Scalable, Semi-supervised Mobile App Fingerprinting using LSH

Fatemeh Marzani, Samer Saleh
University of Twente
Enschede, The Netherlands
Email: f.marzani@utwente.nl, samerashrafwiliamhanasaleh@student.utwente.nl

***Abstract*—Application fingerprinting is essential for network management and security, enabling accurate traffic classification and the enforcement of Quality of Service (QoS) policies. In this work, we propose a scalable method for mobile application fingerprinting that leverages MinHash and Locality-Sensitive Hashing (LSH) to efficiently identify behavioral similarities in encrypted network traces. By restricting comparisons to high-similarity candidates, our approach significantly reduces computational complexity while preserving accuracy and enabling the detection of previously unseen applications. Evaluated on the ReCon dataset, the method achieves an average accuracy of 83% across app identification and unseen app detection tasks, with a reduction in comparison complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.**

Keywords: Mobile App Fingerprinting, Network Traffic Analysis, Encrypted Traffic Analysis, Scalability, LSH

I. Introduction

The rapid growth of mobile applications and the widespread use of encryption have posed significant challenges for network traffic analysis. In particular, identifying which applications generate observed traffic—commonly known as *application fingerprinting*—is a crucial capability for modern network operators. It supports a wide range of applications, including traffic shaping, intrusion detection, access control, and Quality-of-Service (QoS) enforcement [1]. However, traditional inspection-based methods are no longer effective, as most mobile traffic is encrypted and lacks distinguishing payload content [2, 3].

To address this, prior research has shifted toward behavioral and structural analysis of traffic metadata. Supervised learning methods, such as ML-NetLang [4], model application behavior by extracting symbolic representations of destination sequences and training classifiers for app identification. These methods achieve high accuracy but rely heavily on labeled datasets and cannot generalize to unseen applications [5, 6]. Unsupervised and semi-supervised approaches like FlowPrint [2] attempt to cluster behavior based on co-occurrence of destinations, but often incur substantial computational costs due to the need for pairwise similarity computation.

A recent line of work has explored hash-based representations for efficient traffic fingerprinting. For instance, LSIF [1] applies Locality-Sensitive Hashing (LSH) to device-level traffic for fast classification without relying on handcrafted features or training. However, such approaches remain device-centric and lack the granularity required to distinguish between applications running on the same device.

In this work[1], we present a lightweight and scalable method for *application-level* fingerprinting that overcomes these limitations. Our approach models each network trace as a symbolic sequence of destination addresses, converts them into sets of k-shingles to capture local behavioral patterns, and applies MinHash to produce compact, similarity-preserving signatures. These signatures serve as fingerprints of application behavior and are indexed using LSH to enable fast approximate matching. Unlike classifier-based methods, our technique does not require labeled training data and supports open-set identification, allowing the system to identify both known and previously unseen applications. This makes it particularly suitable for dynamic, large-scale network environments where efficiency and adaptability are essential.

The objective of this work is to evaluate the feasibility and scalability of MinHash–LSH for encrypted mobile application fingerprinting. Specifically, we assess whether Jaccard similarity is suitable for comparing execution traces, examine the accuracy of MinHash approximations, analyze the impact of key parameters such as the number of hash functions and band size, and demonstrate the effectiveness of the approach in recognizing both known applications and previously unseen ones. Unlike classifier-based methods, our technique requires no retraining, supports open-set identification, and is well-suited for dynamic, large-scale network environments where efficiency and adaptability are essential.

The main contributions of this paper are as follows:

- We introduce a scalable application-level fingerprinting method that combines MinHash with LSH for encrypted traffic analysis.
- We systematically evaluate the accuracy and parameter sensitivity of the approach using ReCon dataset [7, 8].
- We demonstrate that the method enables both recognition of known applications and detection of unseen ones, making it suitable for open-world deployments.

II. Background: Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH) is a probabilistic technique for efficient similarity search in large datasets. It is particularly effective for approximating Jaccard similarity between sets, enabling scalable matching without exhaustive pairwise comparisons [9].

[1]The source code is available at: https://gitlab.utwente.nl/s2888327/minhash-lsh.git

978-3-903176-75-1 ©2025 IFIP

A. From Traces to Sets: Shingling

Network traces can be abstracted as sequences of symbolic events, such as destination identifiers represented by tuples of (IP address, port), or TLS certificate fields. To convert these sequences into sets, we apply k-shingling, which extracts all contiguous substrings of length k from a trace.

Example. Let $k = 2$ and consider the following three traces:

- Trace A: `abcde`
- Trace B: `abfde`
- Trace C: `xyzpq`

We obtain the shingle sets:

$$S_A = \{\texttt{ab}, \texttt{bc}, \texttt{cd}, \texttt{de}\}$$
$$S_B = \{\texttt{ab}, \texttt{bf}, \texttt{fd}, \texttt{de}\}$$
$$S_C = \{\texttt{xy}, \texttt{yz}, \texttt{zp}, \texttt{pq}\}$$

Jaccard Similarity:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

$$J(A, B) = \frac{|\{\texttt{ab}, \texttt{de}\}|}{|\{\texttt{ab}, \texttt{bc}, \texttt{cd}, \texttt{de}, \texttt{bf}, \texttt{fd}\}|} = \frac{2}{6} \approx 0.33$$
$$J(A, C) = 0, \quad J(B, C) = 0$$

B. MinHash: Compact Similarity Estimation

To avoid storing full sets, we compute MinHash signatures—fixed-size vectors that approximate set similarity using multiple hash functions.

Example: Suppose the hash function $h_1(x)$ maps shingles to integers as follows:

$h_1(x)$: $\texttt{ab} = 1,\ \texttt{bc} = 3,\ \texttt{cd} = 4,\ \texttt{de} = 2,\ \texttt{bf} = 6,$ $\texttt{fd} = 7,\ \texttt{xy} = 8,\ \texttt{yz} = 9,\ \texttt{zp} = 10,\ \texttt{pq} = 11$

Then the minimum hash values for each trace are:

$$\min h_1(S_A) = \min\{1, 3, 4, 2\} = 1$$
$$\min h_1(S_B) = \min\{1, 6, 7, 2\} = 1$$
$$\min h_1(S_C) = \min\{8, 9, 10, 11\} = 8$$

Traces A and B share the same minimum hash under h_1 and are thus considered similar by this function. Trace C has a different minimum and is considered dissimilar.

Estimated similarity:

$$\hat{J}(A, B) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[h_i(A) = h_i(B)]$$

C. LSH Bucketing: Efficient Candidate Generation

MinHash signatures are divided into b bands of r rows each. A match in any band flags a candidate pair.

Example: Assume we use two hash functions, h_1 and h_2, to generate MinHash signatures for traces A, B, and C. After applying both hash functions to the shingle sets, we obtain:

$$\text{A} = [1, 1]$$
$$\text{B} = [1, 0]$$
$$\text{C} = [8, 6]$$

With $b = 2$ bands of $r = 1$ row:

- A and B share band 1 $(1) \rightarrow$ candidate pair.
- C differs in both bands $\rightarrow$ not a candidate.

D. Collision Probability

Given Jaccard similarity s, the probability of at least one band match:

$$P_{\text{candidate}}(s) = 1 - (1 - s^r)^b$$

This behaves as a soft threshold: high-similarity pairs are likely to collide, low-similarity pairs are filtered out. LSH enables scalable approximate matching by combining symbolic representation via k-shingling, compact MinHash signatures, and banded hashing for efficient candidate generation. This framework is well-suited for mobile application fingerprinting on encrypted traffic at scale.

III. Methodology

A. Dataset and Trace Generation

We evaluate our method using the ReCon dataset [7, 8], which comprises $7,665$ traffic traces collected from 512 popular Android applications across multiple versions and years. The dataset is generated by executing apps on physical devices using automated interaction scripts to simulate user behavior. For this study, we randomly select 100 applications and extract 10 traces per app, resulting in $1,000$ total traces.

Each trace is transformed into a symbolic behavioral sequence using the Trace Generator from ML-NetLang [4]. The transformation consists of the following steps:

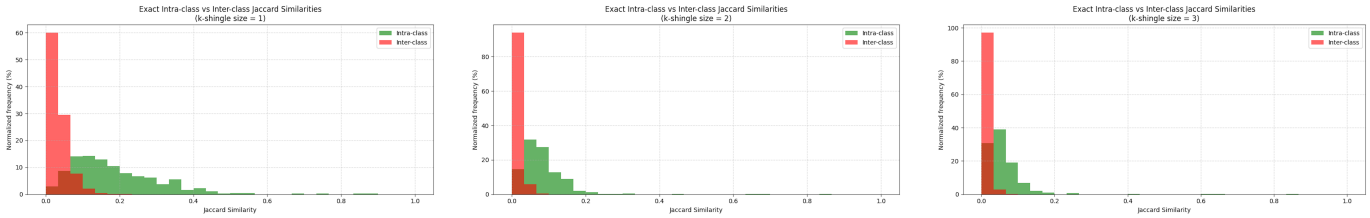
1) Extract flow-level information from packet captures, including timestamps, destination IPs, ports, and TLS certificates.
2) Sort all flows chronologically based on the timestamp of their first packet.
3) Map each unique destination (e.g., (IP, port) or TLS certificate) to a symbolic token S_j.
4) Replace flows with their corresponding symbols to produce an ordered sequence representing the application's behavioral trace.

These symbolic traces serve as compact, application-specific representations of network behavior and form the basis for fingerprinting.

B. Shingling and Similarity Analysis

To quantify the similarity between symbolic traces, we represent them as sets of k-shingles—contiguous subsequences of length k. We compute the actual Jaccard similarity $J(A, B)$ between all trace pairs using Equation 1.

To evaluate whether this metric separates traces of the same application (intra-class) from different ones (inter-class), we plot the normalized similarity distributions for varying k. Figure 1b shows that $k = 2$ produces the clearest separation, with the smallest overlap (0.227), indicating that 2-shingles capture behavioral structure effectively.



(a) 1-shingle (b) 2-shingle (c) 3-shingle

Fig. 1: Normalized frequency distributions of actual Jaccard similarity values for different k-shingle sizes. Lower overlap indicates better separation between same-app and different-app traces.

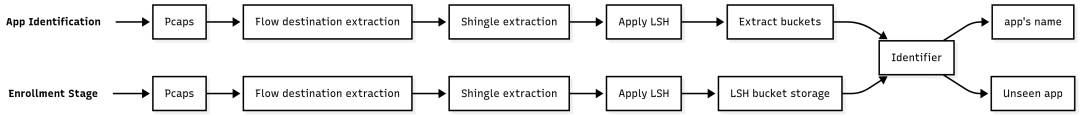


Fig. 2: System overview of the proposed mobile application fingerprinting pipeline using MinHash and LSH.

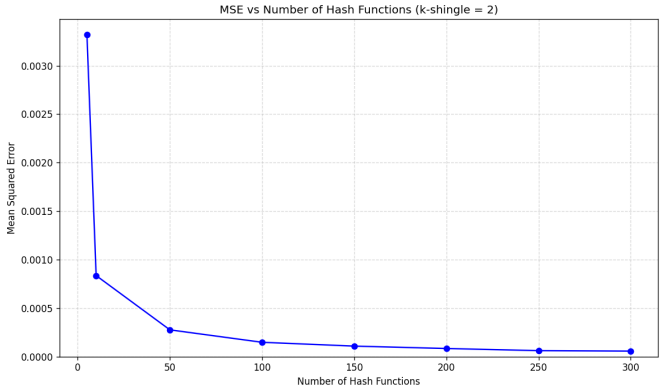


Fig. 3: MSE across different numbers of hash functions after setting the number of shingles to 2.

C. MinHash Approximation

To reduce the cost of computing exact Jaccard similarities, we employ MinHash to approximate set similarity. Each trace is hashed using m independent hash functions, and the minimum value under each function forms a signature vector. The estimated Jaccard similarity is given by:

$$\hat{J}(A,B) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}[h_i(A) = h_i(B)] \tag{2}$$

We analyze the trade-off between estimation accuracy and computational overhead by computing the mean squared error (MSE) between J and $\hat{J}$ for different values of m. As shown in Figure 3, the error decreases with increasing m and stabilizes around $m = 150$, which we adopt for all experiments.

D. LSH-Based Fingerprinting System

Our fingerprinting pipeline has two main stages: enrollment and identification (Figure 2). **Enrollment:** Given a new trace, we generate a 2-shingle set, compute a MinHash signature, and insert it into b LSH buckets (bands). Each bucket acts as an index for fast retrieval of similar traces.

Identification: An incoming trace is processed identically. Its MinHash signature is compared against enrolled signatures within matching buckets. If a match is found, the trace is assigned to the corresponding application. If not, it is flagged as an unseen app.

This architecture enables scalable, semi-supervised fingerprinting without the need for training classifiers, while remaining robust to previously unseen behaviors.

IV. Evaluation and Results

We evaluate our MinHash and LSH-based fingerprinting framework through two key tasks: multi-class app identification and binary unseen app detection. All evaluations use $1,000$ symbolic traces (10 per app) from 100 Android applications in the ReCon dataset [7, 8], transformed into 2-shingles and encoded into 150-dimensional MinHash signatures.

A. Metrics and LSH Parameters

We report accuracy, precision, recall, and F1-score metrics used in traffic fingerprinting literature [4, 2, 10, 11]. These metrics measure not only the classification quality of the system, but also its robustness under open-world assumptions.

MinHash signatures are divided into b bands of size $r = m/b$. Table I summarizes our configurations. As shown in [12], higher b improves sensitivity (recall) while lower b increases selectivity (precision).

TABLE I: Band size for different number of bands ($m = 150$)

Bands (b)	**Band Size** (r)
150	1
75	2
50	3
30	5

B. Application Identification (Seen Apps)

We evaluated the system's ability to identify traces from known apps using 5-fold stratified cross-validation. For each app, 8 traces are used for training and 2 for testing. Queries are matched using Jaccard similarity to LSH candidates, and majority voting determines the predicted label.

TABLE II: Classification performance on seen apps across different band configurations.

Bands (b)	Accuracy	Precision	Recall	F1 Score
150	**0.8527 ± 0.0428**	**0.8144 ± 0.0733**	**0.7752 ± 0.0851**	**0.7772 ± 0.0778**
75	0.7795 ± 0.0570	0.7343 ± 0.0957	0.6777 ± 0.1001	0.6858 ± 0.0948
50	0.3186 ± 0.0708	0.4153 ± 0.0827	0.2877 ± 0.0710	0.3287 ± 0.0725
30	0.0645 ± 0.0341	0.0954 ± 0.0534	0.0629 ± 0.0335	0.0736 ± 0.0399

C. Detection of Previously Unseen Apps

We evaluate unseen app detection by splitting the dataset into 80 "seen" apps for training/testing and 20 "unseen" apps for testing only. For seen apps, we use 8 traces for training and 2 for testing; all 10 traces of unseen apps are used in the test set, simulating real-world scenarios with new app appearances. We apply 5-fold cross-validation for stability. During inference, if a test trace does not exceed a Jaccard similarity threshold with any training trace, it is labeled ‘"unseen"; otherwise, we assign the most similar app using majority voting among LSH candidates. Table III reports results for thresholds 0.05 to 0.25

TABLE III: Unseen app detection results across different bands and similarity thresholds.

Threshold	Bands (b)	Accuracy	Precision	Recall	F1 Score
0.05	150	0.5783 ± 0.0067	0.9664 ± 0.0226	0.2500 ± 0.0141	0.3969 ± 0.0168
	75	0.6839 ± 0.0079	0.8813 ± 0.0274	0.4990 ± 0.0102	0.6369 ± 0.0070
	50	0.7083 ± 0.0107	0.6676 ± 0.0099	0.9470 ± 0.0068	0.7830 ± 0.0055
	30	0.5906 ± 0.0022	0.5757 ± 0.0013	1.0000 ± 0.0000	0.7307 ± 0.0011
0.10	**150**	**0.8011 ± 0.0111**	**0.8419 ± 0.0181**	**0.7910 ± 0.0128**	**0.8155 ± 0.0093**
	75	0.7883 ± 0.0116	0.8073 ± 0.0187	0.8140 ± 0.0116	0.8104 ± 0.0084
	50	0.6928 ± 0.0057	0.6482 ± 0.0049	0.9780 ± 0.0024	0.7796 ± 0.0029
	30	0.5906 ± 0.0022	0.5757 ± 0.0013	1.0000 ± 0.0000	0.7307 ± 0.0011
0.15	150	0.7433 ± 0.0080	0.6950 ± 0.0062	0.9590 ± 0.0037	0.8059 ± 0.0055
	75	0.7372 ± 0.0054	0.6894 ± 0.0040	0.9590 ± 0.0037	0.8022 ± 0.0038
	50	0.6544 ± 0.0028	0.6183 ± 0.0022	0.9880 ± 0.0040	0.7606 ± 0.0015
	30	0.5906 ± 0.0022	0.5757 ± 0.0013	1.0000 ± 0.0000	0.7307 ± 0.0011
0.20	150	0.6628 ± 0.0067	0.6237 ± 0.0045	0.9910 ± 0.0020	0.7656 ± 0.0038
	75	0.6628 ± 0.0067	0.6237 ± 0.0045	0.9910 ± 0.0020	0.7656 ± 0.0038
	50	0.6289 ± 0.0062	0.6000 ± 0.0040	0.9960 ± 0.0020	0.7489 ± 0.0032
	30	0.5906 ± 0.0022	0.5757 ± 0.0013	1.0000 ± 0.0000	0.7307 ± 0.0011
0.25	150	0.6161 ± 0.0054	0.5918 ± 0.0034	0.9960 ± 0.0020	0.7425 ± 0.0028
	75	0.6161 ± 0.0054	0.5918 ± 0.0034	0.9960 ± 0.0020	0.7425 ± 0.0028
	50	0.6117 ± 0.0067	0.5890 ± 0.0041	0.9960 ± 0.0020	0.7403 ± 0.0035
	30	0.5906 ± 0.0022	0.5757 ± 0.0013	1.0000 ± 0.0000	0.7307 ± 0.0011

The highest F1-score (81.55%) is achieved at threshold 0.10 with 150 bands. Lower thresholds favor precision, while higher thresholds inflate false positives. These results affirm that moderate similarity thresholds best balance accuracy, recall, and robustness to new behaviors. The system demonstrates reliable fingerprinting across known and unknown applications. A configuration of 150 bands and a Jaccard threshold of 0.10 consistently delivers the best performance, confirming the method's suitability for scalable and resilient mobile traffic analysis.

V. Related Work

Prior work on mobile traffic analysis spans supervised deep learning models, symbolic approaches, and hash-based fingerprinting. Mashnoor et al. [1] employ Nilsimsa-based LSH for IoT device identification, achieving 94% accuracy. Their LSIF-R method computes digest-based fingerprints tailored to device traffic but lacks support for unseen-device detection. Our method instead applies MinHash over tokenized symbolic sequences for scalable, app-level identification, including novel apps. Wang et al. [5] introduce App-Net, a hybrid CNN–biLSTM architecture that learns representations from TLS traffic sequences. It reports 93.2% accuracy on 80 apps but is restricted to closed-world classification and requires extensive labeled training data. By contrast, our LSH-based method is lightweight, training-free, and supports unknown-app detection. FlowPrint [2] generates fingerprints by clustering flows using destination IPs and temporal patterns, achieving an accuracy of 85.5% to detect unknown applications. Although effective, it relies on pairwise comparisons in all generated fingerprints, assigning labels based on the most similar matches, resulting in a computational complexity of $\mathcal{O}(n^2)$. NetLang [4] models traces as k-TSS automata and classifies with SVMs, achieving 97% accuracy on Android/iOS apps. It requires pairwise comparisons to generate numerical features and cannot detect unseen apps. Our semi-supervised method, while less accurate, supports new-app detection. Liu et al. [6] propose TransECA-Net, combining CNNs and Transformers with attention for general encrypted traffic classification, achieving $\sim$98% accuracy. Though highly accurate, it lacks support for new-class detection and is computationally intensive.

TABLE IV: Comparison of App Identification Methods

Method (Source)	Accuracy	Precision	Recall	F1 Score	Detection of Previously Unseen Apps	Complexity
Our method (LSH)	**85.27%**	**81.44%**	**77.52%**	**77.72%**	**Yes**	$\boldsymbol{\mathcal{O}(n \log n)}$
App-Net [5]	93.2%	–	–	91.2%	No	$\mathcal{O}(n^2)$
FlowPrint [2]	94.47%	94.7%	94.47%	94.58%	Yes	$\mathcal{O}(n^2)$
ML-NetLang [4]	97.0%	97.0%	97.0%	96.0%	No	$\mathcal{O}(n^2)$
TransECA-Net [6]	≈98.0%	≈98.0%	≈98.0%	≈98.0%	No	$\mathcal{O}(n^2)$

Our LSH pipeline offers a trade-off: while less accurate than deep models, it is simple, scalable and not requiring labels or retraining.

VI. Discussion

We evaluated the feasibility of using Locality-Sensitive Hashing (LSH) for fingerprinting mobile application behavior over symbolic execution traces. By applying 2-shingle tokenization, 150 MinHash functions, and a banding scheme of 150 bands (with band size 1), we achieved high separation between known and previously unseen applications. These results suggest that approximate similarity detection using MinHash provides a robust foundation for structural pattern extraction in encrypted mobile traffic.

While promising, several factors limit the generalizability and robustness of this approach.

Robustness to evasion. LSH-based fingerprints reflect surface-level structural properties of traffic sequences. As such, they remain vulnerable to adversarial noise injection or mimicry attacks. An application designed to interleave benign-looking flows or replay parts of another app's trace may bypass similarity thresholds. Although maintaining such evasion without compromising user experience is non-trivial,

this threat highlights the need for robustness testing under adversarial conditions.

Shared infrastructure effects. We observed that applications heavily reliant on shared third-party services (e.g., ad platforms, telemetry endpoints) produce traces with high inter-class similarity. These overlaps increase false positives and reduce discriminability. Techniques that incorporate frequency weighting or temporal segmentation may help de-emphasize generic flows and amplify app-specific behaviors.

Multi-app concurrency. Our evaluation assumes that each trace corresponds to a single foreground application. In practice, traces may represent blended activity from multiple apps or background services. Accurately decomposing such traces remains challenging without fine-grained context (e.g., foreground app labels, timestamps). Extending LSH-based fingerprinting to mixed-traffic scenarios will require traffic segmentation or session attribution mechanisms.

Scalability and deployment. A key advantage of our approach is its low computational cost. MinHash signatures are compact, and LSH lookup scales sublinearly with database size. Unlike supervised models, our method requires no retraining to accommodate new applications. This property enables deployment on constrained devices or in high-throughput monitoring settings. However, overall accuracy falls due to the probabilistic nature of LSH. Future work should aim to improve accuracy, for example by creating signatures over the most temporal correlated destination addresses instead of using all destination addresses, as suggested in FlowPrint [2].

VII. Conclusion and Future Work

We presented a scalable fingerprinting framework for mobile applications using MinHash signature generation, and LSH-based similarity detection. Our method transforms behavioral traces into 2-shingle sets, compresses them with 150 hash functions, and organizes them in 150 LSH bands for fast retrieval and comparison. Evaluated on a dataset of $1,000$ traces from 100 Android apps, the system achieved 85.37% accuracy for known applications, and 80.11% accuracy for detecting unknown apps. The approach demonstrates practical utility, supports encrypted traffic, and scales well without retraining. Future work includes exploring adaptive thresholds, improving resistance to generic or adversarial traffic, and extending support to concurrent app execution scenarios. Additionally, integrating frequency or temporal weighting into symbolic signatures may further enhance robustness and accuracy. Our results position LSH as a viable, interpretable baseline for efficient mobile app fingerprinting.

Acknowledgements

This research is funded by the Dutch Research Council (NWO) through the PERSPECTIEF Program P21-08 'XCARCITY'. We gratefully acknowledge their financial support.

References

[1] N. Mashnoor, J. Thom, A. Rouf, S. Sengupta, and B. Charyyev, "Locality sensitive hashing for network traffic fingerprinting," in *Proceedings of the IEEE International Conference on Communications (ICC)*, 2023.

[2] T. van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. Choffnes, M. van Steen, and A. Peter, "Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2020.

[3] E. Papadogiannaki and S. Ioannidis, "A survey on encrypted network traffic analysis applications, techniques, and countermeasures," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–42, 2021.

[4] F. Marzani, F. Ghassemi, Z. Sabahi-Kaviani, T. Van Ede, and M. Van Steen, "Mobile app fingerprinting through automata learning and machine learning," in *2023 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, 2023.

[5] X. Wang, S. Chen, and J. Su, "Automatic mobile app identification from encrypted traffic with hybrid neural networks," *IEEE Access*, vol. 8, pp. 182065–182077, Jan. 2020.

[6] Z. Liu, Y. Xie, Y. Luo, Y. Wang, and X. Ji, "Transecanet: A transformer-based model for encrypted traffic classification," *Applied Sciences*, vol. 15, no. 6, p. 2977, 2025.

[7] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. R. Choffnes, "Demo: Recon: Revealing and controlling PII leaks in mobile network traffic," in *Proc. the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion*, p. 117, 2016.

[8] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. R. Choffnes, and N. Vallina-Rodriguez, "Bug fixes, improvements, ... and privacy leaks - A longitudinal study of PII leaks across android app versions," in *Proc. 25th Annual Network and Distributed System Security Symposium*, 2018.

[9] A. Broder, "On the resemblance and containment of documents," in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pp. 21–29, 1997.

[10] L. Li, Z. Zhang, Y. Chen, and Y. Xiang, "Hsfl: Http-based similarity fingerprinting and localization of mobile applications," in *International Conference on Computational Science (ICCS)*, 2021.

[11] Y. Zhu, Z. Zhang, H. Hu, Y. Chen, and L. Zhang, "Appsniffer: Mobile application fingerprinting over vpn using encrypted traffic features," in *Proceedings of the Web Conference 2023 (WWW)*, 2023.

[12] J. Meira, C. Eiras-Franco, V. Bolón-Canedo, G. Marreiros, and A. Alonso-Betanzos, "Fast anomaly detection with locality-sensitive hashing and hyperparameter autotuning," *Information Sciences*, 6 2022.