Cost-Aware Tracker Blocking in Firefox

Per-request privacy intervention accounting at browser scale

James Han

April 29, 2026

Privacy Team
Full-Stack Engineering Intern

What this talk is about

Firefox serves **more than 200 million users** worldwide, each blocking thousands of tracker requests per month.

Our new-tab privacy widget will surface them as **counts**.

This work provides a model that converts those counts into **costs**.

What we estimate per blocked request:

- **Transfer size** – how many bytes the response would have moved over the wire if Firefox hadn't blocked it
- **Download duration** – how long pulling those bytes would have taken

Each is predicted separately. Transfer size is the deployed target; download duration is a secondary metric.

Today's privacy widget is undercounting

Enhanced Tracking Protection (ETP) is Firefox's tracker blocker. It uses the **Disconnect list** (an open-source list of tracker domains) to decide what to block. Today the new-tab widget tells the user *how many* were blocked.

But not all blocks are equal:

- A `bat.bing.com` tracking pixel: **258 bytes**, ∼5 ms
- A `googletagmanager.com` SDK: **170 KB**, ∼300 ms
- Today the widget treats them the same

Three audiences need the cost answer:

- **Users** – what is ETP actually doing for me
- **Blocklist maintainers** – which rules pull weight
- **Privacy research** – compare ETP to Brave Shields, Safari ITP, uBlock Origin honestly

What a single browsing session looks like today

Same blocked requests. The cost number is what's missing.

A 30-minute browsing session triggers ∼150 tracker blocks. The user sees a count, not a magnitude. Without magnitudes, there's no way to tell whether ETP is doing meaningful work for that user, or whether tightening one rule would matter more than another.

The catch: we blocked the request, so we never see the response

What the browser sees at block time:

- URL, resource type (script/image/beacon), initiator (which page asked), HTTP method

What we want to predict (never observed):

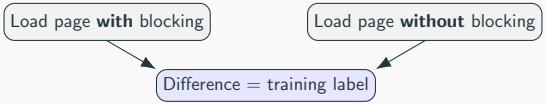
- Transfer size in bytes, download duration in milliseconds

The response is **structurally never received.**

Counterfactual prediction: estimate the cost of something that didn't happen because we prevented it.

Brave already shipped something. Here's why we did it differently.

Brave (2019) ships a bandwidth savings predictor in their browser today. They got there with a shortcut: paired crawls for training labels, page-level prediction.

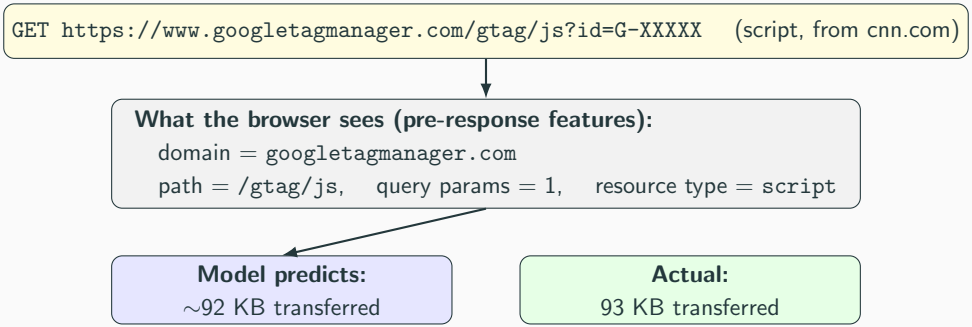


Three problems for our use case:

- **Expensive to refresh.** Every training row needs a paired crawl.
- **Less accurate.** $R^2 = 0.68$, median error 2.7 MB per session.
- **Wrong granularity.** Page totals only. Can't say *which tracker* cost the user, or which Disconnect rule pulls weight.

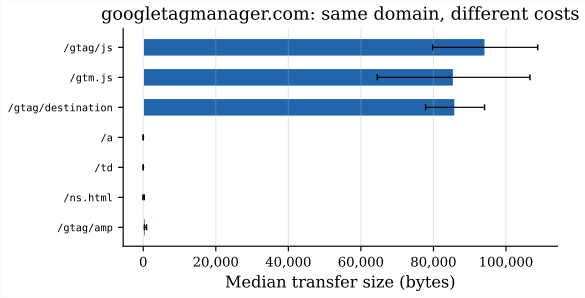
Source: "Accurately Predicting Ad Blocker Savings," Brave blog, October 2019.

One real example: what the model gets and what it predicts



The browser had this information **before** *blocking. The model converts that information into a cost estimate.*

Domain-level estimation is structurally wrong



Same domain. Three orders of magnitude.

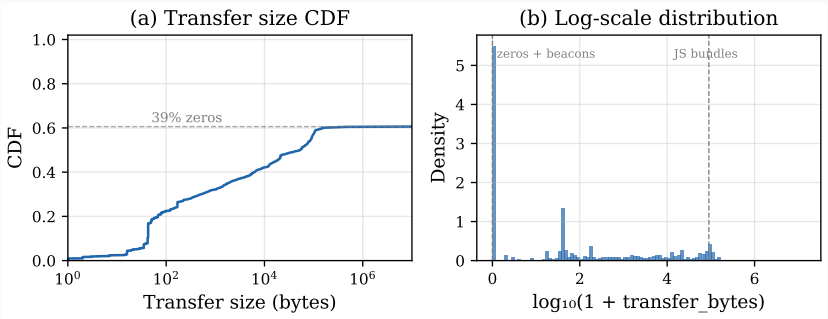
`googletagmanager.com`:

- `/gtag/js` → ∼93 KB
- `/collect` → 0 bytes

Within-domain spread (CV = std-dev/mean): **0.94 at median**, **3.0 at p90**.

Lookup-by-domain cannot tell these apart. **The URL path is the primary determinant of cost, not the domain.**

The shape of the data



- **39.5% exact zeros** (tracking beacons return nothing — they're "fire and forget")
- Heavy right tail out to **8.6 MB** (giant JavaScript SDKs)
- Mean 13.6 KB, median 43 bytes

Spike at zero, long tail of large bundles. *This shape will dictate the loss function later.*

Data

HTTP Archive June 2024 mobile crawl — public dataset of synthetic web crawls run from a fixed test environment. Filtered to third-party tracker requests in 5 Disconnect categories.

3.49M requests across **3,723 tracker domains**

Why training on Chrome data and deploying in Firefox is fine:

Transfer size is *server-determined*. The same URL returns the same payload to any browser. Crawl population (Chrome via HTTP Archive) and deployment population (Firefox real users) see different traffic mixes, but the bytes are the bytes.

No domain-adaptation tricks needed.

Lookup table vs. model: a one-slide intuition

Lookup table

Memorize the answer.

Saw `/gtag/js` → 93 KB before?
Predict 93 KB.

Never saw it?
Fall back to a coarser bucket.

Pro: simple, fast, exact when seen
Con: can't generalize, huge at scale

Model

Learn the pattern.

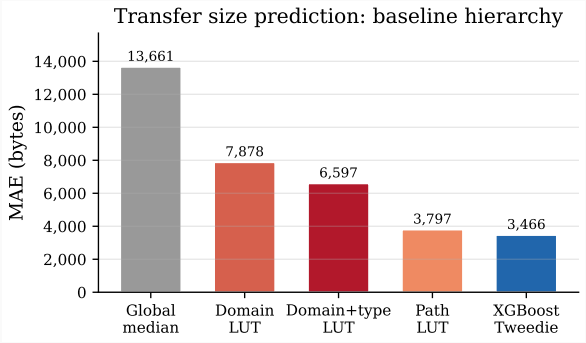
Sees URLs that look like "`js bundle`"
and learns: those are big.

Never saw this exact URL?
Still uses the pattern.

Pro: generalizes, compact
Con: per-request answer is fuzzier

We'll see in a moment that the model wins even where the lookup table has the answer memorized.

First, try the boring thing: lookup tables



Make the lookup key finer, error drops:

- Global median: 13,661
- Domain: 7,878
- Domain + type: 6,597
- Path: 3,797
- **Our model: 3,466**

Path LUT looks competitive on offline eval.

It **cannot deploy.**

MAE (mean absolute error) = average bytes off per request. Lower is better.

Why path lookup is a trap

Memorization scales the wrong way:

- 227K entries from a 1% sample
- Extrapolates to ∼50M entries / **1.1 GB** at full deployment scale
- 75% of test paths have no training match
- URL space evolves continuously: SDK versions, campaign IDs, endpoint structures

Size comparison (this is what kills it):

- Path LUT: ∼**1.1 GB**
- Disconnect blocklist (the thing the LUT would ship next to): ∼**5 MB**
- **Our model:** ∼**500 KB** (∼2 MB with vocab) — **2,000**× **smaller**

What we need: a model that **generalizes from URL structure** rather than memorizing exact paths.

The model

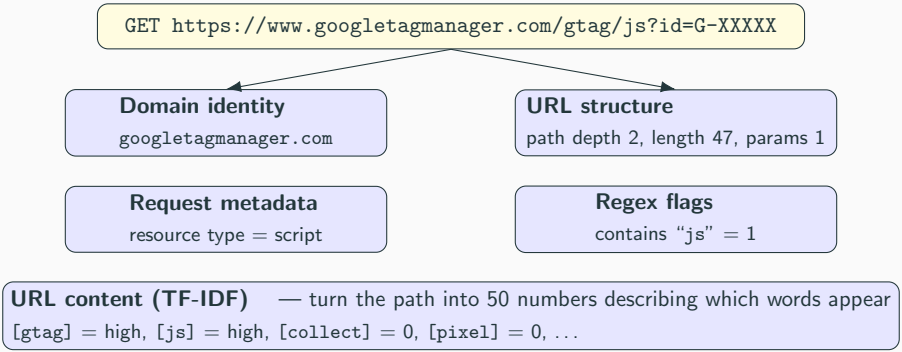
XGBoost regression on **80 features in 5 groups** (SHAP % of total signal):

- **Domain identity** – 2 features, **48%**
- **URL content** – TF-IDF over URL path tokens (50-dim SVD), **32%**
- **Request metadata, URL structure, regex flags** – 28 features, 20%

Deployment-friendly: ∼500 KB ONNX, microsecond inference, ships via Remote Settings (same channel as the blocklist), quarterly retrain.

Architecture is the smallest thing that works.
The interesting modeling choice is the loss function.

What does “feature” mean here? One URL, decomposed.



The model takes 80 numbers like this and produces a single bytes-prediction.

Two pieces of jargon, plain English

TF-IDF

What words distinguish this URL from others?

Tokens like `gtag`, `collect`, `pixel`, `ad`, `sync` appear in tracker URLs with very specific frequencies. TF-IDF gives each URL a score on each token. Tokens common across all URLs (like `com`) are downweighted; tokens that mark a specific kind of resource (like `gtag`) are upweighted.

We then compress these scores down to 50 numbers per URL using SVD (a standard dimensionality-reduction trick).

SHAP

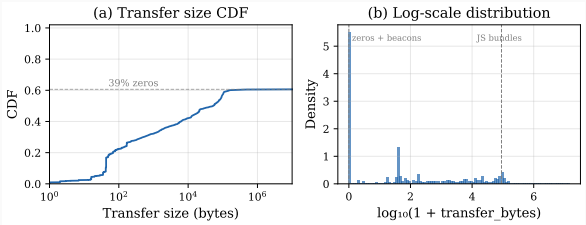
For one prediction, how much did each feature contribute?

Take the model's prediction for one URL. SHAP tells you how much each of the 80 features pushed that prediction up or down, relative to the average.

Sum SHAP values across many predictions and you get a global picture: **domain identity drives 48% of the model's signal, URL content drives 32%**, the rest is split among the other groups.

Different from gain-based importance — gain over-counts high-cardinality features.

Why the standard loss function is wrong here



Squared-error regression assumes the target is roughly bell-shaped.

Tracker responses are not: 40% are zero, the rest heavy-tailed.

Squared error wastes capacity getting beacons exactly right (zero is easy), and underweights the rare large bundles *that dominate total bandwidth*.

We need a loss that natively handles "spike at zero + long tail."

The loss function: borrow from actuaries

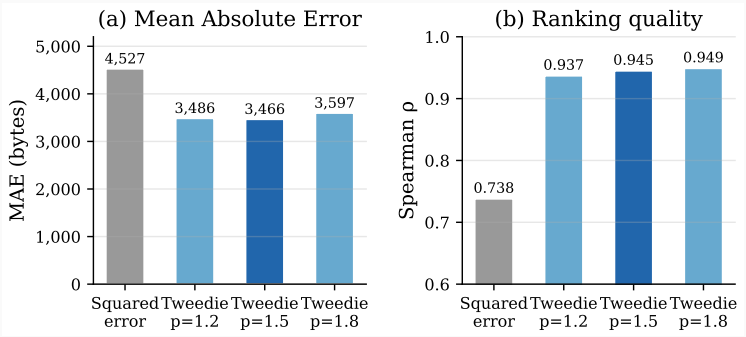
Web response sizes look just like insurance claims:
most filings are $0 (no claim), the rest are heavy-tailed (a few catastrophic claims dominate the total payout).
40% of tracker requests return zero bytes (beacons). The rest are heavy-tailed (a few JS bundles dominate the total bandwidth).

Actuaries solved this in 1987: **Tweedie loss** (compound Poisson-Gamma).
A single loss function that handles the spike-at-zero *and* the long tail naturally.

Power parameter $p = 1.5$. Sensitivity test: $p \in [1.2, 1.8]$ all within $\pm 4\%$ — the choice is robust.

The methodological transfer is the contribution.
Not "we trained an XGBoost." **The right loss for the data shape was sitting in another field.**

The loss function dominates everything else



Same architecture. Same features. Just swap the loss.

Squared error: MAE 4,527, Spearman 0.74. Tweedie ($p = 1.5$): MAE **3,466**, Spearman **0.95**. **23% MAE gap from the loss alone.**

MAE = average bytes off per request. Spearman = how well the model orders cheap-to-expensive (1.0 perfect). Architecture and feature engineering moved MAE by 4–8%; the loss moved it 23%.

Headline result

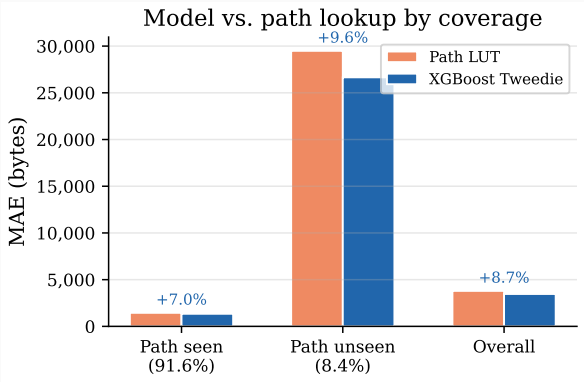
Test set: **523K requests** (held out from training)

Method	MAE (bytes)
Global median (everyone gets the same prediction)	13,661
Domain LUT	7,878
Domain + type LUT (today's status quo for cost estimation)	6,597
Path LUT (offline-only, can't deploy at 1.1 GB)	3,797
XGBoost + Tweedie (ours)	**3,466**

47.5% MAE reduction over the deployable LUT baseline.
Bootstrap confidence intervals (resample test set 1,000×, see how stable the number is): model CI does **not** overlap path LUT CI. The improvement is real, not noise.

But this number understates the actual finding.

The model wins even where the LUT has the answer



Split test set by whether the path was **seen in training**:

Path seen (91.6% of rows):

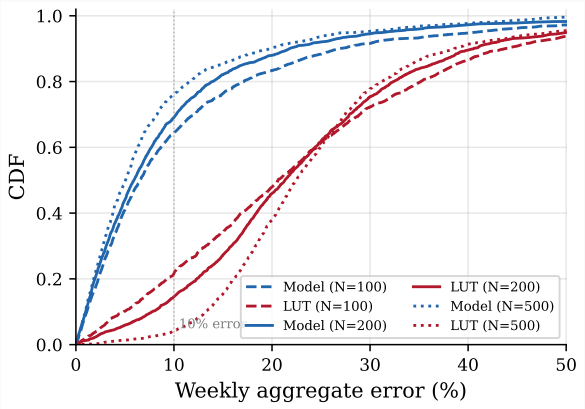
- Path LUT: 1,448 bytes MAE
- Model: **1,346 bytes MAE**

Confidence intervals don't overlap — statistically significant.

The model isn't winning on unseen paths. **It's winning on seen paths by regularizing.**

Even where the lookup has the exact answer memorized, the model still wins.

What the user actually sees: weekly aggregates



Users don't see per-request predictions. They see **weekly totals**.

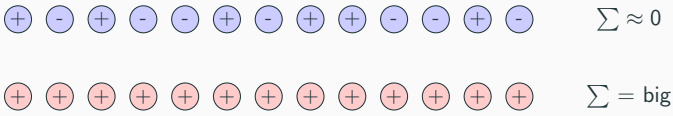
At $N = 200$ blocked requests/week (moderate browsing):

- Domain+type LUT: **22%** median error
- **Our model: 6%**

68.5% of weeks within 10% of true cost.

Per-request noise cancels.
LUT errors are **systematic biases that compound**.

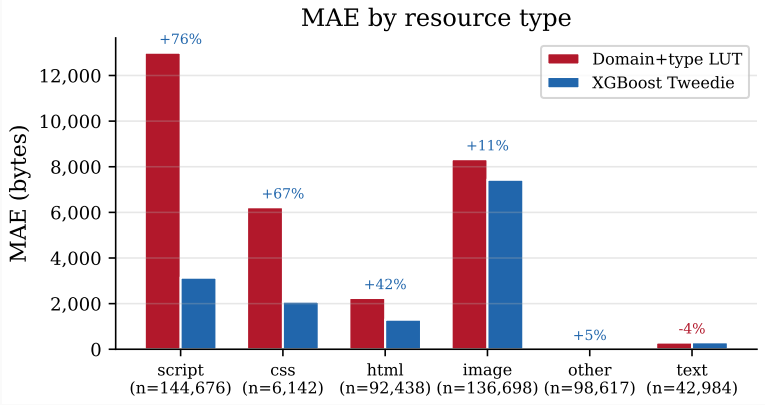
Why per-request noise cancels (and LUT bias doesn't)



Model: per-request errors switch sign — they cancel in a weekly sum.
LUT: errors point the same direction (consistently over- or under-predicts certain URL patterns) — they accumulate.

This is why the model's advantage *widens* with more requests per week — and why the LUT's per-request accuracy is misleading.

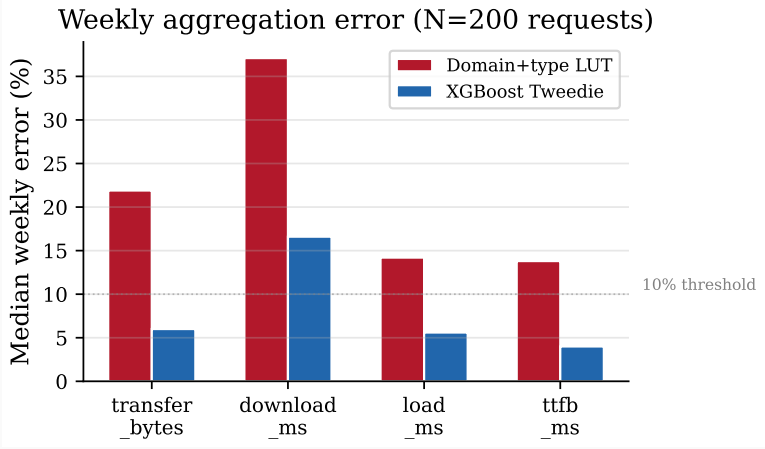
Where the wins concentrate: by resource type



Scripts (+76%) and CSS (+67%) are where the model crushes the LUT — also the *high-cost* resources that dominate weekly bandwidth. Beacons (+5%) and text are unchanged because the LUT was already near-optimal for low-variance types.

The model's gains land exactly where bandwidth actually goes.

Aggregation rescues even noisy targets



Per-request, the model is *worse* than the LUT on TTFB and load time (timing is network-dependent, not URL-dependent). **But in weekly aggregation, the model wins on all four targets.** LUT bias compounds; model noise cancels.

Does it hold up in realistic conditions? Yes.

Domain-correlated browsing (real users return to the same sites — uniform random sampling is unrealistic):

- Domain+type LUT: 36.4% weekly error
- **Our model: 12.9% weekly error**
- Model's advantage *widens* when browsing is concentrated

Temporal generalization (train June 2024, test September 2024 — 3 months later):

- Both methods degrade $\sim$30% as URL space evolves
- **Model retains a 33% advantage** over LUT
- Recommended retraining cadence: **quarterly**

Deployment status

Integration into Firefox's new-tab privacy widget is planned, not yet scheduled.

Proposed integration:

- **Distribution:** Remote Settings (model updates without browser releases)
- **Inference path:** alongside the existing block decision – no impact on the hot path
- **Features:** all available via `nsIChannel` and `nsILoadInfo` at block time – **no new instrumentation**
- **Retraining:** quarterly on fresh HTTP Archive crawls
- **Artifact:** ∼500 KB ONNX (∼2 MB with TF-IDF vocab)

Production browsers today (Firefox, Brave, Safari, Chrome) all surface tracker blocks as **counts**. Brave's prior work was page-level research, not per-request. The per-request cost primitive is what's new here.

Limitations and honest caveats

Deploy transfer size + download duration only. Timing metrics like TTFB and total page load depend heavily on network and device — HTTP Archive crawls from a fixed test environment, so timing predictions don't transfer cleanly to real users.

Cascading effects. Blocking a tag manager prevents downstream injected requests we never see. Predictions are a **lower bound** on true bandwidth saved.

Server-side variation. A/B tests, geo-targeting, SDK versioning all make the same URL return different bytes on different requests. Irreducible per-request noise. Aggregation mitigates.

The model makes no blocking decisions. The Disconnect list decides what to block. Our model only estimates the cost of an already-decided block. A miscalibrated estimate produces a misleading aggregate, never an over-block or under-block.

Adversarial robustness out of scope. Threat model is *honest measurement*, not evasion.

Takeaways

1. **Within-domain transfer size varies by 3 orders of magnitude.** The URL path – not the domain – is the primary cost determinant.
2. **Tweedie loss beats squared error by 23%** on identical features. Loss function selection dominates architecture and feature engineering for this kind of data.
3. **The model beats exact-path lookup on seen paths.** Regularization across URL structure beats memorization.
4. **6% weekly aggregation error.** Per-request noise cancels; LUT biases compound. The widget becomes honest.
5. **Deploys via Remote Settings.** Same channel as the blocklist. No new instrumentation. No hot-path impact. 2,000$\times$ smaller than the path LUT.

Thanks

Tim for the advising work.
Privacy team for the guidance.

Questions?