 3 predates that integration entirely. This is not a new finding about the scorer; it is a clean confirmation, on a corpus this paper's own Table 1 never exercised, of exactly the mechanism Section 4.1 already reports.

13.4 Results: real 1:N search behavior on both corpora

Unlike Section 8's simulation, which samples a synthetic database against a real-scored probe, `cmd/simulaten`'s corpus mode performs an actual search: each real device's first visit is scored against every other real device's first visit as a single gallery, and each device's later visits are scored the same way as genuine repeat-visit probes, directly measuring, rather than extrapolating, the false-positive and true-positive rate of a real 1:N query at the population size this project's real data supports.

Corpus	Rule	N	FPR	TPR	Abstain
Wave 4 (999000112)	Current, threshold 83	104	16.19% (17/105)	98.3% (117/119)	—

Corpus	Rule	N	FPR	TPR	Abstain
Wave 4 (999000112)	Calibrated	104	1.90% (2/105)	68.9% (82/119)	10.5%
Waves 1-3 (999000111)	Current, threshold 83	121	62.30% (76/122)	89.1% (278/312)	—
Waves 1-3 (999000111)	Calibrated	121	1.64% (2/122)	2.6% (8/312)	47.5%

Table 4: Real 1:N search, both real corpora, current fixed threshold versus the submission's calibrated rule. N is the gallery size (every other device in the corpus). FPR is the fraction of innocent first visits for which the search returned at least one candidate at or above the decision bar; TPR is the fraction of genuine repeat visits for which the search correctly identified the true prior visit.

Two things stand out. First, the current fixed-threshold rule's false-positive rate on wave 1 through 3, 62.30 percent, is dramatically worse than wave 4's 16.19 percent, at a gallery size two orders of magnitude below anything Section 8 argues a real deployment would actually face, and is a direct, real-search confirmation, rather than a simulated one, of Section 8's compounding argument: the wave 1 through 3 corpus's higher per-pair collision rate, driven by the absence of FingerprintJS, compounds into a materially worse search-level failure at the same N. Second, the calibrated rule holds false-positive rate to under 2 percent on both real corpora, which is the specific property it was designed to guarantee, but the true-positive rate it pays for that guarantee is not uniform: 68.9 percent on wave 4, 2.6 percent on wave 1 through 3.

To understand that gap precisely rather than attribute it loosely to "worse data," we broke down, gate by gate, why each of the 312 genuine repeat-visit probes on wave 1 through 3 and 119 on wave 4 were or were not flagged.

Outcome	Wave 1-3, no FingerprintJS	Wave 4, has FingerprintJS
Flagged correctly	8 (2.6%)	82 (68.9%)
Abstained: multiple candidates simultaneously at threshold	182 (58.3%)	19 (16.0%)
Below the calibrated bar	88 (28.2%)	14 (11.8%)
Below the raw 83% floor	34 (10.9%)	2 (1.7%)
Other (margin, wrong candidate)	0 (0.0%)	4 (3.4%)

Table 5: Gate-by-gate disposition of every genuine repeat-visit probe, both real corpora, submission's decision rule with default parameters.

The gap is dominated by one gate. On wave 1 through 3, 58.3 percent of genuine repeat visits are abstained specifically because three or more gallery members independently score at or above the raw threshold against the visitor at once, the single-trigger guard described above, correctly refusing to name one of several simultaneously plausible candidates as the match. This connects directly to Section 4.1 and Section 5: without FingerprintJS, the remaining canvas, audio, and GPU-bench signals are not, on this corpus, sufficient to keep same-model peers separated from the true match, so a genuine visitor's real prior device and several impostor same-model peers cross the threshold together, and the rule that Section 8.2 already showed protects against exactly this correctly declines to guess among them. The remaining gap, a higher share of visits falling below the calibrated bar on wave 1 through 3 (28.2 percent against 11.8 percent), is a second-order consequence of the same crowding: the calibration itself is built from this guild's own member population, so a population whose same-model scores already cluster near the true-match score

raises the bar the calibration computes. We regard this as one mechanism observed at two points in the pipeline, not two independent findings, and as real-data confirmation, not simulation, of the correlated entropy suppression Section 5 already establishes: the same missing signal that inflates the naive false-positive rate also drives the calibrated rule's true-positive collapse, because both are downstream of the same loss of separation between real devices.

13.5 Results: population of ten thousand

Section 8's own simulation was bounded to a population of five hundred by the real device pool available when it was written, and Section 8.1's compounding argument for populations in the thousands was, at that point, analytic rather than measured against the submission's own decision code. We extended `cmd/simulaten`'s Monte Carlo mode, whose synthetic population generator is calibrated against the real wave 4 pairwise operating point before any 1:N figure is reported, to a population of ten thousand, run against two specific device classes chosen to bracket this paper's own finding: a collapsed-entropy iPhone that shares its screen panel with two other models (Section 5's worst case, for which the iOS screen-geometry cap in Section 4.2 has nothing to key on), and a Galaxy S25, representative of the well-separated Android tier Section 8.3 already reports at 0.0 percent false-positive rate. Each population of ten thousand is expensive to score in full (every trial scores the probe against all ten thousand gallery members), so each cell below is 80 innocent-visitor trials and 50 genuine-repeat-visit trials, smaller than Section 8's own 400/200 default, and we report the 95 percent Wilson interval on each point estimate rather than presenting bare percentages, consistent with Section 8's own practice of reporting bootstrap intervals rather than single numbers.

Visitor device	Rule	FPR	TPR	Abstain	95% CI (FPR, n=80)
Collapsed-entropy iPhone	Current, threshold 83	95.00%	100.0%	—	[87.8%, 98.0%]
Collapsed-entropy iPhone	Calibrated	0.00%	0.0%	95.0%	[0.0%, 4.6%]
Galaxy S25	Current, threshold 83	0.00%	98.0%	—	[0.0%, 4.6%]
Galaxy S25	Calibrated	0.00%	98.0%	0.0%	[0.0%, 4.6%]

Table 6: Measured (not extrapolated) 1:N behavior at a synthetic population of ten thousand, submission's own decision code, calibrated against the real wave 4 operating point. FPR confidence intervals are Wilson score intervals on 80 trials; the collapsed-entropy iPhone's calibrated TPR of 0.0% carries a 95% Wilson interval of [0.0%, 7.1%] on its 50 genuine-visit trials, and the Galaxy S25's calibrated TPR of 98.0% carries [89.5%, 99.6%] on the same 50 trials.

At this scale the two device classes diverge completely, and the divergence is the paper's central finding restated in the submission's own numbers. Where real separating entropy exists, the Galaxy S25 result, the current fixed threshold was already safe at this population size (its own per-pair false-positive rate is low enough to survive compounding even at ten thousand), and the calibrated rule is a measured no-op: identical false-positive rate, identical true-positive rate, its own computed bar unchanged from the uncalibrated 83. Where it does not, the collapsed-entropy iPhone result, the current fixed threshold is, as Section 8.1's compounding argument predicted, essentially guaranteed to false-flag an innocent visitor (95.00 percent measured, 95 percent CI [87.8, 98.0], against a mean of 167 to 294 gallery members independently crossing threshold per query, consistent with the naive independence estimate this paper's Section 8.1 derives from the real measured per-pair rate), and

the calibrated rule's answer is not a better threshold; it is complete abstention. False-positive rate falls to 0.00 percent, and true-positive rate falls with it to 0.0 percent, an interval tight enough (upper bound 7.1 percent at 50 trials) to say with real confidence that this is nowhere close to the 98 to 100 percent true-positive rate this same device class reaches under the current rule, not a trial-count artifact. The system does not misidentify this population at scale. It also, at scale, cannot identify it at all.

This same run answers a question Section 11 raises and explicitly leaves open: "given a server's real join rate and the compounded per-visitor flag probability from Section 8, how many false review-queue entries does a moderator team actually have to process per day, and is that volume tolerable... we did not reach the point of answering this question with real numbers before this project's cancellation." `cmd/simulaten` computes exactly this, at an illustrative 300 joins per day and a 1 percent evader base rate, and we report it here since Section 11 asked for it and the submission's own tooling already produces it.

Visitor device	Rule	False flags/day	Precision
Collapsed-entropy iPhone	Current, threshold 83	282.1	1.1%
Collapsed-entropy iPhone	Calibrated	0.00	—
Galaxy S25	Current, threshold 83	0.0	100.0%
Galaxy S25	Calibrated	0.00	100.0%

Table 7: Review-queue economics at N=10,000, 300 joins/day, 1% evader base rate, directly answering Section 11's own open question with the submission's tooling. Precision is the share of review-queue entries that are genuine, the number a moderator team actually experiences.

Read plainly: on the device class this whole section is about, the current rule would hand a moderator team 282 false review-queue entries a day, 98.9 percent of everything the queue contains, for a signal that catches genuine evasion at a rate this same table's TPR column already shows is real but swamped. The calibrated rule reduces that to zero entries a day, which sounds like an unqualified win until it is read alongside the same cell's 0.0 percent true-positive rate above: zero false flags and zero true ones is not a working detector, it is a detector that has stopped operating for this population. Section 11's question, whether the false-flag volume is "tolerable," turns out to have a sharper answer than the section anticipated: at this device class and this scale, the current rule's volume is not tolerable, and the calibrated rule's fix is not to make the volume tolerable but to make it zero by making the true-positive volume zero alongside it. Precision is not a meaningful number to report for that cell, and we report it as such rather than as a defined percentage.

13.6 Interpretation

We read this submission, on the evidence above, as a genuine and well-engineered piece of work that we would not have built ourselves given this project's remaining resources, and as confirming rather than refuting this paper's central conclusion.

It succeeds completely on the narrow property it targets. The false-positive rate is now bounded by construction (the calibration's own budget parameter is an explicit upper bound on the per-query false-flag rate, not an estimate of it) and, on every real and simulated measurement we ran, that bound holds: under two percent on both real corpora at real scale, 0.00 percent on the hardest synthetic device class at ten thousand candidates. Sections 8 through 10 of this paper document a system whose false-positive rate was unbounded in the specific sense that it grew

toward certainty as population size grew and could not be safely raised without destroying recall, and whose safety, per Section 10, could not even be verified with the real data available. This submission converts that into a system whose false-positive rate is bounded, verifiable from data a real deployment generates for free, and, critically, honestly reported: an unanswerable query returns an explicit ABSTAIN rather than a confident-looking percentage a moderator has no way to know is meaningless. We regard the reframing of Section 10's calibration problem as a false-positive identification rate estimated from ordinary unlabelled traffic, rather than a per-pair collision rate estimated from scarce labelled same-model pairs, as a real methodological contribution independent of everything else in this section, and worth recording for its own sake.

It does not, however, close the gap this paper's Sections 5 through 10 describe, and the ten-thousand-candidate measurement above shows exactly why not, in the submission's own numbers rather than in argument. Bounding a false-positive rate and recovering lost discriminating entropy are different problems, and this submission solves the first without touching the second. On the device population Section 5 identifies as having the least separating entropy left, real correlated entropy suppression across canvas, audio, GPU, and third-party-hash signals simultaneously under iOS Safari, the calibrated rule's honest response at realistic scale is to stop answering, not to start answering correctly. A moderator reading an ABSTAIN is getting a true statement, which is a real improvement over a false CONFIRMED, but the product's stated purpose is detecting ban evasion, and for a device class this paper's own Section 8.4 already identified as the dominant source of remaining risk (47.3 percent false-positive rate under uniform sampling at N=100, entirely an iOS phenomenon), the measured outcome at population scale is that the system can no longer perform that function at all. The wave 1 through 3 result in Section 13.4 is the same finding measured on real historical data rather than simulation: removing one signal (FingerprintJS) from an otherwise identical real device population does not move the calibrated rule's false-positive rate, which stays near its bound, but it collapses its true-positive rate from 68.9 to 2.6 percent, because the calibration cannot manufacture separation that the underlying comparison no longer has to give it.

We also want to flag, for completeness rather than to relitigate Section 9, that this submission's narrowing of the comparison gallery from full membership to ban history is itself a real change in the problem this paper's Sections 8 through 10 were written against, one we had not evaluated in the original report. We believe the argument for it is sound on its own terms, a match against a member in good standing is not actionable and is a disproportionate claim to make, and it substantially reduces the typical-case population size a real deployment's general matcher would search against. It does not, however, cap the worst-case population size at a small constant (a long-lived, high-churn guild can accumulate a ban history in the hundreds or low thousands), and it changes nothing about what this paper's Sections 5 and 10 establish about how per-pair error compounds and how much real same-model data would be needed to certify any decision rule safe, calibrated or not, once that population size is reached. We regard it as a legitimate, worthwhile reduction in the scope of the problem, and a separate contribution from the calibrated decision rule itself, not as a resolution of Section 5's entropy shortfall.

Our overall reading is that this project's original conclusion stands, and that this submission demonstrates why in a more precise and more directly falsifiable way than this paper's original Section 8 through 10 could, since it is measured against the same real device corpus at a population size an order of magnitude beyond what this paper's own simulation could reach when written. A decision rule can be made honest about what it does not know. It cannot be made to know what the underlying signal no longer contains.

13.7 The broader signal search, for context

This paper states in several places, most directly in Section 12, that closing the entropy gap documented in Section 5 "would require... a genuinely new, iOS-available entropy source that Safari's anti-fingerprinting hardening does not suppress... which this project searched for directly and did not find." We did not, in the body of this paper, document the extent of that search, and record it briefly here for anyone assessing how thorough it was.

Independent of the measurements reported in Sections 4 through 10, this project maintained a running catalog of fingerprinting techniques considered, separate from the roughly one hundred and ten checks actually shipped in production scoring. That catalog covers, across four dated research passes, transport and network-level techniques (JA3/JA4-class TLS fingerprinting, found to be architecturally unreachable through this project's Cloudflare Tunnel deployment at any paid tier short of replacing the tunnel with self-hosted TLS termination, itself a real option we did not pursue given the scope of this paper's finding), installed-software and OS-integration probing, GPU and rendering behavioral side-channels distinct from capability enumeration, and roughly a dozen further categories of browser and platform API surface. Within that catalog, a specific subpass focused narrowly on this paper's Section 5 finding, whether any single-shot, model-invariant, hardware-manufacturing-variance signal exists that could distinguish two physically identical units of one phone model, examined eight candidate mechanisms in named detail (among them DVFS-based CPU silicon-lottery timing, crystal-oscillator timing variance, WebGPU execution-unit timing, sensor factory-calibration residue, and DRAM latency variance), and independently arrived at the same three candidates this paper's Section 6.3 through 6.5 report having built and measured, CPU-timing, clock-oscillator, and WebGPU execution-timing fingerprinting, as the most promising before any of them was tested against real hardware. All three were subsequently built and included in the wave 4 campaign. Two are documented in Section 6 as rejected on measurement, not on the strength of the literature search alone: the search correctly narrowed the field to testable candidates, and real-device measurement, not further reading, is what closed both of them out. The third, WebGPU execution timing, is documented in Section 6.5 as genuinely inconclusive rather than rejected, a real same-model, different-unit gap appears in the data but rests on eleven pairs, far too few by Section 10's own standard to trust, and is the one candidate in this whole catalog we would flag as worth a further, properly-powered real-device measurement before it is written off.

We record this not to claim the search was exhaustive, no search of this kind can claim that, but to state plainly what its scope actually was, so that this paper's repeated claim of a search having been conducted is auditable rather than asserted.

Acknowledgments

The real-device measurement campaigns described in this paper were conducted using AWS Device Farm. FingerprintJS and ThumbmarkJS are open-source libraries used here for comparison and integration purposes under their respective open-source licenses.

References

[1] Peter Eckersley. *How Unique Is Your Web Browser?* In Privacy Enhancing Technologies Symposium (PETS), 2010.

[2] Carlo E. Bonferroni. *Teoria statistica delle classi e calcolo delle probabilità.* Pubblicazioni del R. Istituto Superiore di Scienze Economiche e Commerciali di Firenze, 1936.

[3] Ivan P. Fellegi and Alan B. Sunter. *A Theory for Record Linkage.* Journal of the American Statistical Association, 64(328):1183–1210, 1969.

[4] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. *Clock Around the Clock: Time-Based Device Fingerprinting.* In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2018.

[5] FingerprintJS, Inc. *FingerprintJS: Open-source browser fingerprinting library.* `https://github.com/fingerprintjs/fingerprintjs`.