Predicting the Cost of Blocked Tracker Requests from Pre-Response Features

James Han
Mozilla
Toronto, Ontario, Canada
jhan@mozilla.com

Tim Huang
Mozilla
Berlin, Germany
thuang@mozilla.com

Abstract

Within-domain transfer size varies by three orders of magnitude across tracker domains—yet this variance is predictable from URL structure alone. We exploit this to estimate the performance cost of blocked tracker requests in Firefox's Enhanced Tracking Protection, which intercepts requests before any response arrives, leaving costs structurally unobservable.

We train XGBoost with Tweedie loss on 3,490,824 HTTP Archive requests across 3,723 tracker domains. The model outperforms even exact-path lookup tables on the paths they can match (MAE 1,346 vs. 1,448 bytes), a statistically significant improvement, demonstrating that learned regularization across URL structures beats memorization. Overall, it reduces MAE by 8.7% over the path-level lookup table (the most accurate non-model baseline). Tweedie loss—long standard in actuarial science for zero-inflated claims data—outperforms squared error by 23% on identical features, confirming that loss function selection is the dominant modeling choice: the 23% gap from switching losses dwarfs the 0.6% gap from tuning the Tweedie variance parameter ($p = 1.2$ vs. $p = 1.5$). The model retains a 32.8% advantage on a 3-month temporal holdout and reduces weekly aggregation error from 21.9% to 6.0% over the deployed domain+type lookup table, with 68.5% of weeks falling within 10% of true cost.

Keywords

web privacy, tracker blocking, cost estimation, Tweedie regression, URL embeddings

1 Introduction

Firefox's Enhanced Tracking Protection (ETP) blocks third-party tracker requests identified by the Disconnect list [6], preventing advertising, analytics, and social tracking scripts from loading. Firefox serves hundreds of millions of daily active users, each of whom sees a privacy dashboard reporting how many trackers were blocked—but not the performance cost prevented. Every blocked request is treated equally: a 258-byte tracking pixel from `bat.bing.com` counts the same as a 170 KB JavaScript SDK from `googletagmanager.com`.

Quantifying the cost of blocked requests is a counterfactual estimation problem: the response was never received, so its size and timing are unknown. The browser observes only the request URL, resource type, initiator, and HTTP metadata—features available before the block decision. The response—transfer size, content type, download duration—is unobserved.

A natural baseline is a lookup table (LUT) mapping each (domain, resource type) pair to a historical median response size. However, a single tracker domain serves resources of vastly different sizes:

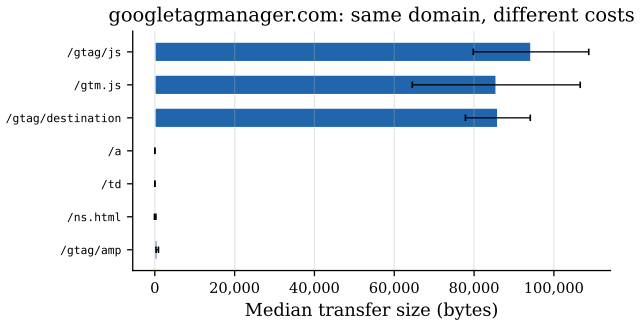

Figure 1: Transfer size varies by orders of magnitude within a single tracker domain (`googletagmanager.com`). A domain-level lookup table conflates 90 KB script bundles with near-zero beacon endpoints.

`googletagmanager.com/gtag/js` returns a 93 KB script bundle while `googletagmanager.com/collect` returns a 0-byte beacon (Figure 1). We measure within-domain transfer size variance across 3,723 tracker domains and find a coefficient of variation of 0.94 at the median domain and 3.0 at the 90th percentile, confirming that domain-level aggregation loses critical information.

A finer-grained path-level lookup table achieves lower per-request error when it has an exact match, but 75% of unique URL paths in the test set are unseen in training (though high-frequency paths recur, so only 8.4% of test rows fall on unseen paths), and the URL path space changes continuously as tracker SDKs evolve. A model that generalizes from URL *structure* rather than memorizing exact paths is required.

This raises the central question of our work: can a learned model, trained on completed tracker requests, produce more accurate and more compact per-request cost estimates than any lookup table at matched feature availability? We investigate this by comparing XGBoost with Tweedie loss against a hierarchy of lookup tables of increasing granularity, using only features Firefox can observe before a response arrives.

We formulate per-request cost prediction as supervised regression with a structural inference-time information gap. The training set consists of completed tracker requests from the HTTP Archive [10], where both pre-response features and post-response labels are observed. At deployment, the model predicts costs for blocked requests where only pre-response features exist. Unlike domain-level cost estimation—which we show does not require machine learning (Section 7.1)—the per-request formulation has a genuine prediction gap: the response is structurally unobservable.

Contributions.

(1) **Measurement finding.** We measure within-domain transfer size variance across 3,723 tracker domains and find that it spans three orders of magnitude (CV 0.94 at median, 3.0 at 90th percentile), with the URL path—not the domain—as the primary determinant. This variance is predictable from pre-response features, enabling per-request cost estimation rather than crude domain-level averages (§3–4).
(2) **Methodological finding.** Tweedie loss—long standard in actuarial science for zero-inflated claims data—transfers to web response data, outperforming squared error by 23% on identical XGBoost features. The loss function gap (23%) dwarfs the Tweedie parameter sensitivity (0.6% between $p = 1.2$ and $p = 1.5$), making loss selection the dominant modeling choice. Data-driven TF-IDF URL embeddings subsume hand-crafted regex features with a 16.5% MAE reduction (§6.2–6.3).
(3) **Generalization finding.** The model outperforms even exact-path lookup tables on paths with training matches (MAE 1,346 vs. 1,448 bytes), a statistically significant improvement (non-overlapping bootstrap CIs), demonstrating that learned regularization across URL structures beats memorization. Per-request prediction noise cancels in weekly aggregation, producing 6.0% median error under uniform sampling and 12.9% under domain-correlated browsing at $N = 200$ (§6.1, §6.6).
(4) **Negative finding.** Domain-level tracker cost estimation does not require machine learning. We diagnose a feature-space disjointness between labeled and unlabeled domain populations that explains a self-training null result and motivates the per-request reformulation (§7.1).

2 Related Work

Third-party web performance. The HTTP Archive [10] crawls millions of pages monthly, storing per-request HAR data and Lighthouse audit results. The Web Almanac [9] reports that the median page loads resources from 21 third-party domains. Pourghassemi et al. [18] developed adPerf, which instruments Chromium's activity trace to attribute CPU costs to individual ad scripts. Kontaxis and Chew [15] measured Firefox's Tracking Protection, finding a 44% median reduction in page load time and 39% reduction in data usage on Alexa top-200 news sites. The Ghostery Tracker Tax study [7] found a 2.25× page load speedup with trackers blocked, though Castell-Uroz et al. [4] challenged these findings at larger scale, reporting only 10–20% bandwidth savings and negligible latency improvement across 100K sites. WhoTracks.Me [13] monitors tracker prevalence from 780M+ page loads, reporting average content lengths per tracker domain. All of these works *measure* the cost of trackers on completed page loads; none *predicts* the cost of requests that were prevented from completing.

Predictive cost estimation for blocked resources. The closest prior work is Brave's ad-blocker savings predictor [1], which trains a linear regression model to estimate *page-level* bandwidth and JavaScript execution savings from ad blocking ($R^2 = 0.68$ for bandwidth). Their approach requires a paired crawl (with and without blocking) to generate labels and predicts aggregate page-level totals, not individual request costs. Our work differs in three respects: (1) we predict at per-request granularity, enabling attribution to specific URL patterns; (2) we require no paired crawl—the model trains on completed requests and deploys on blocked ones; (3) we address the counterfactual setting where the blocked response is structurally unobservable, not merely unobserved.

Gradient boosted trees on tabular data. XGBoost [5], LightGBM [14], and CatBoost [19] are the dominant methods for structured prediction tasks. Grinsztajn et al. [8] systematically compared trees to deep learning across 45 datasets and found trees dominant below ~10K samples and competitive at larger scales. Tweedie regression has long been standard in actuarial science for zero-inflated, heavy-tailed claims data [24, 26], and has been applied in ecology [23] and other domains with similar distributional properties. Our contribution is not the use of Tweedie loss itself but confirming that this actuarial insight transfers to web response data, where the distributional properties (39.5% exact zeros, heavy right tail) closely parallel insurance claims.

Covariate shift and counterfactual prediction. Our problem involves training on one population (completed HTTP Archive requests) and deploying on another (blocked Firefox requests). This is fundamentally a covariate shift setting [20, 22, 25]: $P(X)$ differs between training and deployment while $P(Y|X)$ is invariant, because transfer size is server-determined and URL-dependent, not browser-dependent. The shift is between HTTP Archive's crawl population (systematic, monthly, from fixed vantage points) and Firefox's real browsing population (organic, continuous, global). Importance weighting [22] could in principle correct for this marginal shift, but is unnecessary given that the prediction target is nearly deterministic given the URL and server state. The setting differs from standard covariate shift in one respect: deployment-population labels are permanently absent (blocked requests will never have responses), so the shift cannot be validated post-hoc. It also differs from counterfactual estimation [3, 11, 21], where both treated and control units have outcomes; here, the blocked response was never generated. Domain adaptation [2, 17], which allows $P(Y|X)$ to shift, is not needed because the same URL returns the same payload to Chrome and Firefox.

Tracker blocking. Firefox ETP [6], Brave Shields, Privacy Badger, and uBlock Origin make binary block/allow decisions. None estimates the performance cost of what was blocked, which is the gap this work addresses.

3 Problem Formulation

3.1 The Prediction Gap

When Firefox blocks a request to a tracker domain, the browser observes a feature vector $\mathbf{x} \in \mathcal{X}_{\text{pre}}$ consisting of the request URL, resource type, initiator, and HTTP metadata. The response quantities—transfer size y_{bytes}, download duration y_{dl}, and others—are structurally unobservable because the request was blocked before any response arrived. The task is to learn $f : \mathcal{X}_{\text{pre}} \to \mathbb{R}_{\geq 0}$ from completed requests in the HTTP Archive, where both $\mathbf{x}$ and y are observed.

LUT granularity	Entries	MAE (bytes)	Improvement
Global median	1	13,661	—
Domain	2,606	7,878	42.3%
Domain + resource type	~7,800	6,597	51.7%
Domain + URL path	227,336	3,797	72.2%

Table 1: Lookup table accuracy by granularity on the test set (523,624 requests). Adding URL path reduces MAE substantially but the table requires 227K entries, and 75% of unique test paths have no training match. The domain LUT has 2,606 entries (not all 3,723 domains appear in the training split; the remainder fall back to the global median). The "Improvement" column shows percent MAE reduction relative to the global median (MAE 13,661); higher is better. Section 6 uses the domain+type LUT (MAE 6,597) as the reference baseline for model improvement figures.

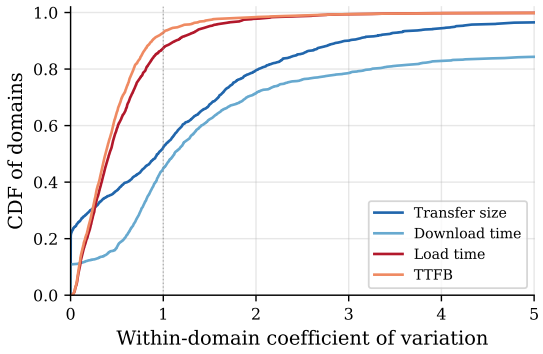

Figure 2: CDF of within-domain coefficient of variation by target. Transfer size and download time have high intra-domain variance (URL-predictable); TTFB and load time have low variance (network-dominated).

The labels are not randomly missing but are *structurally absent* for the inference population: blocked requests will never have responses, regardless of future data collection. This makes the problem a covariate shift with no possibility of post-deployment label collection, motivating careful offline evaluation (Sections 6.1–6.8).

3.2 Training–Deployment Compatibility

Transfer size is server-determined: the same URL returns the same payload regardless of the requesting browser. This means the conditional distribution $P(Y|X)$ does not shift between the training domain (Chrome, via HTTP Archive) and the deployment domain (Firefox). Standard domain adaptation concerns [2] therefore do not apply to transfer size prediction: the labeling function is invariant, and only the marginal $P(X)$ differs between crawl and organic browsing traffic. Training on Chrome-collected HTTP Archive data and deploying in Firefox is valid without importance weighting or domain-adversarial training. We verify the random split is IID: a gradient-boosted classifier trained to distinguish HTTP Archive training samples from held-out test samples achieves AUC = 0.557 (chance = 0.5), confirming no substantial systematic feature shift between splits. Deployment validity—training on Chrome HTTP Archive data and inferring in Firefox—is a separate argument and rests on server-determinism: transfer size is a property of the URL and server state, not the requesting client. Timing metrics depend on network conditions and client hardware; we discuss this limitation in Section 9.

3.3 Why Lookup Tables Are Insufficient

We systematically evaluate lookup tables at increasing granularity to establish the ceiling of non-model approaches.

Adding the URL path to the lookup key reduces MAE from 6,597 to 3,797 (Table 1), but this improvement is fragile. The path-level LUT achieves MAE of 1,448 on the 91.6% of test rows where an exact path match exists, but falls back to the domain+type median (MAE 29,491) on the remaining 8.4%—which are disproportionately expensive requests (mean transfer size 35,876 vs. 11,636 bytes for matched paths). Furthermore, the path LUT cannot scale to deployment: 227K entries from a 1% sample extrapolate to ~23M entries at full scale, and the URL space changes continuously as tracker SDKs update version numbers, campaign IDs, and endpoint structures.

The within-domain coefficient of variation of transfer size has median 0.94 and 90th percentile 3.00 (Figure 2). The within-(domain, resource type) CV drops to median 0.14, confirming that resource type explains most intra-domain variance, but a substantial tail remains (90th percentile 1.86) that only URL-level features can resolve. Figure 2 also shows that timing targets (TTFB, load time) have much lower within-domain variance, foreshadowing the multi-target results in Section 6.7.

4 Data

4.1 Source and Sampling

We use the HTTP Archive June 2024 mobile crawl, filtering to third-party tracker requests in five categories (advertising, analytics, social, tag-manager, and consent-provider) as classified by HTTP Archive's third-party entity table. The full dataset contains approximately 348 million tracker requests across 5,186 domains.

A 1% deterministic sample is drawn via hash-based sampling:

```
MOD(ABS(FARM_FINGERPRINT(page||url)), 100) = 0
```

yielding 3,490,824 requests across 3,723 domains. Hash-based sampling ensures reproducibility and uniform coverage across page–request combinations.

4.2 Features

Features are organized by what Firefox observes at block time (Table 2).

Domain target encoding. The tracker domain (3,723 unique values) is encoded as two statistics computed from the training split: median `transfer_bytes` per domain, and median per (domain, resource type). Unseen domains receive the global median. This encoding is the single most important feature (Section 6.4).

TF-IDF URL embeddings. We tokenize each URL path by splitting on delimiters (`/ ? & = . - _`) and camelCase boundaries, lowercased, filtering to tokens ≥ 2 characters. TF-IDF vectors are

Group	Features	Dim.
Domain identity	Target-encoded median, median by type	2
URL structure	Path depth, URL length, #query params, extension	12
URL content	TF-IDF + SVD embedding of URL path	50
Request metadata	Resource type, initiator type, HTTP method	10
URL patterns	Regex flags (js, collect, image, sync, ad, api)	6
Total		**80**

Table 2: Feature groups. URL structure includes one-hot encoded file extension (8 categories). Request metadata includes one-hot encoded resource type (6), initiator type (3), and POST indicator.

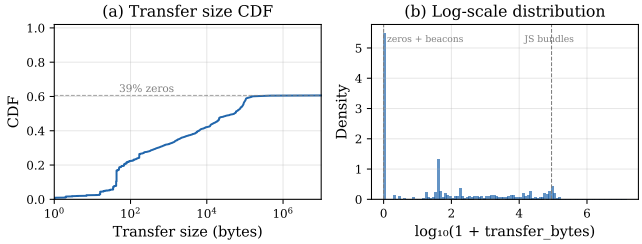

Figure 3: Transfer size distribution. (a) CDF showing that 39% of requests have zero transfer size (beacons) and the distribution spans five orders of magnitude. (b) Log-scale histogram revealing the bimodal structure: a spike at zero/near-zero and a secondary mode around 90 KB (JavaScript bundles).

computed with sublinear TF (log(1 + tf)), unigram and bigram features, vocabulary capped at 50K terms, minimum document frequency 5. Truncated SVD reduces the sparse matrix to 50 dense dimensions, explaining 54.5% of variance.

The leading SVD components are semantically interpretable: component 1 separates `/collect` endpoints (beacons) from all others; component 2 captures `/gtag/js` and `analytics.js` (script bundles); components 3–5 distinguish tracking pixels, ad-server paths, and cookie synchronization endpoints. These correspond to the distinctions that hand-crafted regex features target, but are discovered automatically.

Regex URL features. Six binary features flag URL path patterns: JavaScript (`js|sdk|gtm`), collection endpoints (`collect|beacon|pixel`), images (`gif|png|1x1`), sync (`sync|match|cookie`), ads (`/ad/|pagead|prebid`), and APIs (`/api/|/v[0-9]/`). These are included alongside TF-IDF embeddings; Section 6.3 ablates both.

4.3 Target Distribution

The primary target is `transfer_bytes`: on-wire response size in bytes. Its distribution is severely zero-inflated (39.5% exact zeros, mostly tracking beacons) with a heavy right tail (max 8.6 MB, mean 13,607, median 43), as shown in Figure 3. This bimodal structure—a spike at zero and a long right tail—motivates the Tweedie loss (Section 5). We also evaluate timing targets; see Section 6.7.

4.4 Data Split

Rows are split randomly: 70% training (2,443,576) and 15% each for validation and test (523,624 rows each). Row-level splitting is appropriate because Firefox will have HTTP Archive statistics for all Disconnect-list domains at deployment; domain identity is a known feature at inference time. Target encodings are computed exclusively from the training split.

5 Methods

5.1 Baselines

We evaluate four lookup table baselines of increasing granularity, each using the training-set median as the predicted value: (1) global median; (2) per-domain median with global fallback; (3) per-(domain, resource type) median with domain then global fallback; (4) per-(domain, URL path) median with (domain, type), domain, then global fallback.

5.2 XGBoost with Tweedie Loss

The zero-inflated, heavy-tailed transfer size distribution motivates the Tweedie loss [12], a standard choice for zero-inflated continuous data in actuarial modeling [24, 26], with variance power $p \in (1, 2)$:

$$\ell_{\text{Tweedie}}(\hat{y}, y) = -y\frac{\hat{y}^{1-p}}{1-p} + \frac{\hat{y}^{2-p}}{2-p}$$

This penalizes errors proportionally to predicted magnitude, focusing capacity on high-cost requests rather than the mass of near-zero beacons. We tune p via ablation (Section 6.2).

All XGBoost models use 500 trees, max depth 8, learning rate 0.05, row subsampling 0.8, column subsampling 0.7, minimum child weight 10, with early stopping (patience 20) on the validation set. Hyperparameters were set based on standard XGBoost defaults and prior experience on tabular regression tasks; no automated search was performed. An offset of +1 is applied to targets before training (reversed after prediction) to satisfy the Tweedie constraint $\hat{y} > 0$. The 500-tree count is justified by post-hoc depth pruning of the trained model via `iteration_range` (an upper bound on what fresh retraining would achieve). At 200 trees (~200 KB ONNX), MAE rises to 3,908—worse than the path LUT (3,797)—forfeiting the headline advantage; at 300 trees (~300 KB), MAE is 3,655, marginally better than the path LUT but within its bootstrap CI [3,623, 3,984] and not distinguishable at 95% confidence; only 500 trees (~500 KB) achieves the statistically significant improvement (Section 6.1).

Why no neural baselines. We do not evaluate deep learning architectures (e.g., MLPs or character-level URL encoders). The primary constraint is deployment: the model must ship as a compact artifact (~500 KB ONNX) for in-browser inference, ruling out large embedding-based architectures. Grinsztajn et al. [8] provide supporting evidence that tree-based models are competitive with neural networks on tabular data at this scale (~80 features, ~2.4M rows), though their study finds clear tree dominance primarily below ~10K samples. We note that a character-level URL model could in principle capture finer-grained patterns than TF-IDF embeddings, and leave this comparison to future work.

Approach	MAE	95% CI	Spearman ρ
Global median	13,661	—	—
Domain LUT	7,878	—	—
Domain+type LUT	6,597	[6,426, 6,790]	0.926
Path LUT (with fallback)	3,797	[3,623, 3,984]	0.934
XGBoost Tweedie	**3,466**	**[3,314, 3,627]**	**0.945**

Table 3: Transfer size prediction on the test set (523,624 requests). MAE in bytes. Bootstrap CIs from 1,000 resamples. The model's CI does not overlap with the path LUT's.

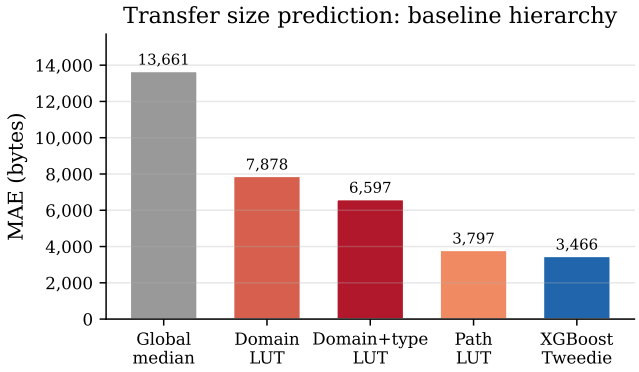

Figure 4: Transfer size MAE across the baseline hierarchy. Each step adds information: domain identity, resource type, URL path, and finally learned features via the model.

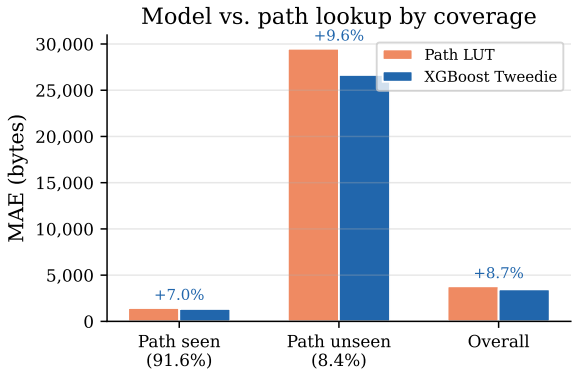

Figure 5: The model outperforms the path LUT on seen paths (regularization beats noisy memorization; non-overlapping bootstrap CIs) and shows lower point-estimate error on unseen paths (overlapping CIs; not statistically significant).

Subset	n	Path LUT MAE [95% CI]	Model MAE [95% CI]
Seen (91.6%)	479,775	1,448 [1,409–1,491]	**1,346 [1,317–1,384]**
Unseen (8.4%)†	43,849	29,491 [27,523–31,586]	26,655 [24,940–28,332]
Overall	523,624	3,797 [3,623–3,984]	**3,466 [3,314–3,627]**

Table 4: Model vs. path LUT decomposed by path coverage (bootstrap CIs from 1,000 resamples). The seen-path improvement is statistically significant (non-overlapping CIs). †The unseen-path improvement is a lower point estimate but CIs overlap; the difference is not statistically significant at 95% confidence.

6 Results

6.1 Baseline Comparison

Table 3 presents the main result: a hierarchy of approaches with bootstrap 95% confidence intervals.

The XGBoost model with Tweedie loss achieves the lowest MAE (3,466 bytes), reducing error by 47.5% relative to the domain+type LUT and by 8.7% relative to the path LUT (Figure 4). Notably, the model's confidence interval [3,314, 3,627] does not overlap with the path LUT's [3,623, 3,984], confirming the improvement is statistically significant. Bootstrap CIs are reported for the main comparison (Table 3) and temporal holdout (Table 12); ablation tables report point estimates only, as they compare models on identical test data and the relative ordering is stable across bootstrap resamples.

The model outperforms exact-path lookup. Decomposing by path coverage reveals a surprising result (Table 4, Figure 5). On the 91.6% of test rows where the path LUT has an exact training match, the model still achieves lower MAE (1,346 vs. 1,448). The model regularizes across similar URL structures: a path appearing 3 times in training has a noisy median, while the model smooths predictions using domain statistics, URL length, and TF-IDF embeddings. On the 8.4% of rows with unseen paths—which are disproportionately expensive (mean 35,876 bytes)—both approaches struggle; the model's point-estimate error (26,655) is lower than the path LUT's domain+type fallback (29,491), though bootstrap CIs overlap ([24,940–28,332] vs. [27,523–31,586]) and the improvement is not statistically significant at 95% confidence.

Smoothed LUT comparison. One might expect that Laplace-smoothed (add-k) path LUTs—which interpolate between the path-specific training mean and the global mean (unlike the median-based standard LUT baselines)—would close the gap to the model by regularizing low-count paths. We evaluate add-k smoothing for $k \in \{0.5, 1, 2, 5, 10, 20, 50, 100\}$. The best variant ($k = 0.5$) achieves MAE 4,011—5.6% *worse* than the path LUT (3,797) and 15.7% worse than the model (3,466). Smoothing degrades overall accuracy because it shrinks predictions for the 91.6% of test paths with reliable training statistics toward the global mean, increasing error on the majority to marginally reduce error on the minority. The model's advantage on seen paths (MAE 1,346) over even the lightly-smoothed LUT (MAE 1,686 at $k = 0.5$) confirms that the model learns genuinely better cross-path representations—combining URL structure, domain statistics, and resource type—rather than merely regularizing noisy low-count path memorization.

6.2 Loss Function Ablation

To isolate the effect of loss function from architecture, we train XGBoost with five loss configurations on identical data and features (Table 5).

Loss function	MAE	vs. D+T LUT	Spearman ρ
Squared error	4,527	+31.4%	0.738
Tweedie ($p = 1.2$)	3,486	+47.2%	0.937
Tweedie ($p = 1.5$)	**3,466**	**+47.5%**	**0.945**
Tweedie ($p = 1.8$)	3,597	+45.5%	0.949

Table 5: Loss function comparison for transfer size. Architecture (XGBoost, 500 trees) and features (80 dimensions) are held constant. "vs. D+T LUT" is MAE improvement over the domain+type lookup table (MAE 6,597). Huber loss ($\delta = 10{,}000$) diverged (MAE 19,373) and is omitted.

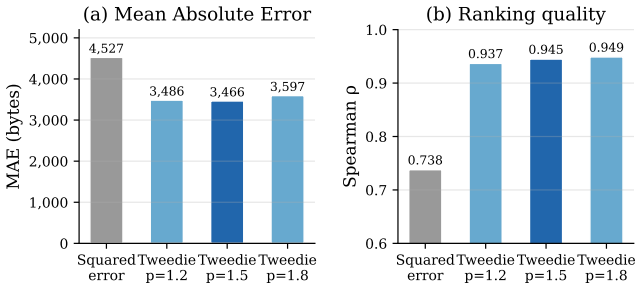

Figure 6: Loss function ablation. (a) Tweedie loss reduces MAE by 23% over squared error. (b) The Spearman ranking quality gap is even larger (0.945 vs. 0.738); ranking quality matters for per-domain attribution, which requires correctly ordering high-cost vs. low-cost requests.

Tweedie loss at $p = 1.5$ achieves the best MAE (3,466), outperforming squared error by 23% (3,466 vs. 4,527) on identical features (Figure 6). The Tweedie power parameter is relatively robust: $p \in [1.2, 1.8]$ all outperform squared error by at least 20%. The 20-byte difference between $p = 1.2$ (MAE 3,486) and $p = 1.5$ (MAE 3,466) is well within bootstrap variance; any $p \in [1.2, 1.5]$ is effectively equivalent, and $p = 1.5$ is used as a representative value. The mechanism is that squared error allocates model capacity uniformly across the prediction range, spending most effort on the 39.5% of near-zero beacons where any small prediction suffices, while Tweedie loss weights errors by predicted magnitude and focuses on the high-cost tail that dominates aggregate accuracy.

6.3 Feature Ablation

Table 6 ablates the two URL content feature groups: hand-crafted regex flags and TF-IDF embeddings.

TF-IDF embeddings alone (MAE 3,548) nearly match the full feature set (3,466) and substantially outperform regex features alone (4,251), as shown in Figure 7. Adding regex features on top of TF-IDF provides only a marginal 2.3% further improvement, suggesting the embeddings subsume most of the signal in the hand-crafted patterns. This is expected: the TF-IDF vocabulary captures `collect`, `pixel`, `sdk`, `sync`, and other patterns that the regex features target, while also discovering patterns the researcher did not anticipate (e.g., Privacy Sandbox `register-trigger` paths).

URL content features	MAE	vs. regex only
Regex only (6 binary features)	4,251	—
TF-IDF only (50 SVD dimensions)	3,548	+16.5%
Both (56 features)	**3,466**	**+18.5%**

Table 6: Feature ablation for URL content representation. All models use Tweedie loss and include domain, URL structure, and request metadata features. TF-IDF embeddings provide a larger improvement (16.5%) than adding regex features on top of them (2.3% additional MAE reduction).

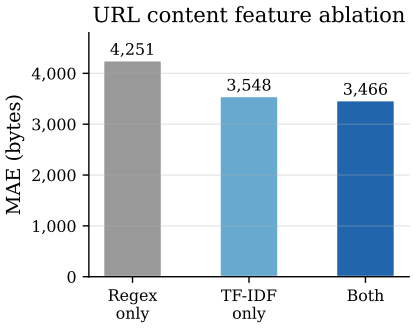

Figure 7: URL content feature ablation. TF-IDF embeddings alone nearly match the full model; hand-crafted regex features add little on top.

Feature group	SHAP %	Gain %
Domain identity	**48.2**	9.8
URL content (TF-IDF)	32.5	**52.9**
Request metadata	12.0	17.0
URL structure	6.0	11.4
URL patterns	1.3	8.9

Table 7: Feature group importance: SHAP (mean $|\phi|$, 8K-row sample) vs. gain-based importance. The two methods give inverted rankings for the top two groups. Domain identity is dominant by SHAP; TF-IDF appears dominant by gain due to gain bias toward high-cardinality continuous features.

6.4 Feature Importance

SHAP values (mean $|\phi|$) are computed on an 8,000-row sample via XGBoost's native `pred_contribs`, with target encodings recomputed from the same sample; this introduces slight in-sample bias for domain identity features, which is acceptable for group-level aggregate statistics. The result gives a substantially different picture from gain-based importance (Table 7). Domain identity features—the (domain, resource type) target-encoded median and the domain-level median—account for 48.2% of total mean $|\phi|$, more than any other group. TF-IDF URL content ranks second (32.5%), followed by request metadata (12.0%), URL structure (6.0%), and URL pattern flags (1.3%).

Gain-based importance inverts the top two groups: domain identity drops to 9.8% while TF-IDF inflates to 52.9%. This confirms

Resource type	n	D+T LUT MAE	Model MAE	vs. D+T LUT
Script	144,676	12,996	3,135	**+75.9%**
CSS	6,142	6,222	2,084	+66.5%
HTML	92,438	2,246	1,294	+42.4%
Image	136,698	8,330	7,428	+10.8%
Other (beacon)	98,617	16	16	+4.6%
Text	42,984	292	303	−3.8%

Table 8: Transfer size MAE by resource type. Improvement is concentrated on script (+75.9%) and CSS (+66.5%) resources, which have both high variance and URL-discriminative structure. The six categories cover 521,555 of 523,624 test rows; the remaining 2,069 rows (<0.4%) span resource types with fewer than 1,000 test examples each and are omitted.

N (req/week)	Model	Path LUT	D+T LUT
Median % error (2,000 trials):			
50	6.8%	8.4%	20.8%
100	6.3%	7.5%	20.9%
200	6.0%	6.8%	21.9%
500	5.1%	6.7%	23.2%
Fraction of weeks within 10% of truth:			
50	61.9%	—	24.9%
100	65.0%	—	23.1%
200	68.5%	—	16.8%
500	75.3%	—	5.3%

Table 9: Weekly aggregation accuracy for transfer size (2,000 simulation trials). Path LUT is impractical at deployment scale (Section 3.3) but shown for median % error; within-10% rates are reported for the model and D+T LUT only. The D+T LUT's systematic bias compounds with more requests (within-10% drops from 24.9% to 5.3%); the model's within-10% rate improves (61.9% → 75.3%) as errors cancel.

the known gain bias toward high-cardinality and continuous features [16]: 50 embedding dimensions each generate many high-gain splits, whereas the two target-encoded scalars accumulate impact in fewer but higher-leverage splits that gain undercounts.

Among individual features, `domain_type_median` is by far the strongest predictor (mean $|\phi| = 3.16$), nearly 8× larger than the next feature (`rt_other`: 0.40). The TF-IDF dimensions contribute collectively: 8 of the remaining top-15 features are embedding dimensions (mean $|\phi|$ 0.07–0.14 each) with no single dimension dominant, consistent with the distributed URL representation learned by SVD.

6.5 Results by Resource Type

The model's improvement is concentrated on resource types with high within-type variance and URL-discriminative structure (Table 8). Scripts see a 75.9% MAE reduction: the model distinguishes `/gtag/js` (93 KB bundles) from `/collect` (0-byte beacons) served by the same domain. For low-variance types ("other," mostly beacons; "text"), the D+T LUT's constant prediction is near-optimal and the model provides no benefit.

6.6 Aggregation Accuracy

Users see a weekly aggregate, not individual predictions. We simulate weekly browsing by sampling N blocked requests (with replacement, 2,000 trials), summing predicted and true transfer sizes, and computing percentage error.

At 200 blocked requests per week (a moderate browsing load), the model's weekly total has 6.0% median error, falling within 10% of truth 68.5% of the time (Table 9). The path LUT achieves a similar 6.8% median error at $N = 200$, but this baseline is impractical at deployment scale (Section 3.3). The practical deployed baseline—the domain+type LUT—has 21.9% median error, with only 16.8% of weeks within 10%.

The gap over the D+T LUT *widens* with more requests (Figure 8): at $N = 500$, the model improves to 5.1% median error while the D+T LUT degrades to 23.2%. This occurs because the D+T LUT has systematic directional bias for certain URL patterns (e.g., consistently overpredicting beacons and underpredicting scripts), which compounds in summation. The model's errors are approximately mean-zero and cancel via the law of large numbers.

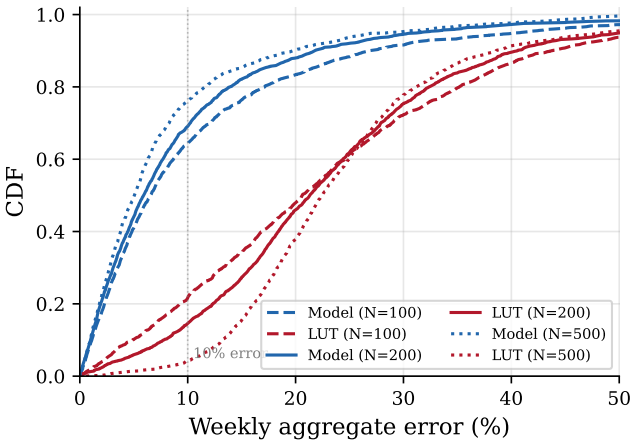

Figure 8: CDF of weekly aggregation error. The model (blue) concentrates mass near zero, while the D+T LUT (red) has a long tail of high-error weeks. The gap widens with more requests as the D+T LUT's systematic bias compounds.

Robustness to correlated browsing. The uniform simulation assumes i.i.d. request sampling, but real browsing is correlated: users tend to visit a small set of sites repeatedly. Since domain target encoding is the model's top feature, within-domain errors may be correlated and cancel less effectively. We test this by sampling 15 domains per trial (with replacement), then drawing N requests from those domains only.

Correlated browsing approximately doubles the model's median error at $N = 200$ (6.3% → 12.9%), confirming that within-domain error correlation reduces cancellation. The uniform column in Table 10 uses an independent 2,000-trial simulation from Table 9; the small difference from the 6.0% reported there reflects simulation variance. However, the D+T LUT's error increases even more

	Uniform		Domain-correlated	
N	Model	D+T LUT	Model	D+T LUT
50	7.1%	21.0%	14.2%	34.5%
100	6.7%	21.1%	12.5%	34.2%
200	6.3%	22.2%	12.9%	36.4%
500	5.0%	23.0%	11.2%	36.7%

Table 10: Aggregation accuracy under uniform vs. domain-correlated browsing (median % error; D+T LUT = domain+type LUT). Correlated browsing roughly doubles the model's error (6.3% → 12.9% at $N = 200$) but nearly triples the D+T LUT's (22.2% → 36.4%). The model's absolute advantage *widens* under correlation: the gap grows from 15.9 pp (uniform) to 23.5 pp (correlated) at $N = 200$.

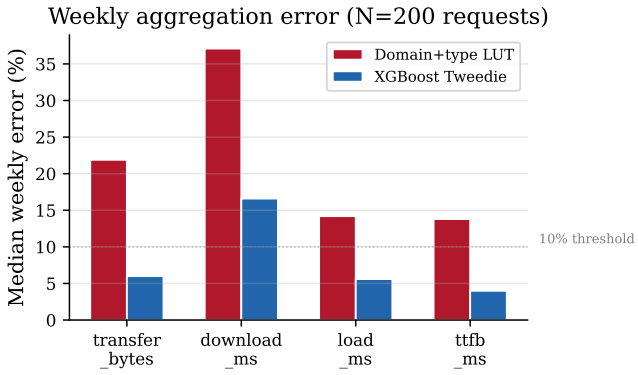

Figure 9: Weekly aggregation error across all four targets ($N = 200$). The model outperforms the D+T LUT on every target, including timing metrics where per-request MAE favored the D+T LUT.

Target	D+T LUT MAE	XGB MAE	vs. D+T LUT	Model agg.	D+T LUT agg.
`transfer_bytes`	6,597	**3,466**	+47.5%	**6.0%**	21.9%
`download_ms`	73	**56**	+23.7%	**16.6%**	37.1%
`load_ms`	117	121	−3.1%	**5.6%**	14.2%
`ttfb_ms`	67	76	−12.7%	**4.0%**	13.8%

Table 11: Multi-target results. "Model agg." and "D+T LUT agg." are median % error in weekly aggregation at $N = 200$. Per-request MAE improvement is strong for content-dependent metrics (transfer size, download duration) but absent for network-dependent latency. Aggregation error (last two columns) favors the model on all four targets, including TTFB and load time where per-request MAE was worse.

(22.2% → 36.4%). The model's absolute advantage *widens* under correlated browsing: at $N = 200$, the gap grows from 15.9 percentage points under uniform sampling to 23.5 pp under domain-correlated sampling—because the D+T LUT's systematic per-domain bias compounds more severely when browsing concentrates on fewer domains (Table 10). The model's advantage is robust to domain count: at $N = 200$, model error ranges from 11.4%–16.5% across $n_{\text{domains}} \in \{5, 10, 15, 20, 25\}$ vs. D+T LUT error from 34.2%–37.6%, confirming that the 15-domain choice does not qualitatively affect the conclusion.

6.7 Multi-Target Results

Beyond transfer size, we evaluate three timing targets using the same features and Tweedie loss.

Per-request MAE tells two distinct stories (Table 11). Transfer size (+47.5%) and download duration (+23.7%) improve substantially—both are *content-dependent* metrics where the URL determines response size, which is the primary driver of download time for large resources. TTFB (−12.7%) and total load time (−3.1%) do not improve—these are *network-dependent* metrics determined by server latency, geographic routing, and connection conditions. The within-domain CV confirms this: transfer size has CV 0.94 and download duration has comparably high within-domain variance (Figure 2), both URL-predictable; TTFB has CV 0.38 (low, noise-dominated variance).

However, weekly aggregation (last two columns) favors the model on *all four targets*: TTFB (4.0% vs. 13.8%), load time (5.6% vs. 14.2%), and notably download duration (16.6% vs. 37.1%), as shown in Figure 9. Even when per-request predictions are noisy, the model's errors are approximately mean-zero and cancel via aggregation; the D+T LUT's systematic bias compounds regardless of the metric. For timing metrics specifically, the D+T LUT's training-set medians reflect HTTP Archive's fixed crawl environment (Moto G4, cable connection); deployment conditions vary, introducing a directional shift in the LUT's predicted medians that does not cancel in aggregation. The model, despite higher per-request MAE on TTFB and load time, is less systematically biased because its URL features capture request-type variation that correlates with timing differences across conditions.

6.8 Temporal Generalization

To evaluate whether the model generalizes across time, we train on June 2024 data and test on a held-out September 2024 crawl (3 months later, 3,137,748 requests).

Drift analysis. Of the 3,612 tracker domains in September, 88.2% also appear in the June training set; 11.8% are new. URL path overlap is lower: only 14.9% of unique September paths were seen in June training, though these cover 85.1% of September *rows* (frequent paths recur; rare paths are ephemeral). The transfer size distribution is stable: June and September have identical medians (43 bytes) and similar means (13,621 vs. 14,258).

Results. The model's MAE degrades by 30.3% (3,466 → 4,517) over three months (Table 12). The path LUT degrades by a similar amount (30.8%), while the domain+type LUT barely changes (+1.9%)—its coarser granularity makes it more temporally stable but permanently less accurate.

Critically, the model retains a **32.8% advantage over the D+T LUT** on September data, compared to 47.5% on June. Ranking quality is nearly unchanged ($\rho = 0.942$ vs. 0.945). The weekly aggregation estimate degrades from 6.1% to 10.1% median error—still substantially below the D+T LUT's 21.5%. Scripts remain the

	June (in-dist)	Sep (temporal)	Degradation
Domain+type LUT MAE	6,597	6,723	+1.9%
Path LUT MAE	3,797	4,966	+30.8%
XGBoost Tweedie MAE	**3,466**	**4,517**	**+30.3%**
95% CI	[3,314, 3,627]	[4,458, 4,581]	
Spearman ρ	0.945	0.942	
vs. D+T LUT	+47.5%	+32.8%	
Aggregation (N=200):			
Model med. err	6.1%	10.1%	
D+T LUT med. err	21.8%	21.5%	
Scripts only:			
Model MAE	3,135	6,428	
D+T LUT MAE	12,996	13,021	
Improvement	+75.9%	+50.6%	

Table 12: Temporal generalization: train on June 2024, test on September 2024. The model degrades by 30% but retains a 32.8% advantage over the D+T LUT. Ranking quality (ρ) is nearly unchanged. Path coverage drops from 91.6% to 85.1%. The model and path LUT degrade similarly (~30%), while the D+T LUT is nearly stable (+1.9%) due to its coarser granularity—explaining why the relative advantage is retained despite absolute degradation.

Predicted size range	*n*	Pred. mean	Actual mean	Within 25%
0–100 B	26,462	20	63	51.3%
100–500 B	3,476	212	458	72.2%
500 B–1 KB	1,117	697	2,144	42.5%
1–5 KB	3,250	2,459	3,153	49.6%
5–10 KB	1,709	7,169	8,550	66.1%
10–25 KB	4,044	17,146	18,993	59.1%
25–50 KB	1,363	34,425	35,911	37.3%
50–100 KB	4,777	82,147	83,234	86.6%
100–250 KB	576	145,427	171,091	74.7%
250 KB+	120	671,042	640,626	25.8%

Table 13: Model calibration by *predicted* size bin. Each row groups test rows by predicted value; "Pred. mean" is the mean prediction and "Actual mean" is the mean true value for that group. The model is well-calibrated for large bundles (50–100 KB: predicted 82 KB, actual 83 KB) but underpredicts in the 500 B–1 KB stratum (predicted mean 697 B, actual mean 2,144 B): rows the model assigns to this range occasionally correspond to much larger actual responses (actual median 563 B; right tail pulls the mean up). Figure 10 plots the calibration curve.

strongest category, with the model achieving +50.6% improvement in September versus +75.9% in June.

The degradation is driven by URL path churn: only 14.9% of unique September paths were seen in June, compared to the 91.6% coverage within the June random split. This suggests **quarterly retraining** on fresh HTTP Archive data to maintain accuracy, which is feasible given the monthly crawl cadence.

6.9 Calibration Analysis

We evaluate whether the model is well-calibrated across the prediction range by partitioning test rows into bins defined by the *predicted* value and measuring the fraction of predictions within 25% of the true value (Table 13). Stratifying by predicted value isolates the model's behavior in each output regime; we report both the actual mean per bin against the predicted mean (testing mean-unbiasedness) and the fraction within 25% of the true value (testing interval accuracy).

The model is well-calibrated for the two dominant prediction regimes: near-zero predictions (0–100 B bin, 51.3% within 25%) and large-bundle predictions (50–100 KB bin, 86.6%)—the two modes of the transfer size distribution (Figure 3). The 500 B–1 KB predicted bin shows systematic underprediction: rows the model predicts in this range have actual mean 2,144 B and actual median 563 B. The divergence between mean (2,144) and median (563) signals that a minority of actual responses within this predicted stratum are much larger than 1 KB, pulling up the mean—consistent with the model correctly predicting *most* responses in this range but occasionally misclassifying large responses as small ones. The 1–5 KB bin shows a similar pattern (predicted mean 2,459 vs. actual mean 3,153, 49.6% within 25%). These cases correspond to JSON API responses and small CSS files that straddle the boundary between the near-zero beacon mass and the large-bundle mode—the predicted range with the sparsest test-set coverage (1,117 rows in Table 13; similarly sparse in training given the random split), where gradient signal from these examples is diluted by the much larger beacon cluster (26,462 rows, 0–100 B) and large-bundle mode. The 25–50 KB bin presents a different pattern: it is mean-calibrated (predicted mean 34 KB vs. actual mean 36 KB) but has notably low within-25% accuracy (37.3%) for a mean-calibrated bin, indicating high within-bin variance around the predicted range rather than systematic bias.

This calibration gap has limited impact on the primary use case. Aggregation across N requests reduces per-prediction bias as errors from different size ranges partially cancel (Table 9), producing 6.0% median weekly error at $N = 200$. However, for applications requiring per-request accuracy in the mid-range (e.g., attributing cost to a specific JSON tracking call), the model's estimates should be treated as order-of-magnitude estimates rather than precise values.

7 Analysis

7.1 Domain-Level Estimation Does Not Need ML

We initially attempted domain-level scoring: assigning each of 4,592 tracker domains (a broader set drawn from the full HTTP Archive entity list, prior to the per-request 1% sample filtering) a static cost score from aggregated HTTP Archive features. This formulation does not require machine learning. Of the 4,592 domains, 48% have direct Lighthouse CPU measurements; the remaining 52% fall below Lighthouse's ~50 ms reporting threshold and are uniformly inexpensive; a KS test confirms the two populations are feature-distinct across all dimensions ($p < 0.001$). A simple non-ML rule suffices: measured domains receive a cost score directly from their Lighthouse metrics; unmeasured domains are classified as

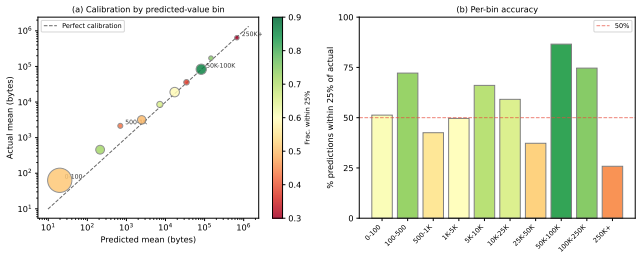

Figure 10: Model calibration by predicted size bin. (a) Log-log scatter of predicted mean vs. actual mean per bin; bubble size proportional to n; color encodes fraction within 25%. The dashed line is perfect calibration. The model is well-calibrated for the beacon mass (<100 B, 51.3% within 25%) and JS bundles (50–100 KB, 86.6%) but systematically underpredicts for rows it assigns to the 500 B–5 KB range. (b) Per-bin accuracy (fraction within 25% of actual).

inexpensive regardless of other features. The disjointness is structural: non-measured domains have no path-level statistics and no domain target encodings analogous to the Lighthouse-measured population—their feature vectors cluster at global-fallback values across all 80 dimensions. Self-training [27] (pseudo-labeling with confidence threshold 0.8, three iterations) produced zero improvement because high-confidence pseudo-labels were concentrated among inexpensive domains already well-covered by simple LUT rules. The per-request formulation, where blocked requests have no response and URL-level variance is too large for lookup tables (Section 3.3), is the genuine prediction problem.

7.2 Tweedie's Tail Prioritization Concentrates Gains in Aggregation

Beacons (39.5% of requests, near-zero transfer size) contribute negligibly to the weekly bandwidth total, while the high-cost scripts that Tweedie loss prioritizes dominate it. This means Tweedie's per-request gains are concentrated where they matter most for aggregation. Consistent with this, the largest per-request MAE improvements by resource type are scripts (+75.9%) and CSS (+66.5%)—precisely the high-transfer types that dominate weekly bandwidth totals (Table 8). Figure 11 confirms the model tracks the diagonal across five orders of magnitude, with tightest predictions in the 10 KB–100 KB range (JavaScript bundles).

8 Deployment Considerations

Feature availability. When ETP intercepts a request, Firefox's `nsIChannel` and `nsILoadInfo` interfaces expose the request URI (maps to all URL features), content policy type (maps to resource type), loading principal (maps to initiator type), and HTTP method. The TF-IDF vocabulary and SVD projection matrix are static artifacts (~2 MB) shipped alongside the model.

Model format. The XGBoost model exports to ONNX format (~500 KB for 500 trees). Inference latency is on the order of microseconds. Since cost estimates are used for background aggregation rather than real-time decisions, latency is not a constraint.

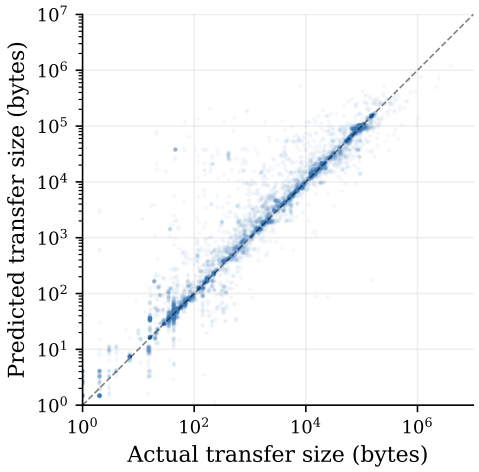

Figure 11: Predicted vs. actual transfer size (log scale, 20K random test samples). The model tracks the diagonal across five orders of magnitude.

Deployment status. This model is planned for integration into Firefox as part of a New Tab privacy widget, surfacing per-request cost estimates to users. Model updates would be delivered via Firefox Remote Settings, the same mechanism used for the Disconnect list, with quarterly retraining on fresh HTTP Archive crawl data to track changes in tracker SDK versions and URL structures (the HTTP Archive crawls monthly, so quarterly retraining is feasible with a one-crawl lag).

9 Limitations

Temporal stability. The model degrades by 30% over three months (June → September 2024; Section 6.8), driven by URL path churn (only 14.9% of unique September paths were seen in June). The model retains a 32.8% advantage over the D+T LUT on September data, but quarterly retraining on fresh HTTP Archive crawls is recommended for deployment. Longer-term degradation (6–12 months) has not been measured.

Timing metric validity. Transfer size is server-determined and browser-independent. Timing metrics depend on network stack, hardware, and conditions. HTTP Archive uses a fixed test environment (Moto G4, cable connection), so timing predictions are standardized-condition estimates rather than personalized predictions. We report timing results but recommend deploying only the transfer size model.

Training–deployment population shift. The model trains on HTTP Archive's third-party entity list and deploys on Firefox's Disconnect list. These overlap substantially but not completely. Disconnect-only domains absent from HTTP Archive receive the global median fallback. Model prediction error is only weakly correlated with domain out-of-distribution status ($r = 0.05$), confirming that covariate shift does not selectively degrade accuracy for Disconnect-only domains.

Calibration bias in the mid-range. As reported in Section 6.9, the model's 500 B–1 KB predicted stratum has actual mean 2,144 B—rows it assigns small predictions occasionally correspond to much larger actual responses. This affects JSON API responses and small CSS files at the boundary between the beacon mass and the large-bundle mode—the region with the sparsest coverage in Table 13 (the 500 B–1 KB predicted bin has only 1,117 test-set rows, and training density is comparably sparse). Per-request accuracy in the 500 B–5 KB predicted range should be treated as order-of-magnitude rather than precise; aggregation substantially mitigates this bias (Section 6.6).

Server-side variation. A/B tests, geo-targeting, and SDK versioning cause the same URL to return different payloads. This is irreducible noise: no pre-response feature can predict server-side randomization. Aggregation across many requests mitigates this.

Cascading blocking effects. Our predictions estimate the cost of each blocked request in isolation. In practice, many trackers act as loaders that dynamically inject additional third-party scripts: blocking a tag manager or consent provider can prevent an entire chain of downstream analytics and advertising requests from ever being issued. Since our cost estimates cover only the directly-blocked request, aggregated weekly totals represent a *lower bound* on the true bandwidth saved by ETP. Quantifying the cascading multiplier would require paired crawls (with and without blocking) of the kind Aucinas et al. [1] use to estimate page-level savings, which is orthogonal to the per-request prediction problem we address.

Ecological validity of aggregation simulation. Domain-correlated browsing simulation (Section 6.6) roughly doubles the model's weekly error (6.3% → 12.9%), confirming that error correlation reduces cancellation. The model's advantage is robust to domain count across $n_{\text{domains}} \in \{5, 10, 15, 20, 25\}$ (Section 6.6). The remaining unquantified uncertainty is not the domain count but the shape of the domain-visit distribution: the simulation selects domains uniformly at random from the test set, whereas real browsing tends to be more concentrated, with visits clustering on a small set of frequently-visited sites. Per-user telemetry would be required to characterize this further.

10 Conclusion

We have shown that the performance cost of blocked tracker requests—structurally unobservable because the response is never received—can be predicted from pre-response features with 6.0% median weekly aggregation error over the deployed baseline and 68.5% of weeks within 10% of true cost. Our analysis of 3,490,824 tracker requests reveals that within-domain transfer size variance spans three orders of magnitude, but this variance is predictable from URL structure: learned models outperform even exact-path memorization by regularizing across similar URL patterns.

Four findings generalize beyond the Firefox deployment context. First, for zero-inflated web response data, loss function selection (Tweedie vs. squared error, 23% MAE gap) is the dominant modeling choice—a gap that dwarfs the sensitivity to Tweedie parameter tuning (0.6%) and a reusable insight for web measurement tasks with similar distributional properties. Second, data-driven TF-IDF URL embeddings subsume hand-crafted regex features, reducing the need for domain-specific feature engineering in URL-based prediction tasks. Third, per-request prediction noise cancels in aggregation: model error is 6.0% under uniform sampling and 12.9% under domain-correlated browsing at $N = 200$—substantially below the D+T LUT's 21.9% and 36.4% respectively—enabling practical cost estimation from inherently noisy individual predictions. Fourth, for models with target-encoded categorical features, gain-based importance gives an inverted picture: gain attributes 52.9% to TF-IDF embeddings while SHAP attributes 48.2% to domain identity. Practitioners on URL-based prediction tasks should treat gain-based feature importance as unreliable when target encodings coexist with high-dimensional continuous features, and should prioritize domain-coverage data collection over further URL feature engineering. These results hold under correlated-browsing simulation, and the general framework—training on completed requests, deploying on blocked ones—applies to any browser intervention where labeled examples of the intercepted resource type can be collected from a proxy population.

11 Ethics

This work raises no ethical concerns and was conducted in accordance with the principles of the Menlo Report.

Data source. All training and evaluation data comes from the HTTP Archive [10], a public dataset of synthetic web crawls performed from a controlled test environment. No human subjects, real users, or browser telemetry are involved. The crawl visits public web pages with a standard test client (Moto G4 emulator on a cable connection) and records the resulting HTTP requests and responses. We use only the request and response metadata; no personally identifiable information is present in the dataset.

Institutional context. Both authors are employees of Mozilla, which develops Firefox. The work was performed as part of Mozilla's Privacy Engineering effort and reviewed internally prior to submission.

Use of predictions. The model is intended to produce aggregate metrics that quantify the bandwidth and time costs prevented by Enhanced Tracking Protection, surfaced to users in privacy reporting interfaces. It is not used to make blocking decisions: the Disconnect list [6] determines what is blocked, and our model only estimates the cost of an already-decided block. Misclassification therefore cannot cause over-blocking of legitimate content or under-blocking of trackers. The downstream risk of a poorly calibrated estimate is a misleading aggregate statistic in the user-facing privacy widget, which we mitigate through aggregation (Section 6.6) and quarterly retraining (Section 8).

Disclosure to affected parties. The tracker domains studied here are public infrastructure listed on the Disconnect blocklist. We do not interact with these domains beyond what HTTP Archive's public crawl already records, and we publish no information about them that is not already observable from a standard browser session.

Acknowledgments

The authors used Claude (Anthropic) for writing assistance during manuscript preparation. All research contributions, including problem formulation, methodology, experiments, results, and analysis, are the authors' own. The authors reviewed and edited all text and take full responsibility for the content of this paper.

References

[1] Andrius Aucinas, Istvan Haller, and Ben Livshits. 2019. Accurately Predicting Ad Blocker Savings. https://brave.com/blog/accurately-predicting-ad-blocker-savings/.

[2] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A Theory of Learning from Different Domains. *Machine Learning* 79, 1–2 (2010), 151–175.

[3] Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. 2013. Counterfactual Reasoning and Learning Systems: The Example of Computational Advertising. *Journal of Machine Learning Research* 14 (2013), 3207–3260.

[4] Ismael Castell-Uroz, Pere Barlet-Ros, and Josep Solé-Pareta. 2020. Demystifying Content-Blockers: A Large Scale Study of Actual Performance Gains. In *Conference on Network and Service Management (CNSM)*.

[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.

[6] Disconnect. 2024. Disconnect Tracking Protection List. https://github.com/disconnectme/disconnect-tracking-protection. Accessed: 2024-06-01.

[7] Ghostery. 2020. Tracker Tax: The Impact of Third-Party Trackers on Website Speed. https://www.ghostery.com/tracker-tax.

[8] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In *Advances in Neural Information Processing Systems*, Vol. 35.

[9] HTTP Archive. 2022. Web Almanac 2022: Third Parties. https://almanac.httparchive.org/en/2022/third-parties.

[10] HTTP Archive. 2024. HTTP Archive: Tracking How the Web is Built. https://httparchive.org.

[11] Fredrik Johansson, Uri Shalit, and David Sontag. 2016. Learning Representations for Counterfactual Inference. In *Proceedings of the 33rd International Conference on Machine Learning*. 3020–3029.

[12] Bent Jørgensen. 1987. Exponential Dispersion Models. *Journal of the Royal Statistical Society: Series B (Methodological)* 49, 2 (1987), 127–145.

[13] Arjaldo Karaj, Sam Macbeth, Rémi Berson, and Josep M Pujol. 2018. WhoTracks.Me: Shedding Light on the Opaque World of Online Tracking. In *arXiv preprint arXiv:1804.08959*.

[14] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, Vol. 30.

[15] Georgios Kontaxis and Monica Chew. 2015. Tracking Protection in Firefox for Privacy and Performance. In *IEEE Workshop on Web 2.0 Security and Privacy (W2SP)*.

[16] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, Vol. 30.

[17] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359.

[18] Behnam Pourghassemi, Jordan Ayers, Bobby Bhattacharjee, and Michelle L Mazurek. 2021. adPerf: Characterizing the Performance of Third-party Ads. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, Vol. 5. 1–26.

[19] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: Unbiased Boosting with Categorical Features. In *Advances in Neural Information Processing Systems*, Vol. 31.

[20] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. 2009. *Dataset Shift in Machine Learning*. MIT Press.

[21] Paul R Rosenbaum and Donald B Rubin. 1983. The Central Role of the Propensity Score in Observational Studies for Causal Effects. *Biometrika* 70, 1 (1983), 41–55.

[22] Hidetoshi Shimodaira. 2000. Improving Predictive Inference under Covariate Shift by Weighting the Log-Likelihood Function. *Journal of Statistical Planning and Inference* 90, 2 (2000), 227–244.

[23] Hiroshi Shono. 2008. Application of the Tweedie Distribution to Zero-Catch Data in CPUE Analysis. *Fisheries Research* 93, 1–2 (2008), 154–162.

[24] Gordon K Smyth and Bent Jørgensen. 2002. Fitting Tweedie's Compound Poisson Model to Insurance Claims Data: Dispersion Modelling. *ASTIN Bulletin* 32, 1 (2002), 143–157.

[25] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. 2007. Covariate Shift Adaptation by Importance Weighted Cross Validation. *Journal of Machine Learning Research* 8 (2007), 985–1005.

[26] Mario V Wüthrich and Michael Merz. 2008. *Stochastic Claims Reserving Methods in Insurance*. John Wiley & Sons.

[27] David Yarowsky. 1995. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. 189–196.