Universiteit Leiden

Master Computer Science

Analysing Static Tracker Detection: Comparative Insights into HTTP Response Header and Graph Topology

Name: Ruiqing Sun

Date: 20/08/2025

Specialisation: Data Science

1st supervisor: Dr. Akrati Saxena
2nd supervisor: Dr. Saber Salehkaleybar

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

For many users, “cookies”—small data files follow them from site to site, have come to symbolise online tracking. However, cookies represent only the most visible trace of tracking activities. Modern tracking techniques have developed rapidly beyond visible traces, relying on subtle server signals and deeply embedded resources that often evade defences. As a result, traditional methods such as detecting cookies or matching domains using blocklists alone is insufficient.

In this study, we revisit two emerging detection strategies: one based on protocol-level HTTP response headers, the other on static *page*→*resource* inclusion structures, and evaluate their effectiveness under static data collection constraints. We construct four feature spaces including two baselines and two combinations. These features are used to train and evaluate ten supervised classifiers under leak-free grouped cross-validation at upper/lower bound.

Results show that, under the upper bound (in-distribution) split, structure-based graph features perform best (OOF F1 ≈ 0.925, $P \approx 0.98$, $R \approx 0.88$). Enriching protocol features with header values and simple structural cues narrows the gap: the value space reaches F1 ≈ 0.915 ($P \approx 0.98$, $R \approx 0.86$) and the semantic space F1 ≈ 0.900 ($P \approx 0.923$, $R \approx 0.847$), whereas the header presence-only baseline is weakest (F1 ≈ 0.565, $P \approx 0.40$, $R \approx 0.98$). In the lower bound split that withholds entire eTLD+1 hosts (open-world proxy), all spaces degrade mainly through a precision drop; graph features remain strongest (F1 ≈ 0.689, $P \approx 0.544$, $R \approx 0.941$), the value and semantic spaces trail, and presence-only stays high-recall/low-precision. These patterns indicate complementary strengths: enriched protocol-layer information achieves precise recognition in familiar settings, while structural signals recover more trackers on unseen links but require additional precision control. Future work will integrate dynamic runtime signals, broaden dataset diversity and temporal coverage, reduce labelling noise, and explore hybrid two-threshold, human review involved designs.

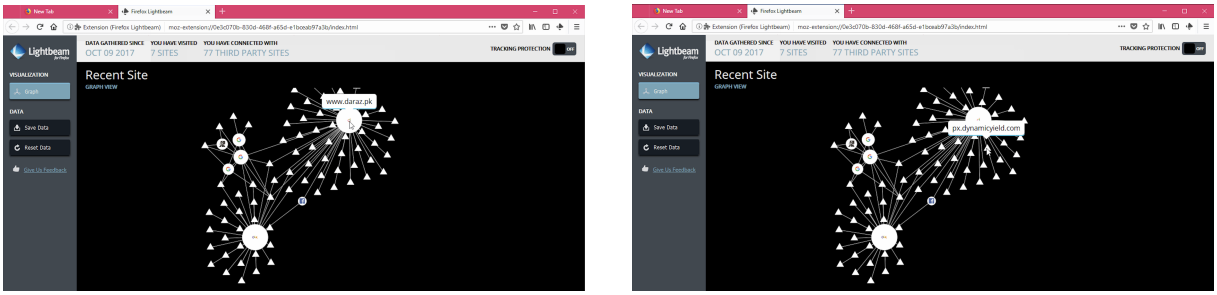
Contents

1 Introduction

It has become a common experience that cookie banners pop up when we click on a new website. These are not always the “cookies” we consent to, but traces of our online activities—activities that once remained private or shared with a few, but now leave digital trails revealing our interests and intentions [2]. Although cookies have become a visible representation in the context of online tracking, they are only one among various user data collecting methods. Other prevalent tracking mechanisms rely on embedded resource requests (e.g., images, scripts, or iframes) or identification techniques such as fingerprinting [14, 25]. These mechanisms are designed to collect device attributes, inject tracking pixels (typically small, invisible 1×1 images used to signal user presence or behaviour to external servers [28, 36]), or initiate cross-site requests involving background data exchanges with external domains [35].

1.1 Problem Statement

Empirical measurements have demonstrated that even a single webpage visit involves numerous third-party entities. For instance, Libert (2015) found that data-leaking sites contact an average of 9.47 distinct external domains [28]; Englehardt and Narayanan (2016) reported that a typical page includes 17.7 third-party resource requests [16]. Figure 1 shows that, a single website visit may expose users to a wide range of external entities including advertising networks. These figures are based on the Firefox Lightbeam extension, which visualises the relationships between visited first-party sites and third-party sites. The tracking techniques behind these hidden connections are intentionally engineered to evade detection, resist blocking strategies, and maintain persistent operation across platforms [1]. Consequently, the widespread deployment of tracking techniques introduces significant privacy concerns, while efforts to assess and quantify web tracking face substantial technical and methodological difficulties [16].



(a) `daraz.pk` as a visited site

(b) `px.dynamicyield.com` as a third-party node

Figure 1: Firefox Lightbeam visualisations showing connections between visited websites (circles) and third-party sites (triangles) during typical browsing. Multiple types of first-party sites can serve as hubs, connecting to numerous third parties. *Image source: Mozilla Lightbeam*

In response to public concerns and increased regulations such as GDPR which carried out in 2018, major browsers have tightened third-party cookie practices. Instead of cookies, they are promoting alternatives that claim to enhance privacy protection [38]. Google has

carried out this transition through new frameworks such as the Privacy Sandbox, which replaces individual-level tracking with on-device processing schemes. Other used frameworks are interest-based categorisation (Topics API) and decentralised model training(Federated Learning) [15,19,30]. Figure 2 shows the architecture of the Privacy Sandbox. However, this transition reflects not a commitment to privacy, but a strategic effort to maintain dominance in the ad tech ecosystem [15], and also leaves more potential space for first-party tracking. In fact, tracking technologies are not eliminating but evolving beyond cookies. They are becoming more technically sophisticated and less detectable to average users, regulators and privacy tools.

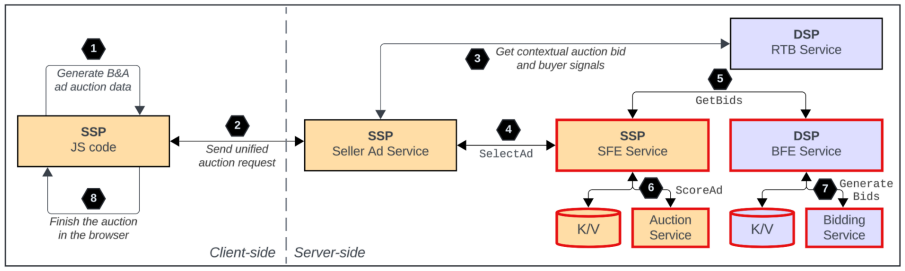


Figure 2: Architecture of the Privacy Sandbox Bidding & Auction (B&A) services for web advertising. The framework separates client-side (browser) and server-side operations. *Image source: Google Privacy Sandbox Documentation*

1.2 Research Objectives and Questions

Given this context, detecting trackers based solely on cookie presence or static blocklists is insufficient. There is a growing need to analyse a broader range of signals from resources embedded in web pages. Furthermore, the task is shifting from merely recognising known tracking entities to identifying previously unseen links that may serve tracking functions. We focusing on the *page* $\rightarrow$ *resource* references and all links embedded in those resources. Therefore, we put forward the following research questions:

RQ: Given an external link, how to determine whether it is a tracker?

To guide our study, we surveyed the literature on web tracking and categorised it into two broad classes: rule-based defences and machine learning (ML)-based detection. Following Mayer and Mitchell (2012), we treat opt-out mechanisms, static blocklists, and Do-Not-Track as representative rule-based approaches; they rely on manual preference review and voluntary compliance by service providers, and consequently suffer from structural limitations in scalability and coverage [29]. By contrast, ML-based detection seeks to address these gaps through features that transfer across sites. Among recent works, we found two directions: **protocol-based** approaches that exploit discriminative signals in HTTP response headers conducted by Rieder et al. (2025) [33], and **graph-based** models that capture webpage-resource inclusion topology and related context by Iqbal et al. (2020) [23].

1.3 Approach and Contributions

Building on ML-based approaches, we put forward three sub-questions to guide our study:

1. **Sub-Question 1 (Performance)**: How well do protocol-based and graph-based features detect trackers in static crawls?

2. **Sub-Question 2 (Generalisation)**: To what extent do these models transfer to unseen links, compared with close-world settings?

3. **Sub-Question 3 (Complementarity)**: Do protocol- and graph-level signals provide distinct or complementary evidence, and does their combination improve detection?

By addressing these questions, this study makes the three contributions:

1. We systematically evaluate protocol-based (HTTP response headers) and structure-based (inclusion graphs) detection across two grouped cross-validation schemes that approximate upper-bound and lower-bound deployments.

2. We design and assess heuristic value- and semantic-level encodings beyond presence-only, and combined with simple structural graph signals for the potential of tracker detection.

3. We quantify the generalisation gap to unseen links which challenges for open-world deployment.

1.4 Thesis Outline

The rest of the thesis is organised as follow: Chapter 2 provides an overview of existing definitions and related work on web tracking; Chapter 3 describes background concepts align with practice implementation; Chapter 4 describes our data collection and labelling procedures; Chapter 5 explains our detection methodology; Chapter 6 covers our experimental setup and presents the results; Chapter7 discusses broader implications of our findings and acknowledges the study's limitations; Chapter 8 concludes the thesis.

2 Literature Review

The definition of trackers has been evolving, we reviewed prior research on web tracking in this chapter, from the historical evolution of tracker definitions and techniques to the construction of ground truth labels. From a broader perspective, a "tracker" denotes mechanisms that enable identification or profiling of user behaviour across digital contexts, often by aggregating data and drawing inferences about individuals [2]. On the web, such mechanisms range from traditional identifiers (e.g., cookies, pixels) to advanced analytic techniques (e.g., fingerprinting), which together allow cross-site linkage and profiling [28]. In this study, we do not focus on defining trackers. Instead, we summarise previous works based on the papers we have reviewed to guide our understanding of the ground truth.

2.1 Evolution of Trackers

Krishnamurthy and Wills (2006) revealed that a few key third parties could track users across many unrelated sites. They introduced the "privacy footprint" to quantify aggregator domains [25]. Roesner et al. (2012) developed a behavioural taxonomy for web trackers which distinguishes between single-site and cross-site profilling [35]. Lerner et al. (2016) applied and expanded this framework in a historical study and observed that cross-site tracking had become increasingly prevalent and complex [26]. Su et al. studied the evolution of tracking in online educational websites and the impact of GDPR and CoVID-19 on that [37]. Englehardt and Narayanan's (2016) measurement results further illustrates the complexity of tracking, showing that 45 of the top 50 third-party trackers engaged in extensive cookie syncing (trackers exchanging identifiers to unify user tracking across websited) [16]. Together, these works underscore how the evolution of third-party tracking has significantly enhanced the ability to profile users across diverse websites.

While traditional web tracking relies on **stateful** identifiers such as cookies, researchers soon realised that browsers can be uniquely identified by their observable configuration—enabling **stateless** user tracking [14]. Eckersley (2010) provided the first large-scale empirical demonstration of browser fingerprinting, showing that this approach can uniquely identify users even in the absence of cookies. Later studies revealed that, by the mid-2010s, fingerprinting had become widespread, resilient to blocking, and deeply integrated into commercial tracking infrastructures [1, 31]. Notably, Nikiforakis et al. (2013) and Acar et al. (2014) showed how advanced fingerprinting methods—such as Flash, font probing, canvas fingerprinting, and evercookies—were deployed at scale and could circumvent conventional privacy tools.

More recently, the concept of tracking has further evolved beyond stateful and stateless identifiers through ML-based approaches. For example, Iqbal et al. (2020) introduced a browser-instrumented graph approach, linking HTML structure, JavaScript (JS) behaviour, and network flows to reveal complex tracking relationships [23]. Rieder et al. (2025) shifted focus on HTTP response headers, treating any host matching blocklists such as EasyList as a tracker and emphasising cross-site identification based on third-party request/response patterns [33]. At the same time, researchers such as Fouad et al. (2020) and Bahrami et al. (2022) have developed more detailed behavioural classifications, categorising types such as basic tracking, cookie syncing, analytics, and device fingerprinting, whether or not

explicit identifiers are present [5, 18].

2.2 Ground Truth and Tracker List Usage

A reliable ground truth is fundamental for ML-based detection. In prior work, ground truth is most often derived from publicly maintained filter/tracker lists as supervision. Four influential lists are commonly referenced:

- EasyList(EL): A filter list for hiding advertising elements; widely used in academic studies and real-world ad blockers [17, 28]. EL primarily targets page elements/URLs for visual ad removal.
- EasyPrivacy (EP): An extension of EL focusing on privacy-intrusive requests, often used to label tracking-related data flows in rule-based and measurement studies [17].
- Disconnect: A functionally annotated, frequently updated tracker list that explicitly targets tracker domains and services (e.g., advertising, analytics); widely used for research and benchmarking [28, 35].
- Ghostery: A tracker database with ownership and behaviour annotations; often used to benchmark privacy risks and evaluate blocking solutions [35].

These lists have been used in different roles in prior work: (1) supervised ground truth for training/evaluating ML models and (2) performance baselines to systematically assess how well tools identify trackers [13, 18, 28]. However, supervision based on these lists have limitations since they face manual curation delays and incomplete coverage and are vulnerable to evasion through domain rotation or content obfuscation [18, 23, 32], which causes noise in labelling.

2.3 Detecting Trackers

We further reviewed the literature with categorising them into two rule-based methods and ML-based models.

2.3.1 Rule-based Methods

Rule-based detection remains the key method of both web and mobile tracking defense and measurement. On the web, browser extensions and built-in features—such as Adblock Plus, Ghostery, Privacy Badger, Firefox Tracking Protection, and Safari ITP—use static blocklists and URL pattern matching to block or label tracking and advertising requests [16, 23, 25, 28, 33]. In the mobile context, static rules and SDK or analytics inclusion lists are used to identify tracking libraries, though these approaches are less standardised than web-based blocklists [34].

Policy- and signal-based tools such as Do Not Track, opt-out cookies, and AdChoices were created to give users more control, but in practice, most tracking companies disregard them [9, 28, 29]. Rule-based blocking can substantially reduce known tracker requests and offers clear logic to users, but it depends on ongoing manual curation, is slow to respond to new threats, and is easily circumvented through domain changes, code obfuscation, and

stateless tracking techniques [1,23,39]. On mobile, privacy protection increasingly depends on app store policies and Software Development Kit (SDK) default settings rather than direct network-level blocking. However, developer compliance and user understanding are often limited [13, 34]. Overall, while rule-based methods remain widely used, they are not sufficient on their own. Recent studies advocate for more adaptive detection methods that can keep pace with new and evolving tracking techniques.

2.3.2 ML-based Models and More

Driven by the limitations of rule-based detection, researchers have increasingly adopted machine learning and automated methods for web and mobile tracker detection. Early works applied classifiers such as SVM, decision trees, and Naive Bayes to features from HTTP headers, cookies, and sessions, achieving promising but often dataset specific results on selected web traffic samples [6, 7, 21, 27, 33].

More recent advances used supervised learning (e.g., Random Forests, Gradient Boosting) with richer features including detailed protocol, cookie, response header, and statistical traffic features to enhance detection accuracy and robustness [18, 20, 23, 32, 33]. For example, TrackAdvisor (Li et al. 2015) demonstrated that SVM-based cookie analysis could achieve nearly perfect precision and recall (99.4%/100%) on a manually labelled Alexa Top 10K subset [27].

While supervised learning on various features remains a key aspect of tracker detection, there are also a broadening range of methodological approaches in recent years. Graph embedding, label propagation, community detection, and hybrid clustering are used for fingerprinting, API sequence, and tracking pattern mining [5, 10, 24]. Mobile tracking remains less studied by ML, but initial work applies clustering, signature, and LLM-based policy analysis to identify SDK-driven privacy leaks [34].

Despite high accuracy and resilience to adversarial change, most ML models still depend on blocklist labelled ground truth for training and evaluation, and performance may suffer on truly novel tracking behaviour [18, 23, 32, 33]. The optimal practice is increasingly seen as a hybrid strategy, combining static rules, ML detection, and ongoing behavioural analysis to provide comprehensive and adaptive defense.

3 Preliminaries

In this chapter, we provide fundamental background on the structure of webpages and the granularity choices relevant to our study. These concepts align with our practical implementation.

3.1 Web Tracking Basics

Webpages comprises multiple resources, including HTML markup, cascading style sheets (CSS), JS programs, media files, and remotely hosted third-party components. The way in which these resources are included can reveal tracking behaviour. Therefore, it is important to understand how resources are organised and referenced in a modern browser context for tracking detection. We summarise the key compositions below:

- **The Document Object Model (DOM)**: The DOM is the tree representation of a page, where each node represents an object instance that can be queried and modified through APIs. Static HTML parsing reveals a subset of embedded resources present in the HTML source at crawl time and their immediate context in the DOM hierarchy.
- **Embedded resources**: The `<script>` tag embeds inline or external JS. Remote scripts originating from third-party hosts are a common medium for analytics and advertising code [3]. We download and store such scripts when encountered, and extract any absolute URLs (fully qualified URL) present in their source. Images, stylesheets and fonts from the visual layer of a web page, but can also be used for tracking. An inclusion is considered cross-site when the page and the embedded link have different registrable domains (`eTLD+1`).
- **Effective top-level domain (eTLD)**: Browsers conventionally define a "site" by using the Public Suffix List (PSL) to find the longest effective top-level domain (eTLD) of a hostname, and then combining the part before it (`eTLD+1`) [8, 11].

3.2 Research Granularity

In the web tracking context, the unit of analysis can be defined at several levels. The choice determines what is labelled, what is treated as context, and how generalisation is assessed. We define all resource referenced in a webpage as a **page** $\rightarrow$ **resource** connection. And we extracted all the **links** embedded in these resources. For each link, we record the hostname component, and the corresponding `eTLD+1` for grouping purposes in evaluation splits.

3.2.1 Labelling Strategy

We construct a merged tracker list from two public maintained sources: Disconnect and Ghostery, as introduced in chapter 2.2. From each tracker source we normalise to lowercase, trim whitespace, de-duplicate within source, and take the union across sources. Let $B = \text{Disconnect} \cup \text{Ghostery}$ denote the merged set of trackers. For each external link, we extract its host as the URL's netloc and mark a match by:

$$y = \begin{cases} 1 & \text{if hostname} \in B, \\ 0 & \text{otherwise.} \end{cases}$$

Labelling is therefore global and static at the **hostname** level. Any occurrence of a hostname in B is treated as a tracker, regardless of page context. By contrast, context-level labelling would assign labels per inclusion instance depending on context and blocker behaviour [16, 22]. While this global host-level labelling ensures consistency, it does not capture variations in tracker behaviour that depend on page context, resource path, or runtime execution. We revisit these limitations in Section 7.3.

3.2.2 Contextual Information

The unit of analysis is a (**page, external link**) pair, where the page refers to the full URL of the crawled document and the external link is the hostname component parsed from an HTML attribute. For feature computation, we derive its `eTLD+1` to aggregate statistics across subdomains. Thus, through the practical pineline we categorie:

1. Signals available at crawl time:
    - Page level: HTTP status code of the top-level page load.
    - Reference resource level: the stored external link.
2. Signals derived at analysis time:
    - URL-level cues: the file-path extension or public suffix of the hostname.
    - Domain-level aggregates: counts of how widely each registrable domain is referenced across pages.
    - Site relation: first- vs. third-party, based on whether the page URL and the external reference share the same `eTLD+1`.

Table 1 shows a few examples illustrating different granularities: the full URL from the crawling; the embedded external link, its extract hostname and registrable domain.

Raw URL	external link	hostname	eTLD+1
`https://www.example.com/`	`cdn.example.net`	`cdn.example.net`	`example.net`
`https://blog.example.com/`	`https://pixel.example.org/x.gif`	`pixel.example.org`	`example.org`

Table 1: Example of Different Granularity

4 Data Collection

This chapter describes the static web crawling pipeline and the overview of the resulting database structure. We also conducted a set of preliminary data analysis for guiding further feature engineering.

4.1 Overall Crawling Pipeline

The initial seed list was acquired from the publicly available platform `Common Crawl Index` using filters `.nl`, `HTTP 200`, `HTML` and stored in SQLite by another member of the research team—Daniel Gelencser. The crawl was completed in April 2025, resulting in approximately 1.85 million raw URLs, with 990,451 `.nl` hosts targeted for analysis. Among these, 989,504 pages were successfully retrieved ($\approx 99.9\%$). The crawling architecture is shown in Figure 3.

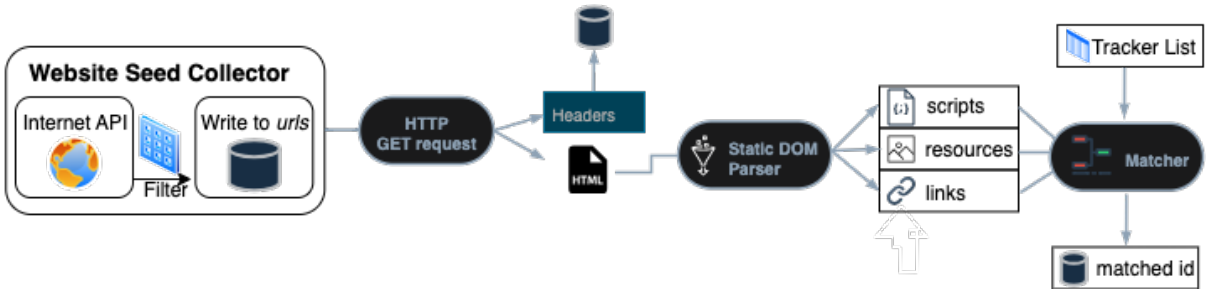


Figure 3: Architecture of Our Data Collection

For each page, we record:

1. HTTP response headers of the page request.
2. External references from static HTML, extracted from `<a href>`, `<script src>`, `<img src>`, and `<iframe src>` elements.
3. Embedded resources and their outbound links from retrieved JS and other resource bodies (`resources_test` and `scripts_test` tables, and their associated tables).

A hostname level matching step is then applied against a ground truth tracker list (see Chapter 3.2.1). Each matched host is assigned a `trackers_id` and labelled as a tracker in all occurrences.

4.2 Database Overview

In our static crawl, we observed that approximately 97.1% of all extracted references were captured by direct HTML parsing (the `urls_test_links_test` table). This result is expected given our design choices: we do not execute JS, we only fetch and statically scan same site `.js` files. Consequently, links injected at runtime (e.g., through DOM APIs), protocol-relative URLs, and third-party script payloads are out of scope. We therefore centre our main experiments on this table of directly embedded links, which dominate under static measurement and are lightweight reproducible. Header-based features for these

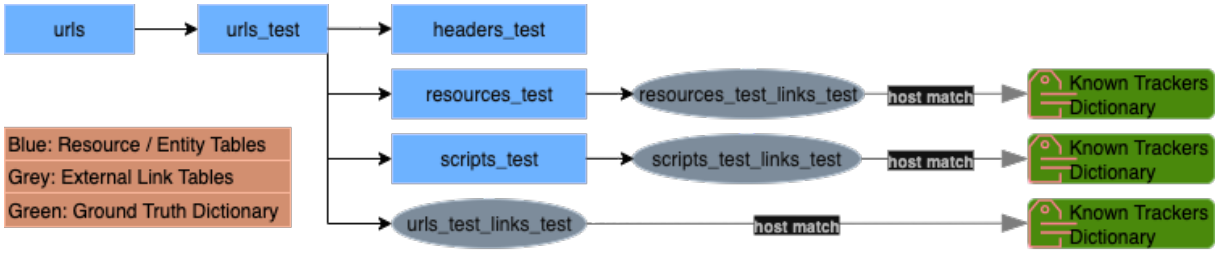


Figure 4: Multi-level Structure of the Static Collection Database

pairs are drawn from `headers_test`, which contains 45,002 total rows of `(urls_id, header_name, header_value)`. All records can be linked back to the source page through `urls_id`. The relation within stored data is illustrated in Figure 4.

4.3 Preliminary Data Analysis

We performed a preliminary analysis to understand label balance, tracker-frequency distribution, and HTTP header patterns.

4.3.1 Label Distribution and Tracker Domain Frequency

The final distribution of the labelled dataset is listed as follows 2.

Label	Records (%)
Tracker	8,457 (38.8%)
Non-tracker	13,336 (61.2%)
Total	21,793 (100%)

Table 2: Label Distribution of the Final Dataset

Tracker prevalence exhibits a **long-tail** [16], which indicates a small number of high-frequency domains account for the majority of tracker inclusions. As shown in Table 3, the top 10 tracker domains account for ≈65.3% of all tracker-labelled samples.

Known Tracker Domain	Count
`gstatic.com`	1530
`parastorage.com`	789
`googleapis.com`	762
`facebook.com`	565
`googletagmanager.com`	489
`wp.com`	335
`instagram.com`	310
`gmpg.org`	267
`google.com`	262
`linkedin.com`	212

Table 3: Top 10 Tracker Domains and Their Frequencies

This observations guide us with following insights into practical design:

- **Class imbalance handling**: The long-tail distribution requires weighting or resampling strategies to prevent bias towards the majority class.

- **Fair evaluation**: To avoid overfitting from frequent domains appearing in both training and validation sets, we consider grouping folds by $\texttt{eTLD}+1$, ensuring that all subdomains of a registrable domain appear in only one fold (e.g., `www.tracker.com` and `analytics.tracker.com` share the same $\texttt{eTLD}+1$). This is further detailed in Chapter 5.2.
- **Feature popularity**: For each registrable domain, we count the number of distinct pages in which it appears, as an indicator of how common the domain is. To avoid information leakage, this statistic is computed within each training fold only, and the resulting values are applied to the corresponding validation fold.

4.3.2 HTTP Response Header Presence

Since our sample unit is a (page, external link) pair, each sample inherits the HTTP response headers of its source page. For each header name, we create a binary presence indicator and compare occurrence rates between tracker and non-tracker samples using a chi-square test (χ^2, $p<0.05$).

Figure 5 shows headers with statistically significant between-class differences. Effect sizes are small, we therefore treat them as weak, additive signals rather than hard rules. While individual gaps are modest (typically ≈0.4–1.5 percentage points), the differences are consistent across multiple fields, indicating that headers can serve as complementary, low-cost features for tracker detection This observation is consistent with findings in prior work. Accordingly, in later experiments we include enhanced header features to evaluate their further potential.

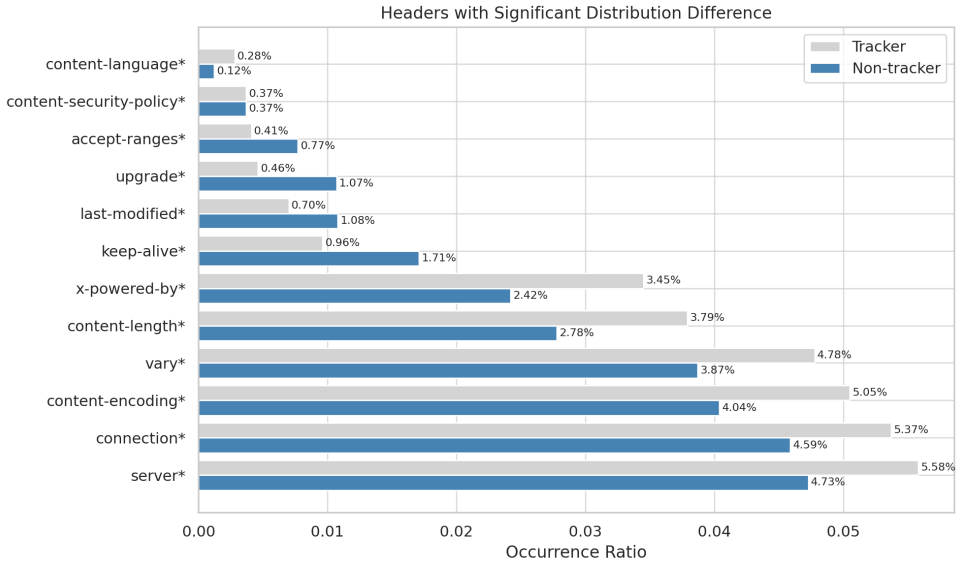


Figure 5: HTTP Response Headers with Significant Distribution Difference

4.4 Data Usage Statement

The initial list of Dutch websites was sampled from Common Crawl's public index (CC-MAIN series) and used strictly in accordance with Common Crawl's Terms of Use. We then fetched

only publicly accessible HTML over HTTP(S) with a standard user-agent; no authentication or access controls were bypassed, and JavaScript was not executed. The primary dataset used in this thesis comes from the `CC-MAIN-2024-51` snapshot (with crawl identifiers stored in the database). As Common Crawl is continuously updated, our corpus should be viewed as a time-bounded snapshot; all findings are interpreted with respect to this crawl window.

The data collection process, including the development and execution of the web crawler scripts, was carried out by another member of the research team. The author of this thesis conducted all subsequent data analysis, feature extraction, model training, and performance evaluation based on the collected dataset.

To avoid reputational concerns, we do not name specific websites or companies in the text. Illustrative examples use reserved placeholder domains (e.g., `example.com`), and we report only aggregate statistics.

5 Methodology

In this chapter, we detail our pipeline for evaluating protocol-based, structure-based and combination features for identifying web trackers on a statically collected dataset. The whole pipeline is semi-automated which separated by data collection part and ML part. We report all experiments under two mutually exclusive validation schemes: (1) page-group, representing an upper bound, and (2) link-group, representing a lower bound for strict generalisation to unseen sites. We construct four distinct feature spaces and evaluate them with a range of supervised classifiers. Performance is assessed using out-of-fold (OOF) predictions from grouped cross-validation with multiple metrics. Figure 6 illustrates the whole pineline.

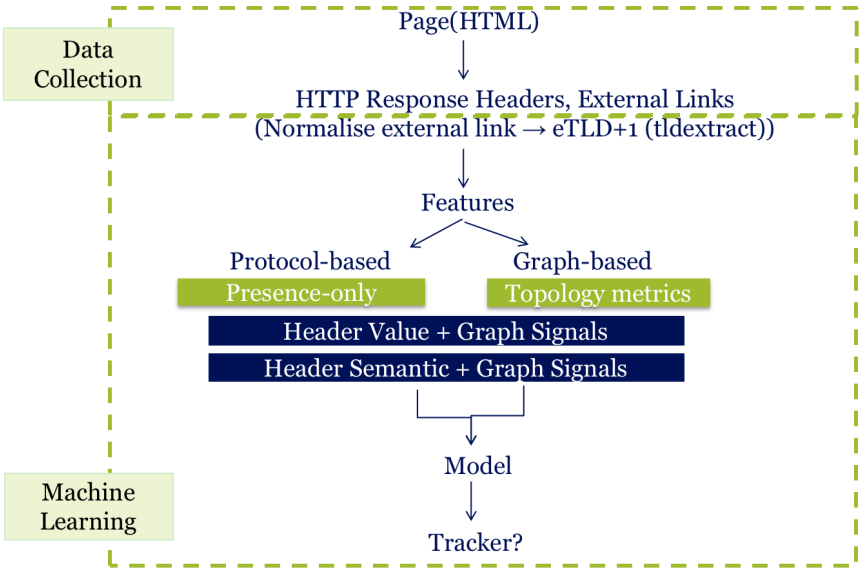


Figure 6: Semi-automated Tracker Detection Architecture

5.1 Problem Formulation

Let $D = \{(x_i, y_i)\}_{i=1}^{N}$ denote the modelling dataset, where each instance corresponds to a unique pair (p, h). Here, p is the unique page ID of a crawled page, and h is the hostname of a referenced resource in the page's HTML or embedded resources (ℓ). From ℓ we derive its hostname $h = \text{host}(\ell)$ and its $\text{eTLD+1}(\ell)$; similarly, from p we derive $\text{eTLD+1}(p)$. We focus on cross-site inclusions, determined by whether the $\texttt{eTLD+1}$ of h differs from that of p.

Each $x_i \in R^d$ is a feature vector derived from the page's HTTP response headers or graph-structural signals, and $y_i \in \{0, 1\}$ indicates whether the referenced link is listed as a tracker in the ground truth traker list. Therefore, our goal is to learn a classifier $f : R^d \to \{0, 1\}$ that minimises the loss:

$$R(f) = \frac{1}{N} \sum_{i=1}^{N} L\big(y_i, f(x_i)\big),$$

where x_i can be sparse and high dimensional due to heterogeneous headers.

5.2 Validation Schemes

In tracker detection, samples from the same registrable domain often share highly similar patterns such as particular HTTP header values, resource paths, or graph structures. Random split may lead the model with high predictive ability due to information leakage. Therefore we chose group split strategy. However, without strict separation, group split can still lead to overly optimistic performance estimates. As shown in table 4, even in different pages, the embedded external links can share the same eTLD+1.

URL ID	Page URL	External Link	Normalised Link
201	https://www.example.com/	cdn.example.net	example.net
201	https://www.example.com/	https://gtm.example.org/gtm.js?	example.org
202	https://blog.example.com/	https://pixel.example.org/x.gif	example.org

Table 4: Example of Data Inhert relationships

Due to the fact that some prior works point out such overlap can cause over positivity [4,12], we design two mutually exclusive grouped cross-validation strategies for close-world and open-world evaluation:

- Page-group (in-distribution): We use k-fold grouped cross-validation where folds are defined by the unique page ID of the crawled page (no page is split across fold). This allows the same eTLD+1 link from different pages appear in both training and validation. We treat this as an upper bound that reflects in-distribution deployment.
- Link-group (open-world deployment): We use k-fold grouped cross-validation split by the normalised external link This grouping strategy ensures no same eTLD+1 appears in both training and validation, which represents model's generalisation to unseen links.

In both cases, all feature selection and aggregation steps are performed strictly within each training fold. Although the page-group could achieve over positive performance, it is still valid according to real-world tracking activities. Page-group evaluation approximates the steady reality in which a detector runs within the same ecosystem and repeatedly encounters the same third-party links on different pages. In practice, advertising and analytics services are reused site-wide; protocol signals (e.g., headers, caching, cookie policies) and inclusion patterns are relatively stable across pages; and deployed detectors are continually updated with data from the same environment. Evaluating on different pages that reference the same links therefore measures in-distribution performance. The upper bound is relevant for setting thresholds and estimating routinely precision/recall—complementary where the link-group could test open-world discovery of unseen links.

5.3 Feature Space

With protocol-based (HTTP response header presence signal) and graph-based (page $\rightarrow$ external link inclusion topology) as baseline, we further designed two combination feature spaces with protocol-based HTTP response header of enhanced encoding rules together with simple structural signals. The four feature spaces are described as follows:

- **Protocol-based baseline:** Let (p, e) denote an instance, where p is the unique page ID of a crawled page and e is the registrable domain of a referenced third-party resource. From the HTTP response of p, let $\mathscr{H}(p)$ be the set of distinct header names. Within each training fold, we retain only those header names whose frequency in the fold is at least τ times, denoted as $\mathscr{S}_{\text{freq}}$. The binary feature vector $x_i \in \{0,1\}^{|\mathscr{S}_{\text{freq}}|}$ is defined by

$$x_{ij} = \begin{cases} 1 & \text{if } s_j \in \mathscr{S}_{\text{freq}} \text{ is present in } \mathscr{H}(p), \\ 0 & \text{otherwise.} \end{cases}$$

- **Graph-based baseline:** Inclusion relationships are modelled as a directed graph $G = (V, E)$. The graph construction and feature extraction are:

  1. Node definition: u is the $\mathrm{eTLD{+}1}$ extracted from the page's URL, treated as the source node; destination node v is the $\mathrm{eTLD{+}1}$ of the external link.
  2. Edge definition: For each (p, h) pair with $\mathrm{eTLD{+}1}(p) = u$ and $\mathrm{eTLD{+}1}(h) = v$, we add a directed edge $u \to v$ to the graph. Only cross-site inclusions ($u \neq v$) are considered; self-loops are removed.
  3. Feature extraction: PageRank and betweenness centrality are computed on the directed G, while triangle count and core number are computed on the undirected projection G^{und}. For every sample $(u \to v)$ we then join both source-side metrics and destination-side metrics to the edge. Nodes unseen in the training fold are imputed with zeros, and feature columns are aligned between train/validation with missing columns filled by zero to avoid any information leakage.

- **Header value with graph signals:** Let $\mathscr{P}(p)$ be the multiset of $(\text{header}, \text{value})$ pairs from page p. Within training fold f, we select the top-N frequent pairs $\mathscr{S}_{\text{pair}}^{(f)} = \{(h_j, v_j)\}_{j=1}^{N}$ using *only* training pages and build a binary vector

$$x_{ij}^{(f)}(p) = \mathbf{1}\big[(h_j, v_j) \in \mathscr{P}(p)\big], \quad x^{(f)}(p) \in \{0,1\}^{|\mathscr{S}_{\text{pair}}^{(f)}|}.$$

  Validation is reindexed to $\mathscr{S}_{\text{pair}}^{(f)}$ with missing entries set to 0. For each link-level sample $(u \to v)$ where $u = \mathrm{eTLD{+}1}(\text{page})$ and $v = \mathrm{eTLD{+}1}(\text{external})$, we append graph signals computed within fold f: (1) third-party indicator $\tau(u, v) = \mathbf{1}[u \neq v]$; (2) file-extension one-hots of the external URL, with the column set fixed by the training split; (3) fold-local destination popularity

$$\text{ref\_by\_pages}^{(f)}(v) = \Big|\{\, p' \in \mathscr{D}_{\text{train}}^{(f)} : \mathrm{eTLD{+}1}(\text{external of } p') = v \,\}\Big|,$$

  joined to $(u \to v)$ and zero-imputed when unseen in validation.

- **Header semantic with graph signals:** From headers on page p, we derive page-level semantic features using transforms fitted *only* on the training fold and then broadcast to links: (1) token cues for a fixed vocabulary (e.g., `gzip`, `nginx`, `cloudflare`), including per-header token count/density; (2) string statistics (max value length, digit

presence, HTTP-date detection for `date`/`last-modified`/`expires`); (3) page-level statistics: distinct header count and header-name entropy

$$\text{header\_entropy}(p) = -\sum_h q_h(p)\log q_h(p), \;\; q_h(p) = \frac{\#\{(h,\cdot) \in \mathscr{P}(p)\}}{|\mathscr{P}(p)|};$$

(4) selected header co-occurrence indicators (e.g., $[\texttt{server} \wedge \texttt{set-cookie}]$); (5) per-header TF–IDF encoders and k-means clusters (with $k \geq 2$ and capped by unique values), applied to validation via transform-only and reindexing with zeros. The same graph signals as above—$\tau(u,v)$, extension one-hots (training-defined column space), and $\text{ref\_by\_pages}^{(f)}(v)$—are appended to each $(u \to v)$ sample.

5.4 Model Selection and Training

Model training follows a supervised learning framework on labelled data (X, y) to identify the best classifier for tracker detection, evaluating the two validation schemes. Class imbalance is addressed by assigning class weight by approximately 2.5–3 due to trackers being the majority class in classification task. The weight for class j is computed as:

$$w_j = \frac{n_{\text{samples}}}{n_{\text{classes}} \times n_j},$$

where n_j denotes the number of samples in class j.

Ten classifiers are evaluated for each feature set: Random Forest (RF), Extra Trees (ET), Gradient Boosting (GB), LightGBM (LGBM), HistGradientBoosting (HGB), AdaBoost, XGBoost (XGB), Decision Tree (DT), Logistic Regression (LR), and Gaussian Naive Bayes (NB). Hyperparameters are fixed to reasonable values without extensive automated tuning. Probability calibration is applied to ensure reliable probability outputs for threshold optimisation.

5.5 Evaluation Metrics

Model performance is assessed on the validation folds using accuracy, F1-score, precision, recall, ROC-AUC, average precision (AUPRC), negative log-loss, and Matthews correlation coefficient (MCC). Given the imbalanced nature of the task, we prioritise F1-score as the primary objective during model selection.

Binary decision thresholds are selected through precision–recall analysis. For recall ≥ 0.7, we select the threshold that maximises the F1-score. In practice, over-flagging non-tracker links can be problematic. Excessive false positives may lead to the blocking of legitimate third-party services (e.g., CDNs, fonts, analytics) and reduce user trust in the system's accuracy. However in our settings, false positives can go into human review. Operating at higher recall (even with lower precision) is therefore intentional: it minimises missed trackers while the review stage filters out benign trackers.

6 Experiments

In this chapter, we present the experimental setup and main results of our systematic evaluation of tracker detection methods. Our experiments implemented under four feature spaces of two baselines and two combinations across ten supervised classifiers and eight evaluation metrics.

6.1 Feature Engineering

All features are extracted from the same labelled set of (page, external link) pairs. We treat header presence-only and inclusion topology as the protocol-based and graph-based **baselines**, respectively. The header value and semantic models are **combinations** with simple structural signals. We give examples of each feature space with exact parameter choice.

6.1.1 Protocol-based Baseline: Header Presence-only

We first construct a binary indicator matrix over HTTP response headers. Header names are lowercased and trimmed. To reduce the sparsity of the feature matrix, we retain only headers that appear at least 10 times in each training fold. Table 5 shows an example encoding with header description (not encoded in the experiment).

Feature	Description	Value
Basic identifiers		
`urls_id`		0
`page_url`		`https://example.com`
`external_link`		`http://tracker.com/script.js`
`True_Label`		0
Protocol-level presence-only features		
`accept_ranges`	`Accept-Ranges` header present	0
`cache_control`	`Cache-Control` header present	0
`server`	Server header present	0
`content_length`	`Content-Length` present	0

Table 5: Example presence-only encoding

6.1.2 Graph-based Baseline: Inclusion Topology

Node metrics PageRank, in/out-degree, triangles, core, betweenness are computed on the train subgraph. For each sample, we join metrics twice. Table 6

urls_id	page_etld1 (src)	link_etld1 (dst)	pr	outdeg	pr_dst	indeg_dst
201	example.com	example.net	0.21	1	0.58	2
202	example.net	example.org	0.58	0	0.21	0
203	new.example.xyz	example.org	0.00	0	0.21	0

Table 6: Example inclusion topology encoding

6.1.3 Header Value combined Graph Signals

The first combination we used heuristic lexical one-hot, header value and simple structural counts together with header presence. Within each training fold, we keep header presence and select the top-100 most frequent (header, value) pairs. We then append three lightweight features: (1) a lexical type cue: file-extension one-hot features (`ext_*`) inferred from the referenced URL (e.g., `.js`, `.gif`); and two structural cues: (2) a `third_party` indicator at the registrable-domain level; and (3) `ref_by_pages`, the count of distinct training pages linking to the same external `eTLD+1`.

Feature	Description	Value
Basic identifiers		
urls_id		
page_url		https://example.nl
external_link		http://tracker.com/script.js
True_Label		1
fold		1
Example value encoding features with hit count 7		
server_nginx	Response header reports Nginx server	1
content_encoding_gzip	Response is compressed with GZIP	1
content_type_application/javascript	MIME type indicates JavaScript file	1
is_third_party	Cross-site inclusion	1
ext_js	URL ends with ".js"	1
ref_by_pages	Linked from 45 distinct pages in training set	45
cache_control_no_store	Cache-Control forbids storing response	1
x_powered_by_php	Header reveals PHP backend	1

Table 7: Example combination of Enriched Header Value Encoding with Graph Structure

6.1.4 Header Semantic combined Graph Signals

We derive interpretable semantic indicators from header values: presence flags, token matches (e.g., vendor/CDN/runtime names), digit occurrence, RFC-compliant date detection, and value-length buckets; plus page-level lexical aggregates such as `header_count` and `header_entropy`. As structural context, we append `third_party` and `ref_by_pages`. We also retain the lexical file-extension one-hots (`ext_*`). Table 8 shows an examples.

Feature	Description	Value
Sample meta-information		
urls_id		846
page_url		https://example.nl
external_link		https://cloudflare.invalid/1
True_Label		1
fold		1
Example semantic encoding features with hit count 7		
report_to__present_y	report-to header present	1
report_to__has_digit	Value contains at least one digit	1
report_to__token_cloudflare	Contains token "cloudflare"	1
report_to__strlen	String length	28
header_count	Total number of response headers	19
header_entropy	Shannon entropy of header values	≈ 4.25
is_third_party	Cross-site inclusion	1

Table 8: Example combination of Enriched Header Semantic Encoding with Graph Structure

6.1.5 Class Weighting

To mitigate class imbalance, we apply per-fold class weights. For sklearn models we set $w_0 = 2.5 \cdot (n_1/n_0)$ and $w_1 = 1.0$, where n_0 and n_1 are the numbers of non-tracker and tracker samples in the training fold. For XGBoost we use the standard `scale_pos_weight` $= n_0/n_1$.

The factor 2.5 was chosen by a small pilot sweep $\{1.0, 1.5, 2.0, 2.5, 3.0\}$ on OOF validation. Under a recall floor of 0.70, 2.5 achieved the most stable precision–recall trade-off across models and both validation schemes, with reduced false positives and no loss in recall after global thresholding.

6.2 Training

We evaluate ten supervised classifiers with fixed hyperparameters (Table 9) selected from prior tuning runs. All classifiers are wrapped in `CalibratedClassifierCV` with sigmoid calibration using a 2-fold stratified split within the training fold.

Model	Key Parameters
Logistic Regression	class_weight=(w_0, w_1), max_iter=2000, random_state
Random Forest	n_estimators=300, class_weight=(w_0, w_1), n_jobs=-1, random_state
Extra Trees	n_estimators=300, class_weight=(w_0, w_1), n_jobs=-1, random_state
Gradient Boosting (sklearn)	random_state
HistGradientBoosting	random_state
AdaBoost	random_state
Decision Tree	class_weight=(w_0, w_1), random_state
Gaussian Naive Bayes	defaults
XGBoost	n_estimators=300, learning_rate=0.07, max_depth=6, subsample=0.9, colsample_bytree=0.9, reg_lambda=1.0, scale_pos_weight=$\frac{n_0}{n_1}$, eval_metric=`logloss`, n_jobs=-1, random_state
LightGBM	n_estimators=400, learning_rate=0.06, num_leaves=31, subsample=0.9, colsample_bytree=0.9, reg_lambda=1.0, class_weight=(w_0, w_1), n_jobs=-1, random_state

Table 9: Key fixed hyperparameters (exactly as used in our code)

Next, we use `StratifiedGroupKFold` ($n = 5$, shuffle, fixed seed), grouping both by `urls_id` (page-level grouping) and by `link_etld1` (link-level grouping). On each validation fold, we scan thresholds from 0.00 to 1.00 in steps of 0.01. Finally, we select the threshold that maximises F1-score subject to recall ≥ 0.7.

6.3 Results

Across four feature spaces, we evaluated ten classifiers using leak-free cross-validation with page-level grouping. Using F1 as the primary selection criterion, the best OOF F1s are **0.915** (DecisionTree, presence-only), 0.915 (XGBoost, value encoding), 0.900 (HistGradientBoosting, semantic), and 0.910 (XGBoost, graph-based). Under the stricter link grouping, performance drops (Table 11), which we treat as a conservative lower bound.

6.3.1 Protocol-base Baseline Model

Using HTTP header presence signal alone for static detection has weak performance under both upper and lower bound, with only 0.565 F1 and low precision ≈ 0.4 for the top models. Results are shown in Table 10.

Table 10: Top 3 presence-only model results grouped by page

Model	AUPRC	ROC-AUC	F1	Precision	Recall	MCC	LogLoss
AdaBoost	0.358	0.459	**0.565**	0.396	0.983	0.086	0.678
LogisticRegression	0.364	0.460	0.562	0.395	0.972	0.065	0.686
ExtraTrees	0.365	0.461	0.562	0.393	0.982	0.060	0.685

Table 11: Generalisation check for best model of presence-only

Setup	AUPRC	F1	Precision	Recall	MCC
Grouped by page	0.358	0.565	0.396	0.983	0.086
Grouped by link	0.354	0.569	0.400	0.985	0.112

6.3.2 Graph-based Baseline Model

The other baseline, graph signals independently achieve strong performance with F1 ≈ 0.9 and high precision in in-distribution evaluation. The lower bound drops to 0.7, reveals the practical potential of graph-based signals for unseen prediction 12:

Table 12: Top 3 graph-based model results grouped by page

Model	AUPRC	ROC-AUC	F1	Precision	Recall	MCC	LogLoss
RandomForest	0.970	0.945	**0.925**	0.979	0.877	0.885	0.169
ExtraTrees	0.968	0.943	0.924	0.960	0.892	0.881	0.177
HistGradientBoosting	0.967	0.948	0.929	0.984	0.880	0.892	0.208

Table 13: Generalisation check for best model of graph features

Setup	AUPRC	F1	Precision	Recall	MCC
Grouped by page	0.970	0.925	0.979	0.877	0.885
Grouped by link	0.741	0.689	0.544	0.941	0.456

6.3.3 Protocol- and Graph- based Combination with Header Value

With page-level grouping, XGBoost achieves the highest OOF F1 (0.915) for value encoding. Adding value information, precision increases subtly with similar recall as shown in Table 14:

Table 14: Top 3 value encoding model results grouped by page

Model	AUPRC	ROC-AUC	F1	Precision	Recall	MCC	LogLoss
XGBoost	0.962	0.938	**0.915**	0.980	0.858	0.871	0.224
LightGBM	0.960	0.938	0.915	0.985	0.854	0.872	0.220
DecisionTree	0.957	0.936	0.911	0.986	0.848	0.868	0.241

Table 15: Generalisation check for best model of value encoding

Setup	AUPRC	F1	Precision	Recall	MCC
Grouped by page	0.962	0.915	0.980	0.858	0.871
Grouped by link	0.598	0.626	0.465	0.958	0.315

6.3.4 Protocol- and Graph- based Combination with Header Semantic

Using F1 as the primary selection criterion, HistGradientBoosting attains the best OOF F1 (0.900) on semantic encoding. This strategy trades a bit of precision for higher recall 16:

Table 16: Top 3 semantic encoding model results grouped by page

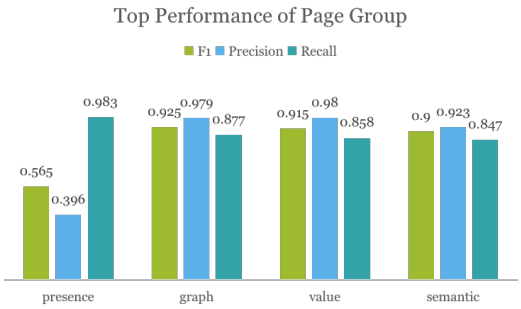
Model	AUPRC	ROC-AUC	F1	Precision	Recall	MCC	LogLoss
HistGradientBoosting	0.961	0.966	**0.900**	0.923	0.879	0.847	0.200
DecisionTree	0.906	0.938	0.899	0.943	0.859	0.849	0.265
XGBoost	0.959	0.964	0.898	0.931	0.868	0.846	0.231

Table 17: Generalisation check for best model of semantic encoding

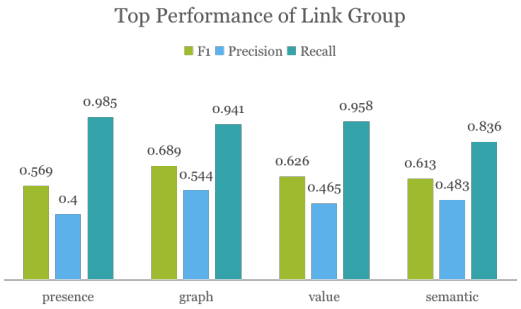
Setup	AUPRC	F1	Precision	Recall	MCC
Grouped by page	0.961	0.900	0.923	0.879	0.847
Grouped by link	0.568	0.613	0.483	0.836	0.333

6.3.5 Results Comparison

Through the comparison charts 7, we can clearly see the difference. Protocol-based signals alone achieve lower performance, whereas feature spaces with structural signals perform better. This confirms that HTTP response header signals are comparatively weak—consistent with the expectation outlined in Chapter 4.3.2—while structural signals emerge as a more reliable indicator.



(a) All Four Feature Spaces In-distribution Performance



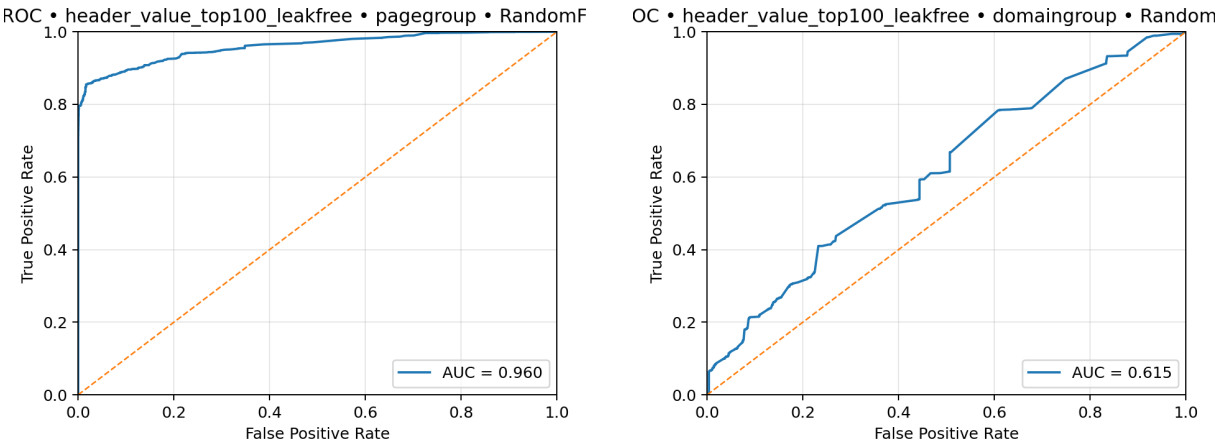
(b) All Four Feature Spaces Open-world Performance

Figure 7: Pie Chart Comparison of Top Results

6.4 ROC Analysis

We report out-of-fold ROC curves for the RandomForest classifier trained with the value-encoding feature space (Top-N *(header,value)* pairs plus file-extension one-hot, `ref_by_pages`,

and the third-party flag) with Platt calibration. Figure 8 contrasts performance between the page-group (in-distribution) and the link-group (unseen registrable domains) splits.



(a) Page-group (in-distribution)

(b) Link-group (unseen links)

Figure 8: ROC of RandomForest with value-encoding. Ranking quality is high in-distribution (AUC $= 0.960$) but drops when withholding registrable domains (AUC $= 0.615$), indicating a substantial open-world generalisation gap (ΔAUC ≈ 0.345).

Under the page-group split, the RandomForest with header value Encoding and graph structure combination exhibits strong OOF ranking quality (AUC $= 0.960$). The ROC curve rises steeply near the y–axis, indicating that a large fraction of positives can be recovered at very low false-positive rates. When entire $\texttt{eTLD+1}$ links are withheld (link-group), the curve becomes much flatter near the origin and shifts toward the diagonal, and AUC drops to 0.615 (ΔAUC ≈ 0.345). This pattern shows a loss of low-FPR recall on unseen links, consistent with the precision drop observed in the open-world setting. Overall, value-level protocol cues are effective in familiar contexts but require additional structural signals or hybrid designs to sustain precision when generalising to new links.

6.4.1 Error Analysis

In our tracker identification task, a True Positive (TP) refers to a request correctly predicted as a tracker by the model, True Negative (TN) is a request correctly predicted as a non-tracker, False Positive (FP) is a non-tracker incorrectly predicted as a tracker, and False Negative (FN) is a tracker incorrectly predicted as a non-tracker. From the confusion matrix results, we can see that TP and true negatives TN dominate the proportions, while the share of FP is larger than that of false negatives FN. Due to our threshold choice as described in chapter 5.5, more FP instances are included in the predicted results in order to avoid missed detections and to facilitate manual inspection.

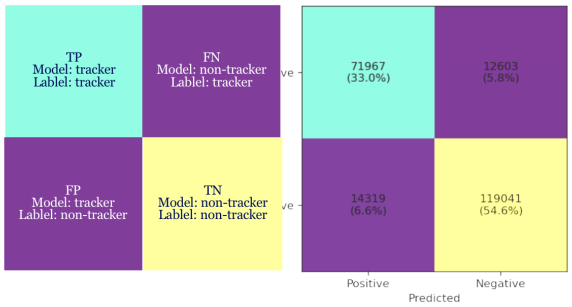


Figure 9: Confusion Matrix of Graph Topology (in-distribution)

Table 18 shows the most frequent links in FP and FN cases aggregated across models. These errors reveal that benign services (e.g., `postnet.nl`) can be misclassified as trackers, while certain popular trackers (e.g., `google-analytics.com`) are occasionally missed.

Table 18: Top FN/FP categories aggregated across models (by `eTLD+1`).

Case	`link_etld1`	Count
FN	`cdninstagram.com`	1401
FN	`wp.com`	1201
FN	`parastorage.com`	1157
FN	`google-analytics.com`	1033
FN	`jsdelivr.net`	986
FP	`postnet.nl`	1060
FP	`envoto.com`	1013
FP	`wa.me`	996
FP	`tickeswap.nl`	921
FP	`marktplaats.nl`	910

Table 19 reports confusion matrix rates for the top-performing models using encoding of header value and simple structural inclusion. While all models achieve a similar TP rate (≈ 0.334) and low FP rate (< 0.02), DecisionTree, XGBoost, and LightGBM provide the most balanced trade-off between false alarms and missed detections.

Table 19: Confusion matrix rates for top models (Feature: presence-only encoding). Rates are proportions over all samples.

Model	TP	FP	FN	TN
DecisionTree	0.3349	0.0094	0.0532	0.6025
XGBoost	0.3348	0.0097	0.0532	0.6023
LightGBM	0.3342	0.0095	0.0539	0.6024
HistGradientBoosting	0.3335	0.0105	0.0546	0.6014
GradientBoosting	0.3349	0.0199	0.0531	0.5920

6.5 Key Findings

Under the page-group (in-distribution) split, the graph baseline performs best (OOF F1 ≈ 0.925 with $P \approx 0.98$). The header value and semantic encodings with light structural signals (cross-site and popularity) follow (value: F1 ≈ 0.915, $P \approx 0.98$, $R \approx 0.86$; semantic: F1 ≈ 0.900, $P \approx 0.92$, $R \approx 0.85$). The header presence-only baseline is weakest (F1 ≈ 0.565, $P \approx 0.40$, $R \approx 0.98$), indicating an over-flagging tebdebct even on seen links.

Under the link-group (open-world) split, precision drops for all spaces while recall remains high; graph features remain strongest (F1 ≈ 0.689, $P \approx 0.544$, $R \approx 0.941$). The value and semantic combinations improve over presence-only (value: F1 ≈ 0.626, $P \approx 0.465$, $R \approx 0.958$; semantic: F1 ≈ 0.613, $P \approx 0.483$, $R \approx 0.836$) but still trail pure graph inclusion.

Overall, protocol headers used alone are unreliable, whereas value/semantic encoding plus simple structural context narrows the gap without surpassing graph-based performance.

When evaluated on unseen links, performance drops for all spaces, with precision decreasing much more than recall. The graph baseline remains strongest (F1 ≈ 0.689, $P \approx 0.544$, $R \approx 0.941$). The value and semantic combinations fall to F1 ≈ 0.626 ($P \approx 0.465$, $R \approx 0.958$) and F1 ≈ 0.613 ($P \approx 0.483$, $R \approx 0.836$), respectively. The presence baseline stays the weakest (F1 ≈ 0.569, $P \approx 0.400$, $R \approx 0.985$). Overall, recall remains high on unseen links, especially for graph and value at the cost of many false positives; presence header-only shows the poorest generalisation.

Error analysis shows that false positives often involve benign high-traffic services (e.g., CDNs, postal services) that mimic tracker-like inclusion patterns, whereas false negatives include well-known trackers that appear in less distinctive contexts.

7 Discussion

In this chapter, we discuss the implications of our results, the natural of false positives and methodological constraints. These analyses lead us a clear understanding on our approach limitation.

7.1 Insights of Model Performance under Boundaries

As proposed in 5.2, we evaluated two grouped cross-validation setups to capture the upper and lower bounds of realistic deployment: a page-group split that reflects **in-distribution** evaluation, and a link-group split that probes **open-world generalisation** where test links are unseen. The results reveal several insights.

In page-group, graph topology is the strongest single signal (F1$\approx$0.925, P$\approx$0.979, R$\approx$0.877). Combining header *value* with structural cues is close behind (F1$\approx$0.915, P$\approx$0.980, R$\approx$0.858). Presence-only HTTP header signals are weak (F1$= 0.565$) and mainly buy recall (R$= 0.983$) at the cost of precision (P$= 0.396$). When we switch to the link-group split, all methods drop—mostly because precision falls on unseen links. The best graph model keeps high recall (R$\approx$0.941) but lands at F1$\approx$0.689 (P$= 0.544$). Value and semantic features show a similar pattern (F1$= 0.626/0.613$, P$= 0.465/0.483$, R$= 0.958/0.836$). These results match practice: models learn many site-specific patterns, so recognising brand-new third-party links is hard.

Why this happens is intuitive. Structural position signals (PageRank, in/out-degree, k-core, betweenness) look at how a link sits in the graph, not at its name. They are naturally robust to renaming tricks and are harder for vendors to bypass. In contrast, HTTP response headers are easy to "decorate"—adding or changing fields without changing behaviour—which quickly hurts precision on unseen links. In other words, **HTTP header response signals alone are weak**, while **structural signals behave like strong rules** within a site or for well-represented patterns; in open-world conditions they remain excellent high-recall hints but need a safety net to control false positives (e.g., benign high-traffic CDNs or font providers that resemble trackers).

These observations lead to a simple trade-off and deployment plan. Protocol-level features are lightweight and fast, so they are well-suited for large-scale, close-world screening. Graph-based features generalise better to new trackers but are heavier—building and updating large graphs often needs GPU/high-memory resources. A practical pipeline is therefore cascaded: use protocol/value features as a first-stage filter for throughput, then escalate suspicious cases to the graph model for stronger open-world coverage; finally, send only high-risk or novel cases to manual review. In production, monitor header-field drift and refresh rules; maintain incremental graphs and approximate centralities to keep costs down. Overall, protocol features provide the scalable base, while graph topology provides robustness to naming tricks and the recall needed to surface unseen trackers. Combining them delivers a detector that aligns with both our measurements and real-world constraints.

7.2 Further Verification

To further discover the result, we did manual review of FPs under header value combination feature. We adopted the annotation protocol described in Iqbal et al.'s (2020) work. A request was labelled as a tracker if its domain belonged to an advertising or analytics network, engaged in cross-site user tracking, or served behavioral advertisements. Requests that served purely functional content such as content delivery networks, site-specific static assets, or messaging endpoints without tracking capability were labelled as non-tracker. Domains exhibiting both functional and tracking behaviours were labelled mixed, while cases with insufficient evidence were labelled undecidable [23].

Instead of selecting the top N ranked predictions for manual review, we adopted a probability-threshold approach (≥ 0.9) to gain insights into more diverse set of -+-s. This decision was motivated by the observation that top-N sampling can be dominated by repeated instances of the same external link. For example, among the top 50 false positive predictions by score, only 18 distinct domains were present, with the remainder belonging to WhatsApp contact pages.

The threshold (≥ 0.9) produced 181 false positive instances for manual inspection. Following Iqbal et al.'s (2020) four-way annotation scheme, each instance was manul labelled as tracker/, non-tracker, mixed, or undecidable. The majority (91.2%) were confirmed as genuine false positives, with typical categories including:

- WhatsApp short links used for merchant or customer service contact (`wa.me`), which do not involve tracking;
- Third-party static resource CDNs serving common frontend frameworks;
- Main JS files essential for site functionality rather than analytics or advertising;
- Other functional resources matching Iqbal et al.'s (2020) definition of non-tracking (e.g., site-specific static assets, first-party utility endpoints).

Only 10 cases (5.5%) were reclassified as true positives, including:

- Domains related to loan advertising (containing "lening"), where such sites often engage in marketing-oriented tracking;
- Embedded social media profiles (TikTok), which in AdGraph's taxonomy are typically treated as trackers due to cross-site identification and ad delivery capabilities.

No mixed-purpose cases were observed. The 6 undecidable instances (3.3%) corresponded to requests returning HTTP 404, where the resource content could not be verified.

The small fraction of reclassified true positives emphasise the value of manual verification for identifying trackers, which are missed by automated labelling. Examples include loan advertising domains that follow known cross-site identification and advertising behaviours described in prior taxonomies. The absence of mixed-purpose cases suggests tracker and non-tracker roles tend to be contextually consistent in our study.

These findings collectively indicate that a notable portion of false positives in strict evaluations stems from structural similarity between functional and tracking inclusions, rather than from random model errors. They also point to the inherent difficulty of third-party tracker discovery in open-world settings: without content- or behaviour-level validation, even highly discriminative protocol features risk over-flagging benign services.

7.3 Limitations

Our current labeling relies on exact host-level matching against a mixed-granularity blocklist (containing both eTLD+1 entries and specific subdomains). This design may miss trackers when only the registrable domain is listed (e.g., a request to `sub.tracker.com` would not match a blocklist entry `tracker.com`), and may over-specialise when subdomain-specific entries are present (other subdomains of the same registrable domain would remain unlabeled). Furthermore, the ground truth list itself (Disconnect and Ghostery) may not cover the full tracker ecosystem, potentially omitting long-tail or recently emerged entities.

Also, our approach is based on static crawling and HTML parsing, without dynamic execution in a real browser. This setup allows us to extract protocol-level features at scale, but it also means that runtime behaviours such as delayed requests, script-triggered loading, or user-driven interactions remain out of view. As a result, the dataset reflects only what is immediately exposed at load time, and does not capture behaviours that unfold during execution.

In addition, while our labelling is global and static at the host level, context-level labelling typically requires dynamic crawling with an instrumented browser. This enables observation of request triggers, DOM placement, timing, and blocking events under real execution, which are absent in our current setup.

This limitation affects both coverage and feature expressiveness. Domains that perform tracking and those that do not often appear in similar positions within the graph, and without additional behavioural signals, these patterns offer little separation. Structural features may thus yield false positives for popular functional CDNs or widely used libraries. Finally, model generalisation may degrade when moving from familiar-site conditions (page-group) to unseen-site scenarios (link-group), indicating that certain learned signals are site-specific rather than universally predictive.

8 Conclusion

This study asks: given an external link observed in a static crawl, how can we decide whether it is a tracker? We compared protocol-based HTTP response features with structure-based inclusion signals and evaluated them under two realistic grouping schemes with grouped cross-validation.

Under the page-group (in-distribution) split, structure-based graph features perform best, with out-of-fold F1 around 0.925, precision about 0.98, and recall around 0.88. Enriching protocol features with header values and simple structural cues (cross-site and referenced-by-page) narrows the gap: the value space reaches F1 about 0.915 with precision about 0.98 and recall about 0.86, while the semantic space attains F1 about 0.900 with precision about 0.923 and recall about 0.847. By contrast, the header presence-only baseline is weakest (F1 about 0.565, precision about 0.396, recall about 0.983), showing an over-flagging tendency even on seen hosts.

Under the link-group split that withholds registrable domains (our open-world proxy), all spaces degrade mainly through a drop in precision rather than recall. Graph features remain strongest (F1 about 0.689, precision about 0.544, recall about 0.941). The value and semantic spaces trail (F1 about 0.626 and 0.613; precision about 0.465 and 0.483; recall about 0.958 and 0.836). Presence-only stays high-recall and low-precision (F1 about 0.569, precision about 0.400, recall about 0.985).

These patterns indicate complementary strengths. Enriched protocol features enable highly precise identification in familiar settings and are attractive for conservative blocking in closed-world deployments. Structure-based features generalise better to unseen hosts by recovering a larger fraction of trackers, but they require additional precision control. Manual checks of high-confidence false positives also reveal host-level labelling noise (for example, functional CDNs or analytics endpoints), underscoring the need for careful ground-truth handling.

In sum, we provide:

- A systematic, static-data evaluation of protocol- and structure-based detection under realistic constraints, using grouped cross-validation at both the page and link levels.
- Evidence that enriched protocol features and inclusion-graph features offer complementary strengths.
- An empirical characterisation of the open-world generalisation gap, driven primarily by precision loss, and a case for evaluation schemes that simulate open-world conditions.

Summarise our findings and limitasions, we therefore bring the following future work:

- Two-threshold workflow with human-in-the-loop review to stabilise precision while keeping recall.
- Hybrid designs that fuse protocol value/semantic cues with graph topology (for example, stacking with calibrated decision rules).

- Integrating dynamic instrumentation to capture runtime signals invisible to static crawls (redirect chains, delayed or script-triggered requests, DNS/CNAME resolution).
- Broadening dataset diversity, accounting for temporal variation, and reducing labelling noise to obtain stronger supervision and more reliable estimates of generalisation.

Overall, these results advance our understanding of tracker detection in static measurement settings and lay a foundation for integrating protocol, structural, and future runtime signals into more robust hybrid detection systems.

References

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. *Annual ACM Conference on Computer and Communications Security (CCS)*, 2014.

[2] Alessandro Acquisti, Laura Brandimarte, and George Loewenstein. Privacy and human behavior in the age of information. *Science*, 2015.

[3] Abdul Haddi Amjad, Danial Saleem, Muhammad Ali Gulzar, Zubair Shafiq, and Fareed Zaffar. Trackersift: Untangling mixed tracking and functional web resources. *ACM*, 2021.

[4] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallarok, and Konrad Rieck. Dos and don'ts of machine learning in computer security. *USENIX*, 2022.

[5] Pouneh Nikkhah Bahrami, Umar Iqbal, and Zubair Shafiq. Fp-radar: Longitudinal measurement and early detection of browser fingerprinting. *Privacy Enhancing Technologies(PETS)*, 2022.

[6] Jason Bau, Jonathan Mayer, Hristo Paskov, and John C. Mitchell. A promising direction for web tracking countermeasures. *IEEE*, 2013.

[7] Sruti Bhagavatula, Christopher Dunn, Chris Kanich, Minaxi Gupta, and Brian Ziebart. Leveraging machine learning to improve unwanted resource filtering. *ACM Workshop on Artificial Intelligence and Security*, 2014.

[8] Christian Böttger, Nurullah Demir, Jan Hörnemann, Bhupendra Acharya, Thorsten Holz, Norbert Pohlmann, Matteo Große-Kampmann, and Tobias Urban. Understanding regional filter lists: Efficacy and impact. *Privacy Enhancing Technologies(PETS)*, 2025.

[9] Juan Miguel Carrascosa, Jakub Mikians, Ruben Cuevas, Vijay Erramilli, and Nikolaos Laoutaris. I always feel like somebody's watching me: measuring online behavioural advertising. *International Conference on emerging Networking EXperiments and Technologies(CoNEXT)*, 2015.

[10] Ismael Castell-Uroz, Josep Sole-Pareta, and Pere Barlet-Ros. Tracksign: Guided web tracking discovery. *IEEE*, 2021.

[11] Quan Chen, Panagiotis Ilia, Michalis Polychronakis, and Alexandros Kapravelos. Cookie swap party: Abusing first-party cookies forweb tracking.

[12] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world. *USENIX*, 2022.

[13] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. We value your privacy...now take some cookies: Measuring the gdpr's impact on web privacy. *Network and Distributed System Security (NDSS)*, 2019.

[14] Peter Eckersley. How unique is your web browser? *Privacy Enhancing Technologies(PETS)*, 2010.

[15] David Eliot and David Murakami Wood. Culling the floc: Market forces, regulatory regimes and google's (mis)steps on the path away from targeted advertising. *Information Polity*, 2022.

[16] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. *Annual ACM Conference on Computer and Communications Security (CCS)*, 2016.

[17] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W. Felten. Cookies that give you away: The surveillance implications of web tracking. *International Conference on World Wide Web*, 2015.

[18] Imane fouad, Nataliia Bielova, Arnaud Legout, and Natasa Sarafijanovic-Djukic. Missed by filter lists: Detecting unknown third-party trackers with invisible pixels. *Privacy Enhancing Technologies(PETS)*, 2020.

[19] Damien Geradin, Dimitrios Katsifis, and Theano Karanikioti. Google as a de facto privacy regulator: Analyzing chrome's removal of third-party cookies from an antitrust perspective. *SSRN Electronic Journal*, 2020.

[20] Alfonso Guarino, Delfina Malandrino, Rocco Zaccagnino, Federico Cozza, and Antonio Rapuano. On analyzing third-party tracking via machine learning. *International conference on information systems security and privacy(ICISSP)*, 2020.

[21] David Gugelmann, Markus Happe, Bernhard Ager, and Vincent Lenders. An automated approach for complementing ad blockers' blacklists. *Privacy Enhancing Technologies(PETS)*, 2015.

[22] Umar Iqbal, Charlie Wolfe, Charles Nguyen, Steven Englehardt, and Zubair Shafiq. Khaleesi: Breaker of advertising and tracking request chains. *USENIX*, 2022.

[23] Umar Iqbal, Peter Snyder† Shitong Zhu, Benjamin Livshits, Zhiyun Qian, and Zubair Shafiq. Adgraph: A graph-based approach to ad and tracker blocking. *IEEE*, 2020.

[24] Vasiliki Kalavri1, Jeremy Blackburn, Matteo Varvello, and Konstantina Papagiannaki. Like a pack of wolves: Community structure of web trackers. *International Conference on Passive and Active Network Measurement(PAM)*, 2016.

[25] Balachander Krishnamurthy and Craig E. Wills. Generating a privacy footprint on the internet. *ACM Internet Measurement Conference (IMC)*, 2006.

[26] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. *USENIX*, 2016.

[27] Tai-Ching Li, Huy Hang, Michalis Faloutsos, and Petros Efstathopoulos. Trackadvisor: Taking back browsing privacy from third-party trackers. *Passive and Active Measurement Conference(PAM)*, 2015.

[28] Timothy Libert. Exposing the hidden web: An analysis of third-party http requests on 1 million websites. *International Journal of Communication*, 2015.

[29] Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. *IEEE*, 2012.

[30] Durjoy Mistry, Jayonto Dutta Plabon, Bidita Sarkar Diba, Md Saddam Hossain Mukta, and M. F. Mridha. Federated learning-based architecture for personalized next emoji prediction for social media comments. *IEEE*, 2024.

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

[32] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P. Markatos. Cookie synchronization: Everything you always wanted to know but were afraid to ask. *International World Wide Web Conference(WWW)*, 2019.

[33] Wolf Rieder, Philip Raschke, and Thomas Cory. Beyond the request: Harnessing http response headers for cross-browser web tracke classification in an imbalanced setting. *Privacy Enhancing Technologies(PETS)*, 2025.

[34] David Rodriguez, Joseph A. Calandrino, Jose M. Del Alamo, and Norman Sadeh. Privacy settings of third-party libraries in android apps: A study of facebook sdks. *Privacy Enhancing Technologies(PETS)*, 2025.

[35] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and defending against third-party tracking on the web. *Network Systems Design and Implementation(NSDI)*, 2012.

[36] Jukka Ruohonen and Ville Leppänen. Invisible pixels are dead, long live invisible pixels! *Workshop on Privacy in the Electronic Society (WPES)*, 2018.

[37] Zhan Su, Rasmus Helles, Ali Al-Laith, Antti Veilahti, Akrati Saxena, and Jakob Grue Simonsen. Privacy lost in online education: Analysis of web tracking evolution. In *International Conference on Advanced Data Mining and Applications*, pages 440–455. Springer, 2023.

[38] Lu Xian, Song Mi Lee-Kan, Jane Im, and Florian Schaub. User-centric textual descriptions of privacy-enhancing technologies for ad tracking and analytics. *Privacy Enhancing Technologies(PETS)*, 2025.

[39] Zhonghao Yu, Sam Macbeth, Konark Modi, and Josep M. Pujol. Tracking the trackers. *International Conference on World Wide Web(WWW)*, 2016.