# Predicting User Internet Activity from Encrypted Network Traffic Patterns

Nishk Patel
University of Illinois at Urbana Champaign
Champaign, United States
nishkdp2@illinois.edu

Ethan Mathew
University of Illinois at Urbana Champaign
Champaign, United States
ethmth2@illinois.edu

## Abstract

While encryption is now a standard feature in modern network communication, it does not fully conceal all aspects of user activity. In particular, even when packet contents are hidden, metadata such as packet size, timing, and direction remain visible to passive observers. This paper explores whether such observable characteristics can be used to infer user activity specifically, the websites a user is accessing despite encryption. We focus on the scenario of web browsing and investigate the feasibility of predicting the destination website using only encrypted traffic patterns. Our technique leverages timing and packet size features to classify browsing behavior without decrypting any content. Experimental results show that with just this limited information, we can identify the target website with 72% accuracy. This demonstrates that encrypted traffic still leaks meaningful information and that passive eavesdroppers may infer sensitive user activity, posing a continued risk to privacy.

Github Repository

**ACM Reference Format:**
Nishk Patel and Ethan Mathew. 2025. Predicting User Internet Activity from Encrypted Network Traffic Patterns. In *Proceedings of* . ACM, New York, NY, USA, 8 pages. https://doi.org/.

## 1 Introduction

As the internet becomes more deeply woven into our daily lives, understanding what can be inferred from encrypted network traffic is increasingly important. Although the widespread adoption of HTTPS has rendered the contents of network packets unreadable via SSL/TLS, it has not eliminated all avenues for observation. Passive eavesdroppers entities who monitor traffic without altering it can still access metadata such as packet size, timing, and frequency. This metadata, while seemingly benign, may leak significant information about user behavior and online activity.

One particularly relevant and evolving area of interest is the analysis of encrypted browser traffic. The websites a user visits, and the amount of time spent on them, can reveal substantial insights into their interests, behavior patterns, and even personal attributes a concept often referred to as "cyber-personality." Prior research has examined similar ideas in the context of mobile applications. For example, NetScope (Saltaformaggio et al., 2016) demonstrated that app-specific network behavior could be used to infer fine-grained user activities, despite encryption. Building on these insights, we aim to extend such analyses to browser-based traffic, continuing the exploration of how encrypted communication patterns can reveal user-level characteristics.

In this study, we investigate whether meaningful predictions can be made about browsing activity using encrypted traffic alone. Our focus is not on content or IP addresses, many of which are obscured by shared infrastructures like CDNs or NATs, but on observable characteristics such as packet size and timing. While such an approach limits the granularity of accessible information, it offers the advantage of being broadly applicable in anonymized or masked environments.

To evaluate this, we selected a set of eight websites reflecting a variety of usage types, ranging from common platforms frequently accessed by university students to more niche sites chosen based on personal interest. This diverse selection was designed to test the feasibility of distinguishing between different types of browsing activity using only network-level features.

Previous work has also touched on broader profiling approaches (eg. creating household-level user profiles based on traffic patterns) Acer et al., 2017, Gaowu et al., 2016, and Zhang et al., 2011, but our goal is to push this analysis to the individual level for web browsing specifically. This capability presents both opportunities and concerns - while it could enable more accurate content delivery or targeted services in public or shared network environments, it also raises serious privacy implications.

Through this work, we aim to demonstrate that even without decrypting content or accessing IP information, a passive observer can infer with substantial accuracy which websites are being visited underscoring the need to consider metadata as a vector for privacy leakage.


ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/.

## 2 Network Traffic Collection

To ensure the reproducibility and realism of our data collection process, we designed a framework that mimics everyday web browsing behavior as closely as possible, while enabling precise and structured data capture and analysis. This section outlines the tools used, the rationale behind design choices, and key observations from the collected data.

### 2.1 Browser Environment

For consistency and relevance, we used the Brave browser, a Chromium-based browser that shares network handling behavior with Google Chrome. This choice was informed by market share data indicating that Chromium-based browsers dominate global usage (StatCounter, 2024). Browser selection is an important consideration, as different browsers, such as Chrome and Safari, can yield different network traces due to variations in how they handle protocols, apply performance optimizations, and enforce privacy protections. By choosing Brave, we were able to emulate a realistic browsing experience consistent with the majority of users while also ensuring compatibility with tools for packet inspection.

### 2.2 Packet Capture Pipeline

To streamline data collection and minimize manual intervention, we developed an automated script that simulates functionality similar to that of Wireshark. Initially, data was captured using tcpdump, a command-line packet analyzer. However, to improve flexibility and parsing efficiency, we transitioned to using PyShark, a Python wrapper for the Wireshark packet capture library. This change allowed for granular control over packet data and enabled easier integration with other Python tools for further analysis.

PyShark offers several advantages over tcpdump:

- Wireshark Integration: Leverages tshark for detailed packet analysis in Python with minimal setup.
- Python Compatibility: Provides a high-level, programmatic environment for easy manipulation and storage of packet data.
- Efficiency: Allows real-time data capture with flexible filtering.

Specific Data Points Tracked. The following packet details are extracted by corresponding PyShark methods and logged by the script:

- Timestamp: Captures the exact time when the packet was sniffed.
  – `packet.sniff_time.strftime('Y-m-d H:M:S.f')`
- Source IP: The IP address from which the packet originated.
  – `packet.ip.src`
- Destination IP: The IP address to which the packet was sent.
  – `packet.ip.dst`
- Protocol: The highest layer protocol used in the packet (e.g., HTTP, TLS).
  – `packet.highest_layer`
- Length: The length of the packet in bytes.
  – `packet.length`

The script continuously listens for packets until manually stopped (`capture.sniff_continuously()`). This allows for persistent monitoring without needing to restart the capture process, offering flexibility for long-duration packet analysis. This pipeline's design also balances simplicity and efficiency, as it leverages `PyShark`'s internal use of subprocesses to handle the packet capture, without requiring complex multithreading or multiprocessing in the main script. This ensures that the packet capture process remains efficient while minimizing the complexity of the code itself.

### 2.3 Handling Port Multiplexing and Background Traffic

During the study, we generated two datasets, one that filtered traffic to include only Chrome-specific ports, and another that retained all observed traffic. While the port-restricted dataset might initially appear cleaner, we ultimately opted for the unrestricted dataset for several key reasons:

Port Multiplexing: Modern browsers, including Chrome and Brave, often multiplex multiple streams through single ports, making it challenging to isolate browser-specific traffic accurately. While our script effectively accounts for this by searching the task-list for the browser's PID and identifying all relevant ports, it's important to note that these ports can change during data collection, complicating the isolation process, or impair it entirely.

Authentic Background Noise: Realistic browsing sessions are typically accompanied by background traffic such as updates from other applications, system services, and secondary requests from embedded elements on websites. By retaining this "authentic noise," we better simulate a true user environment. This not only enhances the realism of the dataset but also presents a more challenging, and therefore more valuable, scenario for data analysis.

### 2.4 Device Configuration and IP Rotation

Data collection was performed on a MacBook Pro with an M1 chip, which periodically rotates its public IP address for privacy. To handle this, we maintained a dynamic log of host IP addresses used during browsing sessions. This tracking was critical for distinguishing incoming and outgoing packet flows during later stages of traffic analysis, a detail that will be discussed in later sections.

### 2.5 Website Selection and Use Case Coverage

Eight websites were selected to represent a range of use cases relevant to a university student, with a mix of multimedia, interactive, and text-heavy content. *Table 1* summarizes the

| Category | Website | Type of Activity | Dominant Behavior | Link |
|---|---|---|---|---|
| Music | Spotify Web Player | Streaming audio tracks and exploring curated or user-created playlists | Streaming, Browsing | https://www.spotify.com |
| Knowledge | Wikipedia | Reading articles and navigating through linked topics for information gathering | Download content, bursts | https://www.wikipedia.org |
| | ChatGPT | Engaging in interactive, conversational queries for knowledge or task assistance | Interactive | https://chat.openai.com |
| Social | Instagram | Scrolling through image and video content in a feed, often with autoplay | Download content, steady | https://www.instagram.com |
| | Reddit | Viewing video, image, text, and comment threads on reddit feeds | Download content, steady | https://www.reddit.com |
| Media Streaming | YouTube | Browsing and watching user-uploaded or recommended video content | Streaming, Browsing | https://www.youtube.com |
| Games | Online Tetris | Participating in fast-paced gameplay that updates in real-time with each move | Interactive | https://tetr.io |
| | Would You Rather | Selecting between binary choices and viewing aggregated responses | Interactive | https://wouldurather.io |

**Table 1.** Training Activities for Selected Websites with Dominant Network Behaviors

sites and their associated activity types. The selection aimed to mimic real-world academic and recreational browsing habits commonly seen in campus environments.

This mix of homogeneity and diversity allowed us to test whether our traffic-based classification technique could distinguish between different browsing behaviors with high accuracy. By incorporating both similarities and contrasts in traffic patterns, we were able to evaluate the model's ability to generalize across content types.

### 2.6 Traffic Pattern Observations

Two websites Tetris and Spotify showed particularly illustrative traffic patterns in *Fig 1, 2*, which help validate our methodology.

Tetris Gameplay: The network traffic during Tetris sessions exhibited an asymmetrical pattern. Client-to-server communication occurred sporadically, triggered by discrete game events, such as completing a line or losing a match. In contrast, server-to-client traffic was more consistent, driven by real-time updates such as opponent moves, leaderboard changes, and continuous music playback. These findings were corroborated by inspecting the browser's network tab, which revealed frequent streaming of audio and dynamic content.

Spotify Web Player: Sessions involving Spotify were marked by high-frequency data spikes, especially during initial playback or when browsing through songs and playlists. These spikes likely correspond to segments of audio being buffered or preloaded, which is consistent with Spotify's adaptive streaming strategy.

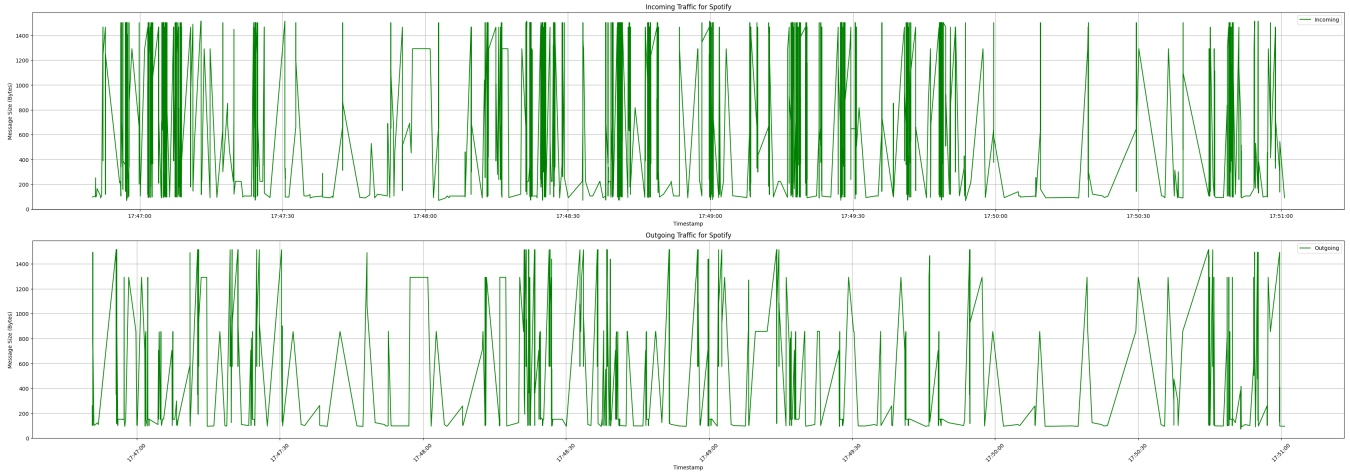


**Figure 1.** Spotify Web Player Network Traffic Data

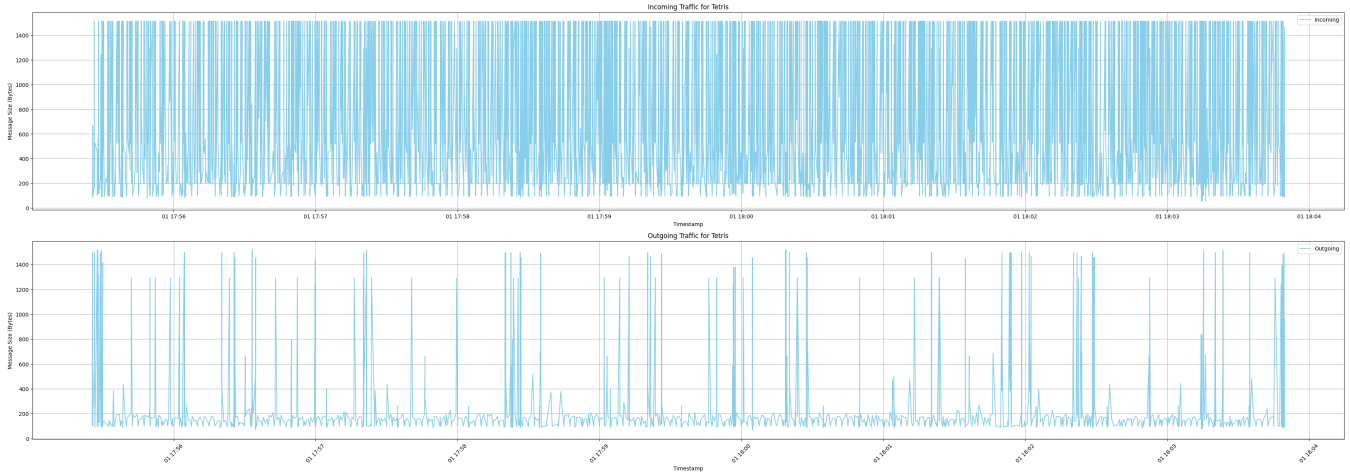


**Figure 2.** Tetris Network Traffic Data

# 3 Traffic Pattern Characterization

## 3.1 Feature Based Modeling

The raw packet capture data obtained during browsing sessions is inherently time-series in nature, consisting of a stream of timestamped packets associated with individual websites. While time-series modeling methods such as Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks are capable of learning sequential dependencies, they often suffer from poor interpretability.

Instead, we opted to aggregate time-series data into fixed-interval statistical summaries, allowing us to train interpretable models and analyze the predictive value of each feature. This transformation enables compatibility with classical machine learning algorithms and facilitates a clearer understanding of which aspects of encrypted traffic contribute most to inference accuracy.

**3.1.1 Temporal Window Strategy.** To extract meaningful information, we segmented the traffic into non-overlapping time windows of fixed length. For each time window of $T = 7$ seconds, we calculated summary statistics using all packets captured within that interval. Initially, we experimented with sliding windows (moving averages), but this resulted in data leakage across training and test splits due to overlapping intervals. As a result, we transitioned to disjoint intervals to preserve evaluation integrity.

We experimented with different window sizes from under 1 second up to 10 seconds. We observed that larger windows often improved model accuracy by averaging more traffic information, though this introduces a tradeoff longer windows require longer observation times, which may be impractical in certain passive surveillance contexts. A 7-second window was chosen as a compromise between information richness and eavesdropping feasibility.

### 3.2 Extracted Features

Each window was transformed into a vector of statistical features capturing packet size, timing, and directional behavior. The direction of traffic (incoming vs. outgoing) was inferred using a predefined list of host IP addresses associated with the data collection device. These directional distinctions are critical and can be determined even under IP anonymization, as packet directionality is observable at the link level.

The features are defined as follows:

**Average Packet Size (Outgoing)**

$$\text{avg\_packet\_size\_sent} = \frac{1}{N_{\text{out}}} \sum_{i=1}^{N_{\text{out}}} s_i^{(\text{out})}$$

where $s_i^{(\text{out})}$ is the size of the $i^{\text{th}}$ outgoing packet and $N_{\text{out}}$ is the total number of outgoing packets in the window.

**Average Packet Size (Incoming)**

$$\text{avg\_packet\_size\_recv} = \frac{1}{N_{\text{in}}} \sum_{j=1}^{N_{\text{in}}} s_j^{(\text{in})}$$

where $s_j^{(\text{in})}$ is the size of the $j^{\text{th}}$ incoming packet.

**Average Time Between Outgoing Packets**

$$\text{avg\_time\_between\_outgoing} = \frac{1}{N_{\text{out}} - 1} \sum_{i=2}^{N_{\text{out}}} \left( t_i^{(\text{out})} - t_{i-1}^{(\text{out})} \right)$$

where $t_i^{(\text{out})}$ is the timestamp of the $i^{\text{th}}$ outgoing packet.

**Average Time Between Incoming Packets**

$$\text{avg\_time\_between\_incoming} = \frac{1}{N_{\text{in}} - 1} \sum_{j=2}^{N_{\text{in}}} \left( t_j^{(\text{in})} - t_{j-1}^{(\text{in})} \right)$$

**Ratio of Outgoing to Incoming Packets**

$$\text{ratio\_outgoing\_to\_incoming} = \frac{N_{\text{out}}}{N_{\text{in}} + \epsilon}$$

A small value $\epsilon$ in theory would need to added to avoid division by zero, however, all our collected data had incoming packets.

**Number of Incoming Packets**

$$\text{number\_incoming} = N_{\text{in}}$$

**Number of Outgoing Packets**

$$\text{number\_outgoing} = N_{\text{out}}$$

**Dataset Label**

The categorical response variable indicating which of the eight websites the data window corresponds to.

Originally, we also included standard deviations for packet sizes and inter-packet times, but these features were discarded after experimental results showed they contributed little to predictive performance.

### 3.3 Feature Correlation Analysis

To assess redundancy among features, we generated a heatmap of pairwise Pearson correlation coefficients. Most features were found to be relatively independent. The only notably correlated pair was the average time between incoming and outgoing packets, which is expected due to the interactive nature of web traffic. No pair exceeded a correlation coefficient of 0.75, and so all features were retained for modeling.

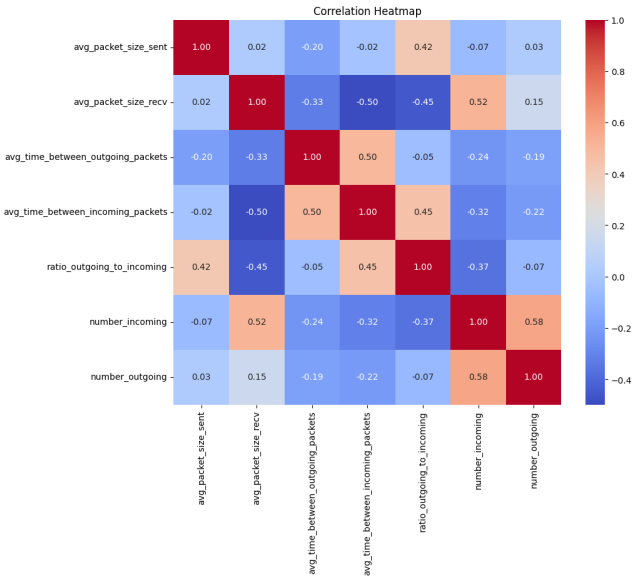


**Figure 3.** Features Correlation Heatmap

### 3.4 Exploratory Visualizations

We visualized feature distributions using boxplots (*Fig. 4*) grouped by website label, which revealed clear separability in several features (e.g., packet count and average sizes). Additionally, we applied Principal Component Analysis (PCA) to project the high-dimensional feature space into two dimensions. The resulting scatter plots indicated partial clustering by site, suggesting that meaningful structure exists in the feature space, a promising sign for downstream classification.

## 4 Modeling and Results

To model the website prediction problem, we utilized two well-established machine learning algorithms: Random Forest and K-Nearest Neighbors (KNN). Both models were selected for their robustness and ability to handle the feature set derived from network traffic data.

,
,



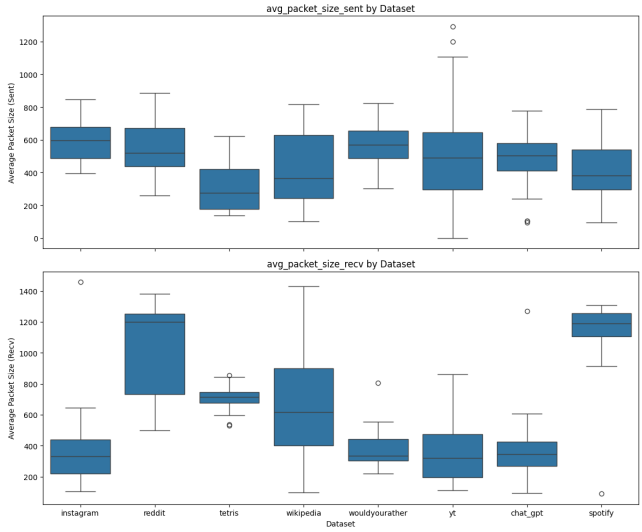


**Figure 4.** Receive and Send Packet by Dataset

### 4.1 Random Forest Classification

Random Forest was chosen due to its ability to handle both numerical and categorical data, and its robustness to overfitting due to the use of multiple decision trees. Additionally, Random Forest is capable of capturing complex relationships between features without extensive tuning.

Below is a single decision tree extracted from the generated Random Forest model, illustrating how individual decisions are made to determine the final classification outcome.

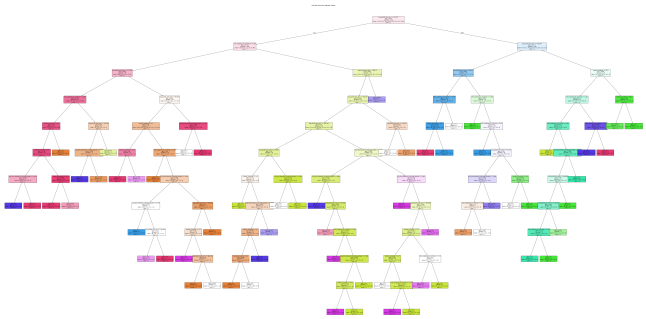

**Figure 5.** Decision Tree from Random Forest

To find the best hyperparameters, we performed a grid search over several candidate values for key hyperparameters. This search aimed to optimize the number of estimators, maximum depth of the trees, and parameters related to the minimum number of samples required to split or be a leaf node.

After performing the grid search, we evaluated the classification results, which are presented in the table below:

| Parameter | Values |
|---|---|
| `n_estimators` | {100, 200, 300, 400, 500} |
| `max_depth` | {None, 10, 20, 30, 40, 50} |
| `min_samples_split` | {2, 5, 10} |
| `min_samples_leaf` | {1, 2, 4} |

**Table 2.** Classification Report for Random Forest Classifier

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Chat GPT | 0.67 | 0.80 | 0.73 | 15 |
| Instagram | 0.57 | 0.80 | 0.67 | 15 |
| Reddit | 0.67 | 0.91 | 0.77 | 11 |
| Spotify | 0.83 | 0.71 | 0.77 | 7 |
| Tetris | 0.86 | 0.86 | 0.86 | 14 |
| Wikipedia | 0.33 | 0.10 | 0.15 | 10 |
| Would You Rather | 0.50 | 0.29 | 0.36 | 7 |
| YouTube | 0.74 | 0.67 | 0.70 | 21 |
| **Accuracy** | | | **0.68** | 100 |
| **Macro Avg** | 0.65 | 0.64 | 0.63 | 100 |
| **Weighted Avg** | 0.66 | 0.68 | 0.66 | 100 |

The overall accuracy achieved by the Random Forest model was 68%, with a macro average F1-Score of 0.63. This indicates the model's ability to correctly classify traffic from various websites, though the model showed mixed results across different classes. For example, Reddit and Tetris had relatively high precision and recall, while Wikipedia and Would You Rather showed poorer performance.

### 4.2 K-Nearest Neighbors

Next, we explored K-Nearest Neighbors (KNN), a simple yet effective model for classification. KNN works by identifying the majority class among the $k$ nearest neighbors of a given data point. It does not make assumptions about the underlying distribution of the data, which is beneficial for traffic data with potentially complex patterns.

We first tested the KNN algorithm without any dimensionality reduction and observed reasonable performance. However, the KNN model's performance could be influenced by the high dimensionality of the feature space. To explore whether we could reduce the complexity of the feature space and improve accuracy, we applied Principal Component Analysis (PCA).

### 4.3 KNN with PCA

After applying PCA, we trained the KNN model with varying values of $k$ (the number of nearest neighbors) and observed the resulting accuracy across different numbers of principal components.

A 3D plot of accuracy against the number of principal components and the number of neighbors showed that the model achieved a peak accuracy of 72% when using 6 principal components and 3 neighbors. This suggests that dimensionality

reduction via PCA significantly improved the KNN model's predictive ability by focusing on the most important features.

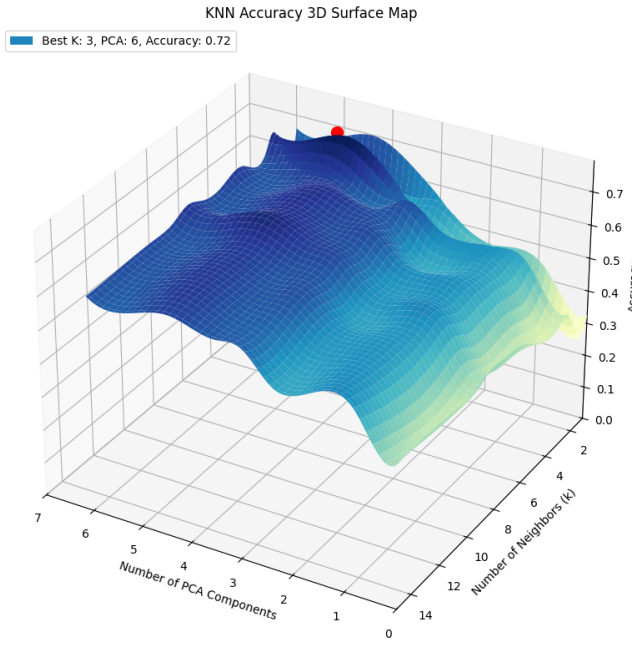


**Figure 6.** PCA & Neighbors to Accuracy

### 4.4 Discussion of Results

5.5 Discussion of Results Random Forest performed reasonably well, achieving 68% accuracy. It was able to distinguish between sites with relatively high certainty (e.g., Tetris, Spotify), but struggled with more subtle behaviors (e.g., Wikipedia, Would You Rather).

KNN with PCA showed an improvement, reaching 72% accuracy. The reduction in dimensionality helped the KNN model focus on the most informative features, leading to better generalization and accuracy.

## 5 Conclusion

This work demonstrates that even in the presence of fully encrypted traffic, meaningful inferences can be made about user behavior and browsing activity through passive observation of network metadata. By transforming time-series packet-level data into interpretable statistical features, we were able to predict the website being visited with up to 72% accuracy, solely based on observable traits such as packet size, timing, and direction without relying on IP addresses or payload content.

### 5.1 Contributions and Implications

Our findings contribute to a growing body of research showing that encryption alone does not equate to complete privacy. While encryption secures the contents of communication, it does not hide traffic patterns leaving room for inference attacks that can compromise user anonymity and behavioral privacy. This study provides a concrete example of how traffic fingerprinting can be applied to web browsing behavior, extending previous work that focused predominantly on mobile applications.

The approach also serves as a blueprint for scalable, interpretable traffic analysis systems that avoid the opaqueness of black-box models like deep neural networks. Instead, our method relies on summary statistics and traditional machine learning algorithms, allowing for transparency in both feature contribution and decision-making processes.

### 5.2 Future Work

There are several directions this research could be expanded:

- **Larger and More Diverse Datasets:** The study was limited in scope to eight websites due to time and computational constraints. Future work should explore a much broader set of websites, including dynamic and user-generated content platforms, to assess the scalability and robustness of this method.
- **Black-Box Models:** While our focus was on interpretable models, more complex models such as convolutional neural networks (CNNs) for time-series data or transformers could potentially capture deeper patterns. These models, however, would require significantly larger datasets and careful consideration of interpretability and overfitting.
- **Unseen Sites and Generalization:** One critical question is whether a model trained on known sites can generalize to new, unseen websites perhaps not identifying the exact site but categorizing the type of site (e.g., media streaming, social networking, news). Exploring unsupervised or semi-supervised learning could offer insights here.
- **Privacy-Preserving Countermeasures:** Given the implications for user privacy, future research could explore methods to obfuscate traffic patterns. These could include techniques like random packet delays, payload padding, or artificial noise injection. Implementing such defenses would involve trade-offs between usability, performance, and privacy, which must be carefully evaluated.

### 5.3 Final Thoughts

As web traffic grows increasingly encrypted and anonymous, this research underscores a subtle but significant gap in privacy metadata remains exposed, and with it, the potential for behavioral inference. Whether for improving network

diagnostics, enabling more targeted content delivery, or raising awareness about digital privacy, understanding traffic fingerprinting is a powerful and necessary step forward.

## References

[1] B. Saltaformaggio et al., "Eavesdropping on fine-grained user activities within smartphone apps over Encrypted Network Traffic," *USENIX*, https://www.usenix.org/conference/woot16/workshop-program/presentation/saltaformaggio (accessed May 3, 2025).

[2] U. G. Acer, F. Kawsar, and X. An, "Profiling and predicting user activity on a home network," *EUDL*, https://eudl.eu/doi/10.4108/eai.7-11-2017.2274043 (accessed May 3, 2025).

[3] G. Xie, "Reverse engineering user behaviors from network traffic," *eScholarship*, University of California, https://escholarship.org/uc/item/9kw1n7m9 (accessed May 3, 2025).

[4] "Deep learning on network traffic prediction: Recent advances, analysis, and future directions," *ACM Computing Surveys*, https://dl.acm.org/doi/10.1145/3703447 (accessed May 3, 2025).

[5] "Browser market share worldwide," *StatCounter Global Stats*, https://gs.statcounter.com/browser-market-share (accessed May 3, 2025).

[6] KimiNewt, "Kiminewt/pyshark: Python wrapper for tshark, allowing python packet parsing using Wireshark dissectors," *GitHub*, https://github.com/KimiNewt/pyshark (accessed May 3, 2025).

[7] "2.5. decomposing signals in components (matrix factorization problems)," *scikit*, https://scikit-learn.org/stable/modules/decomposition.html#pca (accessed May 3, 2025).

[8] "Kneighborsclassifier," *scikit*, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (accessed May 3, 2025).