# ML Project Proposal

## Team Introduction

Our team is named Exchange Team and is composed of six members: Ellen Kim (our team leader) (IES25574), Alexandre Roger Privat (IES25332), Bianca Dina Marine Muccini (IES25663), Alice Juliette Loustau (IES25359), Maya Oh Holmen (IES25496), Jennifer Jang (IES25640)

**Task Assignment**

- Ellen: Load the dataset. Implement the data pipeline, including preprocessing (handling corrupted or incomplete data, normalization). Separate raw data into X1, X2, and y arrays. Manage data Partitioning (training, validation, and testing subsets).
- Alexandre: Research and implement all categories of candidate features: Statistical, Burst, Timing, Directional Pattern, and Sequence-based. Ensure features are correctly formatted for all ML models.
- Jennifer: Build and train the k-Nearest Neighbors (k-NN) model. Document the methodology and results for the k-NN section.
- Alice: Build and train the Support Vector Machine (SVM) model. Document the methodology and results for the SVM section.
- Maya: Build and train the Tree Ensembles model to handle non-linear relationships. Document the methodology and results for the Tree Ensembles section.
- Bianca: Design and execute Hyperparameter Tuning strategies for all models (k-NN, SVM, Tree Ensembles) to optimize model robustness and performance.
- Shared by All: Contribute to proposal drafting, final report writing, and final presentation materials.

**Project Timeline**

Week 1 (Until 10/31) → Proposal Drafting and Submission. Exploratory Data Analysis.
Week 2 (Nov. 1 - Nov. 7) → Initial data separation. Start of statistical feature extraction. Full implementation of all candidate features (Statistical, Burst, Timing, Directional Pattern, Sequence-based).
Week 3 (Nov. 8 - Nov. 14) → Initiation of training and testing for k-NN, SVM, and early exploration of Tree Ensembles. Commencement of Hyperparameter Tuning to maximize model performance and robustness.
Week 4 (Nov. 15 - Nov. 21) → Continuation and finalization of training and testing for k-NN, SVM, and Tree Ensembles. Continuation and finalization of Hyperparameter Tuning across all chosen models. Establishing the best-performing models for the classification tasks.
Week 5 (Nov. 22 - Nov. 28) → Comprehensive analysis of final results. Comparison of achieved classification performance with the State-of-the-Art. Preparation of key graphs and metrics for the report.
Week 6 (Nov.28 - Dec 5) → Final consolidation of results, conclusions, and future work. Final Report Writing and submission. Preparation and rehearsal of the Final Presentation.

## Problem Definition: What is Website Fingerprinting?

Website Fingerprinting (WF) is a traffic analysis technique used to break the anonymity provided by networks like Tor. While Tor protects the content of your communication, WF exploits the patterns of the communication.

**1. The Core Target: Traffic Metadata**
WF does not inspect the content of messages but how they are sent. Each website has a characteristic way of loading resources, which creates a measurable “signature” in encrypted traffic.
The main metadata used are packet size and direction. On Tor, data is split into fixed 512-byte units called cells.
When we request a page (Outgoing), the traffic cells go from Client to Server (we encode this as a positive size, e.g., +512). When the website sends us data (Incoming), the traffic cells go from Server to Client (we encode this as a negative size, e.g., -512).
The sequence of these signed cell sizes creates a pattern like: [+512, +512, -512, -512, -512, +512, ...]
The time intervals between these cells, called packet timing, are also unique, reflecting factors like file sizes, resource dependencies, and network latency.

**2. The Two Phases of the Attack**
Phase 1: Profiling (training)
The attacker builds a library of known fingerprints:

1. Visit a set of target websites (e.g., Google, Facebook, WebMD) many times under controlled conditions and record raw traffic traces (PCAP files).
2. Extract statistical and sequential features from each trace (e.g., total packet count, burst sizes, exact packet-size sequence, timing features).
3. Feed those features into a machine-learning model so it learns to associate each traffic pattern with the correct website label.

Phase 2: Classification (attack)
With a trained model, the attacker can identify a victim’s browsing in real time:

1. Capture the victim’s traffic trace while they browse anonymously.
2. Extract the same metadata features from the victim’s trace.
3. Input these features to the trained model, which outputs the predicted website identity.

In essence, Website Fingerprinting is a sophisticated multi-class classification problem using network metadata as features. The machine learning model is the tool used to match the highly specific, unique "rhythm" of a website's loading process to its true identity.

**Closed world vs. open world**
In a closed world scenario, the attacker assumes that the user only visits from a list of known websites and every test trace has to belong to one of these websites. This makes it easier to trace websites and compare different model performances, but these assumptions are unrealistic, as a user could visit any website, not just from a set list.

In an open world scenario, the attacker assumes that the user can visit any website and is only interested in detecting whether or not the user has visited a monitored or unmonitored website, making it a binary classification problem. This is more realistic, but makes it harder to get a high accuracy.

## Data Analysis and Pipeline
The dataset used in this project consists of two pickle files: mon_standard.pkl, which contains 19,000 monitored traces, and unmon_standard10.pkl, which contains 10,000 unmonitored traces. Each trace represents one website visit, recording both the timing and direction of encrypted traffic. During

preprocessing, the data will be separated into different arrays: X1 will store timestamp sequences for each session, X2 will store the corresponding directional size values (positive for outgoing and negative for incoming packets), and y will contain the site labels for the monitored traces. This structure allows for consistency during data analysis when examining the timing, packet direction, and site identity across all sessions.

The preprocessing stage will focus on ensuring that all traces are complete, consistent, and ready for model training. Any incomplete or corrupted data will be removed, timestamps will be normalized to start at zero to capture the relative timing of the packets, and all packet sequences within each trace will be truncated or padded to a fixed length for consistency. The data will then be divided into training (~70%), validation (~15%), and testing (~15%) subsets to prevent the model from training on imbalanced data, which could potentially cause overfitting.

## Candidate Features (which feature will you explore and why?)

We will extract multiple categories of features from the raw timestamp (X1) and directional size (X2) sequences to capture different aspects of website traffic patterns.

**Statistical features** form the foundation of our feature set and include: total packet count (length of sequences), total number of incoming packets (negative values in X2), total number of outgoing packets (positive values in X2), total bytes transmitted in each direction (sum of absolute values), mean and standard deviation of packet sizes, and the ratio of incoming to outgoing traffic. These features capture the overall scale and balance of communication between client and server.

**Burst features** are designed to capture how websites load resources in clusters. A burst is defined as a sequence of consecutive packets traveling in the same direction. We will extract: the total number of bursts in each direction, maximum and average burst length (number of consecutive packets), maximum and average burst size (total bytes per burst), and burst duration (time span of each burst using X1). Websites with many images or scripts tend to produce distinct burst patterns that can serve as identifying signatures.

**Timing features** exploit the temporal characteristics visible in X1. We will compute: total session duration (difference between first and last timestamp), inter-packet delays (time differences between consecutive packets), and statistical distributions of these delays, including mean, median, variance, minimum, maximum, and various percentiles (25th, 50th, 75th, 90th). We will also examine the time to the first incoming packet and the time intervals between bursts, as different websites exhibit different loading speeds and resource dependency chains.

**Directional pattern features** capture the back-and-forth communication structure. These include: the number of direction changes (transitions from incoming to outgoing or vice versa), the average length of unidirectional sequences, the position and size of the first outgoing burst, and concentration measures that indicate whether traffic is evenly distributed or concentrated in certain time windows. Additionally, we will calculate the ratio of outgoing to incoming packets in different segments of the trace (beginning, middle, end) to capture how the communication pattern evolves over time.

**Sequence-based features** preserve ordering information that pure statistics lose. We will explore: n-grams of directional patterns (e.g., sequences like [+1, +1, -1, -1, +1]), the exact sequence of the first K packet sizes and directions (capturing the initial handshake and request pattern), cumulative byte counts at fixed time intervals, and possibly learned representations using simple recurrent features. These sequential features are crucial because the specific order in which a website sends resources creates a unique "rhythm" that distinguishes it from others.

These features are chosen because they comprehensively exploit the metadata visible in encrypted Tor traffic (packet sizes, directions, and timing) without requiring access to packet contents. By combining volume-based, temporal, and sequential characteristics, we aim to create a rich feature space that allows machine learning models to distinguish between the 95 monitored websites in closed-world scenarios and effectively detect monitored versus unmonitored traffic in open-world scenarios.

## Model Introduction (what models will you explore and why?)

**K-Nearest Neighbors (KNN)**

We will begin by exploring the K-Nearest Neighbors (KNN) model. KNN is a non-parametric, instance-based learning algorithm that makes predictions by comparing a new input to the most similar examples in the training set. This property makes it particularly suitable for WF, where the goal is to recognize patterns in encrypted traffic traces without explicitly modeling their underlying distribution. Since each website produces a distinct traffic “signature” based on the sequence and direction of Tor cells, KNN can effectively capture these similarities by using a suitable distance metric to compare sequences.

Another reason for choosing KNN is its interpretability and simplicity. It requires minimal assumptions about the data and can handle non-linear relationships between features like packet timing and direction. Additionally, KNN can serve as a strong benchmark before exploring more complex models. Its performance will help evaluate whether simple distance-based approaches can distinguish between monitored and unmonitored websites in the dataset. Given that WF relies on subtle timing and directional differences, testing KNN first provides valuable insights into how separable these patterns are using basic similarity measures.

**Support Vector Machine (SVM)**

The second algorithm we are going to explore is the Support Vector Machine (SVM).

The main goal of SVM is to draw an optimal boundary that clearly separates different groups of data points. Imagine our data points scattered on a field, where some belong to “Google,” some to “Facebook,” and others to “WebMD.” In such a case, SVM can be used to draw a line that best divides these groups.

Also, SVM doesn’t just draw any line. It draws the one that is farthest away from the nearest data points of each class. This maximizes the margin between the boundary and the closest data points. Maximizing this margin makes the separation more robust and reduces classification errors.

Furthermore, SVM handles complex, non-linear patterns effectively, which is particularly useful for analyzing network traffic features. This capability comes from the Kernel Trick. In many cases, data cannot be separated easily in 2D with a straight line. By mapping the data into a higher-dimensional space, SVM can separate it more easily.

To sum up, we are going to explore SVM because it is designed to find the clearest and most robust boundaries. This makes it a powerful model for classifying the intricate, hard-to-separate patterns of website traffic using our statistical features.

**Tree Ensembles**

We also plan to use tree ensembles to try to identify the websites. Since the underlying data is not guaranteed to be linearly separable, ensembles of decision trees can model complex, nonlinear relationships without requiring manual feature transformations. They naturally accommodate mixed feature types such as continuous variables (e.g. packet timings) and categorical variables (e.g. direction patterns), which our data has. If any data is missing, tree ensembles can also easily handle that during training.

In addition, tree ensembles are robust to noise and overfitting. Each tree is trained on a random subset of features or samples, introducing diversity that helps reduce variance. Techniques such as boosting or bagging also reduce variance and overfitting. Another advantage of this model is its interpretability. Feature importance can be readily extracted from the reduction in impurity (e.g. gini impurity or entropy gain) across splits, providing insight into which traffic characteristics contribute most to classification performance.

## Hyperparameter Tuning

Hyperparameter Tuning is an important component of our modeling approach, ensuring that our final classifiers achieve maximum performance and robustness. For each selected model, we will systematically explore its key configuration settings. Specifically, we will use techniques like grid search or random search combined with cross-validation on the validation set to optimize parameters. This tuning process is necessary to find the optimal boundary or feature weights that best distinguish between website traffic patterns, thereby maximizing our attack accuracy while mitigating potential overfitting.