Through the Kaleidoscope: Enhancing Generalizability for Machine Learning Approaches to Video Fingerprinting

Dakota Barnes and Nikolas Belle

Abstract

Video fingerprinting through the analysis of packet flow characteristics represents a coveted milestone for network engineers, while also posing an attractive target for cyber adversaries. Whether the motivation to accomplish this task has been driven by moral desires remains questionable, but recent advancements in machine learning have shed light on new potential fingerprinting approaches. The paper “Beauty and the Burst: Remote Identification of Encrypted Video Streams,” [5] argues how burst characteristics are unique to streamed videos and that examining the qualities of these bursts enables us to classify specific videos streamed on platforms such as Youtube, Vimeo, Netflix, and Amazon. According to the paper, since packet burst information can be retrieved from videos streamed in a network environment due to the current MPEG-DASH standards, remote attackers can easily fingerprint videos by training a machine learning model on these characteristics. In their study, they train a convolutional neural network (CNN) with burst-related features from data collected from streaming various videos across different platforms in a stable network environment. In this paper, we attempt to recreate this study utilizing the PINOT [1] infrastructure at the University of California, Santa Barbara, through the platforms netUnicorn [2] and Trustee [4]. Beyond this, we examine how the worrying results of “Beauty and the Burst: Remote Identification of Encrypted Video Streams” [5] might provide false hopes for potential attackers. The paper predicts that the model could struggle to generalize due to encoding mismatches between the streams the model was trained on and the streams the victim is accessing. In adaptive streaming, for example, the encoding changes dynamically as network conditions change. In an attempt to support this prediction, we experiment with a CNN trained on data gathered from different nodes provided to us through the PINOT [1] infrastructure and observe how the accuracy of the model changes when trained and tested on data collected from different network environments. Ensuring generalizability is incredibly important in machine learning as high F1 scores can provide false trust in a model that overfits to the data it was trained on. Despite the incredible potential, the networking industry has been cautious to accept machine learning solutions to learning problems in this domain due to the low threshold for failure. Solutions must be robust and generalizable, which are requirements that many proposed machine learning models fail to meet in the aspect of networking, despite having high F1 scores.

1 Introduction

The problem we are targeting in this paper is the suggestion that the classifier produced in “Beauty and the Burst: Remote Identification of Encrypted Video Streams” [5] may not generalize to different network environments. Examining the potential shortcomings of this model is important to the networking community because high F1 scores enable false trust in solutions that cannot be universally applied.

Looking at the paper, the primary problem we observe is that while the features of burst characteristics they chose to train the model on may be unique to every video on a streaming platform, they are also unique to network environments. Bit rate, for example, will not appear the same for a single video that is streamed using a university network infrastructure when compared to a cloud infrastructure like AWS or Azure. This is due to user capacity, congestion control, and frequency of traffic which varies across different environments and at different times of day.

In this paper, we show that while a fingerprinting model with high accuracy can be produced, it is important to analyze the decisions made by a black box model, like a CNN, to ensure the model is not overfitting to the data it is trained on. To do this, we use Trustee [4] as a post-hoc model interpretation tool. After reproducing the experiment detailed in the paper with relatively high accuracy, we analyze the decisions made by the model to predict whether it can be generalized to different network conditions. With this intuition, we retrain the model on data gathered from various environments, with the help of PINOT [1] and netUnicorn [2], and observe how the accuracy of the model changes, with hopes of less overfitting.

In this paper, we will begin by providing background information on the infrastructures and tools we utilize to conduct our experiments. Then, we will outline our initial technical approach to the learning problem we aim to

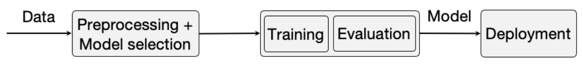


Figure 1: *Current Machine Learning Pipeline [3]*

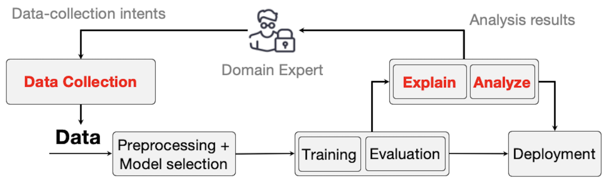


Figure 2: *Closed-loop Machine Learning Pipeline [3]*

address. After this, we discuss the iterative trial and error process we went through to produce better results with higher generalizability throughout the implementation of our plan. Finally, we will analyze the results of our experiments in relation to our learning problem.

2 Background and Motivation

Before exploring our learning problem, it is important to start from the beginning. Despite the incredible developments in the past years, machine learning has struggled to become a reliable method of addressing learning problems in the networking community. A variety of factors play a role in this, including the inability to easily augment data due to many packet contents and flow statistics being interdependent, shortcuts being easily found by the model, under-specification issues, and most importantly the difficulty of gathering diverse data that can be used to train a model with the ability to generalize.

2.1 Closed Loop Machine Learning

The current machine learning pipeline shown in Figure 1 involves the steps of data preparation and model selection, training, evaluation, and deployment.

While measurements such as accuracy and recall evaluate the model's performance, little can be done to analyze how a model makes its decisions when its complexity increases. Models with high explainability, such as decision trees, allow you to see decision rules that can highlight potential shortcuts the model makes, but black box models, such as random forests and neural networks, don't allow for any decision-making evaluation. To address circumstances of shortcut learning, under-specification, and spurious correlations which lead to models with high training accuracy but low ability to generalize to unseen data, it is necessary to adopt a closed-loop machine learning pipeline [3].

The closed-loop pipeline [3] displayed in Figure 2 includes the additional steps of explaining and analyzing a model to address overfitting issues and recollecting data to account for these issues. This loop is repeated until a model is obtained with high accuracy and the ability to generalize. To help close this loop, we utilized the tools PINOT [1], netUnicorn [2], and Trustee [4].

2.2 PINOT

PINOT [1] is a "programmable infrastructure for AI/ML for Networking (NetAI) research that enables network measurements from real-world infrastructure to create trustworthy data-based solutions" [1]. Making use of this infrastructure at the University of California, Santa Barbara, we were able to select from a collection of Raspberry Pi nodes spread across different locations on the university campus and in Isla Vista and send network traffic to these specific nodes. Since these nodes are connected to the public infrastructures in their respective locations, we can imitate a typical user and collect data that is representative of diverse network conditions.

2.3 NetUnicorn

To target these nodes and send network traffic to them, which in the case of our paper is streaming Vimeo videos, we made use of the powerful data collection platform, netUnicorn [2]. NetUnicorn [2] allows a user to set up data collection pipelines targeting specific nodes in a virtual or a physical infrastructure such as PINOT [1] and run experiments that iteratively collect network data from these pipelines. Pipelines are broken down into tasks, which in our case was watching specified Vimeo videos for 60 seconds each, which can be directed towards the Raspberry Pi nodes of our choosing to provide us with data from the same set of tasks collected in different environments. This emulated the changes in network conditions in different locations and at different times of day.

2.4 Trustee

Once training a model on the data collected through netUnicorn [2], it is essential to close the loop and analyze the decision-making process. To do this, we utilize Trustee [4], which extracts a decision tree explanation from the otherwise black box model we employ. The platform takes a training dataset and existing machine learning model as input and creates a "high-fidelity, easy-to-interpret decision tree and associated trust report as output" [4]. Implementing Trustee [4], we can trace the decision-making process of our black box model and theorize on if these decisions will generalize to data collected in other environments. With this intuition, we can easily rerun the data collection process using netUnicorn [2], with the potential modifications of targeting different nodes, altering the streaming time window, and types of videos streamed. We can also adapt our data pre-processing stage to the explanation of the decision tree and format the data differently or extract different features.

By utilizing the tools PINOT [1], netUnicorn [2], and Trustee [4], we hope to pinpoint areas where a relatively high-performing black box model might not generalize to potentially volatile network environments. From there we look toward how the data collection efforts and preprocessing stages can be modified to account for these changes across environments, closing the loop in the machine learning pipeline.

3 Technical Approach

Our initial approach was to try and replicate the technical approach in "Beauty and the Burst" [5] but on a much smaller scale. We initially were going to conduct our experiment using 10 YouTube videos, across 5 different Raspberry Pi nodes, to get a total of 50 video watches. We planned to capture the bits per second (BPS) up/down/all, packets per second (PPS) up/down/all, and packet length (PLEN) up/down/all. Then, we were going to try and copy the custom Convolutional Neural Network they created and see how our model accuracy compared to theirs, given that we were using multiple nodes to train our model versus the single node they used to train their model. We, however, severely underestimated what data we needed to collect to train a well performing model.

Once we started working on the data collection and preprocessing, we realized that we would need much more data. Additionally, we realized that it would be difficult to compare the accuracy of multiple nodes versus the "Beauty and the Burst" [5] paper's single node accuracy, especially if our accuracy was not close to their 99%.

Our experiment led us to two different types of captures. We represented the multiple node collection in Capture_4 which was 50 random nodes, for 5 Vimeo videos, with 60 seconds in streaming, giving us 250 total packet captures. We then represented a single node collection in Capture_6 which was 1 single node, for 5 Vimeo videos, with 60 seconds of streaming, giving us 250 packet captures.

We preprocessed the data with the same pcap_to_dataframe function, however, the single-node model was left with only 178/250 streams, and the multi-node model was left with 217 streams after data processing and normalization. We used graphs to display the down_bps for each video, with the same videos stacked on top of each other.

The plot for the 0.25-second intervals (interval_0.25 branch in the GitHub repository) can highlight the difference in streams between the videos much more visually than the (interval_5 branch in the GitHub repository, also in main branch). Additionally, we included the raw data then after processing, to show how we normalized the packet captures.

We ended up having two methods of storing our data, one for each model type. The concat datatype was each video data frame after the .concat operation, which stacks

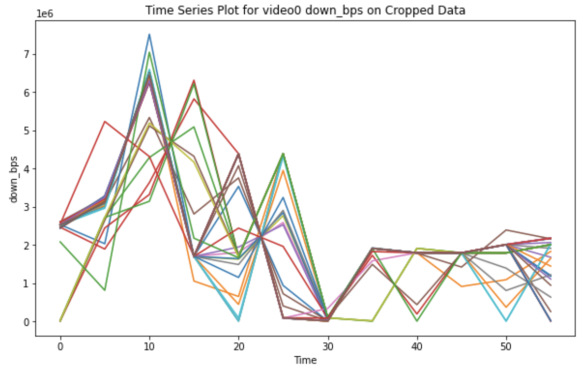


Figure 3: *Down bps for Single Node in 5 Second Intervals*

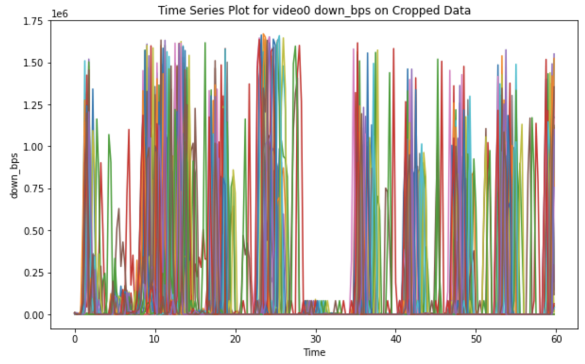


Figure 4: *Down bps for Single Node in 0.25 Second Intervals*

the data frames vertically. This data frame is used for the Random Forest Classifier, and the drawback is that it does not see time as a series, but as a feature. The second method was the numpy array, which was used for the Convolution Neural Network. Each stream was stored as a time series, where each interval had 6 features.

The main purpose of the concat method was for explainability because the numpy array could not be fed into the trustee [4]. Using the Random Forest Classifier for each interval separately, we found the time frame where the model could make the most accurate prediction and ran the trustee [4] on that time frame.

4 Implementation and Evaluation

4.1 Pipeline

The first step in the implementation process was creating a netUnicron [2] pipeline to collect the streaming data necessary to run our experiment. We found the most efficient way to do this was to create a dictionary at the beginning of the program, defining the video label to the link. Then, have a parameterized netUnicorn [2] pipeline with only around 10 lines of code, that could capture any number of videos, through any number of nodes, for any amount of time repeated any amount of time per node. The pipeline also makes sure to include the video label

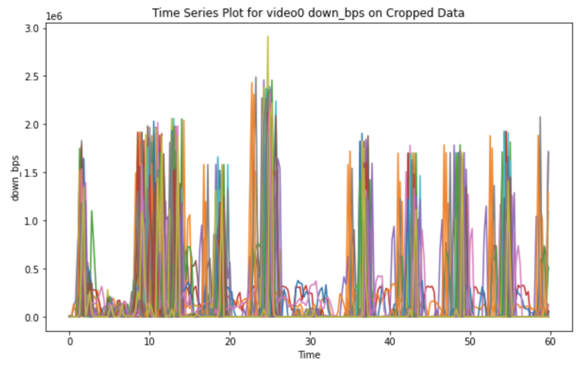


Figure 5: *Down bps for 20 Nodes in 0.25 Second Intervals*

in the saved packet capture files, to be extracted in the preprocessing step.

4.2 Data Collection

All of our packet captures we ran through netUnicorn [2] are labeled in dictionaries at the top of both files, with descriptions about the pipeline. We initially thought we were only going to need 10 YouTube videos, streamed through 10 nodes, for thirty seconds. This was labeled capture_0, and it had many problems. The first was that some of the videos we used were not even a full 60 seconds in time. Additionally, when we tried comparing video streams, they had no visual correlation to each other. Our first hypothesis was that the initial buffering of the YouTube video was lasting a large portion of the capture time and that ads from YouTube might be playing and messing up the packet capture data. To combat this, we believed we needed more data, which led to Capture_2 for 10 new YouTube videos, all above 5 minutes in length, through 50 nodes, and for a packet capture time of 300 seconds. The "Beauty and the Burst" [5] paper had YouTube packet captures for 4.5 minutes and cropped the streams to 3 minutes long. However, 50 nodes was slightly too much for the UCSB PINOT [1] infrastructure, and the videos we did capture did not have much correlation. This led us to switch gears from YouTube to Vimeo. The "Beauty and the Burst" [5] paper stated that they only needed to take 60 seconds from Vimeo for an accurate classifier, so we chose 5 new Vimeo videos and targeted 20 nodes. The resulting packet captures finally had data that was correlated enough to be seen by the human eye on a time graph, so we decided to pursue Vimeo and use 5 videos.

Then we realized that with the amount of data, our time restrictions, and our model complexity, our results were not going to be in the 99.9% accuracy region that the paper had, so we needed to determine a way to compare our results to theirs. This is what led to packet capture_8, and the concept idea behind our paper. Capture_8 used the same 5 Vimeo videos, and instead of multiple nodes, we streamed every single video from the same node, 50 times. Capture_8 represented how the "Beauty and the Burst" [5] collected their packet captures, because they only used one computer "node" to collect their data for the model.

4.3 Data Processing

To begin, each of the packet captures for a stream were placed on our snl-server, with the video number in the file name. We then created a function that would save the label name from the file path, iterate through each PCAP, and extract features using the Scapy function call. The features we chose were bps up/down, pps up/down, and plen up/down. Additionally, to remove some of the empty packets at the beginning of the stream, we started the time aggregation from the first packet capture that contained any of our feature data and cropped all the packet captures at 55 seconds. Once the packets were parsed, we decided to remove all of the empty captures, captures with less than 60 seconds of data, and captures that were missing features.

One of the most important parts of this preprocessing function is the interval variable, which is used to determine the time aggregation for the video. The problem is that we had three levels of features (each video, each time aggregation, and each feature), that we somehow had to fit into a two-level data frame. Our first solution was to try and place arrays in each cell of the data frame, however, the only way to do so would be to take the mean of all the features or choose one feature. This way each video stream would be a row in a large data frame, with the time interval as columns. However, we lost almost all of the critical feature information with this method. We then experimented with the dataframe.concat() function, which essentially could stack data frames vertically. This method had the potential to represent all of our three levels of features, but there was a cost. Each video stream was created into its own data frame with column features (bbs, pps, plen), and time aggregation as the rows, then each video data frame was stacked vertically. The problem with the .concat() function was that our data frame could no longer see relationships over time, and could only compare specific intervals of time. We had some success with the .concat() function, but still, we wanted to dive deeper.

Our research led us to numpy arrays, which could represent more than just two-dimensional data like the pandas data frames. With the numpy arrays, we could now represent each individual video stream as a time series, and each time interval had the corresponding 6 features.

4.4 Model Selection

Our first goal was to place our data into a simple classifier so that we could get a baseline of how useful our data was. Our model of choice was the Sklearn Random Forest

Classifier. Initially, we had our data aggregated into .25-second intervals, as the paper "Beauty and the Burst" did. However, with the concat() function, we realized that the time aggregations had to be much bigger for the model to have any success, because the timestamp was only used as a feature and was not looked at as a time series.

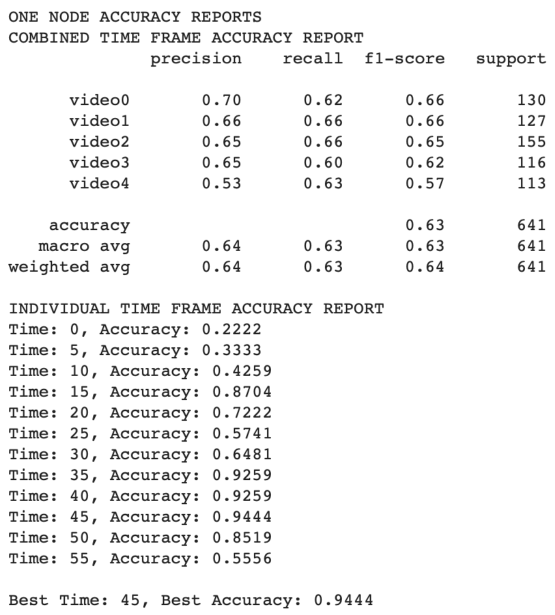
With the concat data frames and the 5-second aggregations, our model achieved 64% accuracy, which initially does not look very impressive. However, this 64% means that given any 5-second packet capture of the YouTube video, it could classify the video with 64% accuracy. We were testing our models with only the multi-node stream packet captures, which we suspected were going to be worse than the "Beauty and the Burst" [5] single-node collection, but we did not know by how much. This is the point in time where we created capture_8, which replicated the one-node setup of "Beauty and the Burst" [5]. When we ran capture_8 through the Random Forest Classifier, it obtained 68%. This was great because our prediction about multi-node vs single-node seemed to be correct, but also made us realize we had to figure out how to get our 68% accuracy to a point where it is comparable with the paper's 99% accuracy.

The time series issue became prevalent once again, and we decided to test the custom Convolutional Neural Networks from Keras. Our next big jump came from the realization of our ability to convert our data to numpy arrays as a time series. With our simple CNN, we obtained 90.9% test accuracy for the multi-node dataset, and 94% test accuracy for the single node dataset. Multi-node accuracy once again suffered in accuracy compared to single-node, which the "Beauty and the Burst" [5] paper created the model on, supporting our initial prediction. We felt confident with our results, but we still had one problem, which was explainability.

Unfortunately, Trustee [4] could not run on our multidimensional data from the Neural Network, so we ran Trustee [4] on the RandomForestClassifier. However, looking at the results, it felt wrong that the Trustee [4] decision tree represented every single time interval with only one decision tree. This forced us to get creative, and we asked the question: Which time interval has the highest chance of being classified correctly?

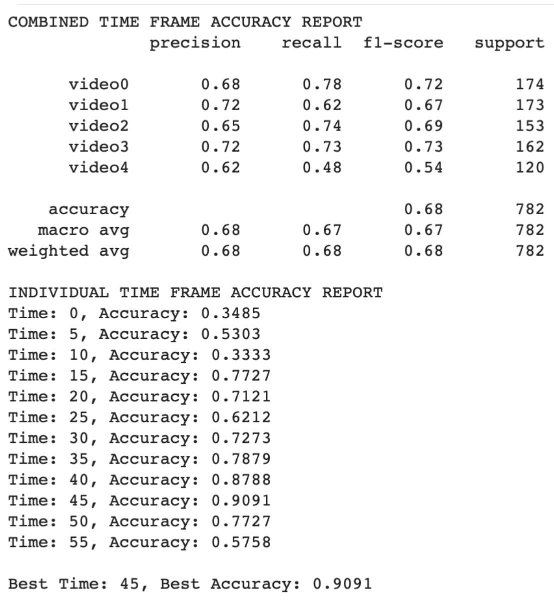
To answer this, we ran the classifier on each individual time period and found surprising results.

From the 0-15 second interval, the classification accuracy was in the 20%-40% range. Even more interestingly, for the multi-node and single-node dataset, the 45-50 second interval was where classification accuracy peaked at above 90%. This was great, but there was even more. For both the single node and multi-node Random Classifier 45-50 second time frame, the accuracy was within 0.01% of the overall accuracy of the corresponding CNN. Therefore, we were able to run Trustee [4] on the Random



```
ONE NODE ACCURACY REPORTS
COMBINED TIME FRAME ACCURACY REPORT
              precision    recall  f1-score   support

      video0       0.70      0.62      0.66       130
      video1       0.66      0.66      0.66       127
      video2       0.65      0.66      0.65       155
      video3       0.65      0.60      0.62       116
      video4       0.53      0.63      0.57       113

    accuracy                           0.63       641
   macro avg       0.64      0.63      0.63       641
weighted avg       0.64      0.63      0.64       641

INDIVIDUAL TIME FRAME ACCURACY REPORT
Time: 0, Accuracy: 0.2222
Time: 5, Accuracy: 0.3333
Time: 10, Accuracy: 0.4259
Time: 15, Accuracy: 0.8704
Time: 20, Accuracy: 0.7222
Time: 25, Accuracy: 0.5741
Time: 30, Accuracy: 0.6481
Time: 35, Accuracy: 0.9259
Time: 40, Accuracy: 0.9259
Time: 45, Accuracy: 0.9444
Time: 50, Accuracy: 0.8519
Time: 55, Accuracy: 0.5556

Best Time: 45, Best Accuracy: 0.9444
```

Figure 6: *Random Forest Classifier Individual Time Frame Accuracy Report for Single Node Data*



```
COMBINED TIME FRAME ACCURACY REPORT
              precision    recall  f1-score   support

      video0       0.68      0.78      0.72       174
      video1       0.72      0.62      0.67       173
      video2       0.65      0.74      0.69       153
      video3       0.72      0.73      0.73       162
      video4       0.62      0.48      0.54       120

    accuracy                           0.68       782
   macro avg       0.68      0.67      0.67       782
weighted avg       0.68      0.68      0.68       782

INDIVIDUAL TIME FRAME ACCURACY REPORT
Time: 0, Accuracy: 0.3485
Time: 5, Accuracy: 0.5303
Time: 10, Accuracy: 0.3333
Time: 15, Accuracy: 0.7727
Time: 20, Accuracy: 0.7121
Time: 25, Accuracy: 0.6212
Time: 30, Accuracy: 0.7273
Time: 35, Accuracy: 0.7879
Time: 40, Accuracy: 0.8788
Time: 45, Accuracy: 0.9091
Time: 50, Accuracy: 0.7727
Time: 55, Accuracy: 0.5758

Best Time: 45, Best Accuracy: 0.9091
```

Figure 7: *Random Forest Classifier Individual Time Frame Accuracy Report for Multiple Node Data*

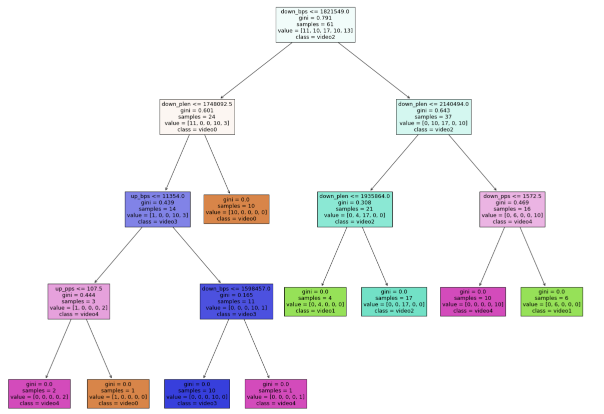

down_bps <= 1821549.0 gini = 0.791 samples = 61 value = [11, 10, 17, 10, 13] class = video2
down_plen <= 1748092.5 gini = 0.601 samples = 24 value = [11, 0, 0, 10, 3] class = video0
down_plen <= 2140494.0 gini = 0.643 samples = 37 value = [0, 10, 17, 0, 10] class = video2
up_bps <= 11354.0 gini = 0.439 samples = 14 value = [1, 0, 0, 10, 3] class = video3
gini = 0.0 samples = 10 value = [10, 0, 0, 0, 0] class = video0
down_plen <= 1935864.0 gini = 0.308 samples = 21 value = [0, 4, 17, 0, 0] class = video2
down_pps <= 1572.5 gini = 0.469 samples = 16 value = [0, 6, 0, 0, 10] class = video4
up_pps <= 107.5 gini = 0.444 samples = 3 value = [1, 0, 0, 0, 2] class = video4
down_bps <= 1598457.0 gini = 0.165 samples = 11 value = [0, 0, 0, 10, 1] class = video3
gini = 0.0 samples = 4 value = [0, 4, 0, 0, 0] class = video1
gini = 0.0 samples = 17 value = [0, 0, 17, 0, 0] class = video2
gini = 0.0 samples = 10 value = [0, 0, 0, 0, 10] class = video4
gini = 0.0 samples = 6 value = [0, 6, 0, 0, 0] class = video1
gini = 0.0 samples = 2 value = [0, 0, 0, 0, 2] class = video4
gini = 0.0 samples = 1 value = [1, 0, 0, 0, 0] class = video0
gini = 0.0 samples = 10 value = [0, 0, 0, 10, 0] class = video3
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 1] class = video4

Figure 8: *Trustee [4] Decision Tree for Single Node Random Forest Classifier on 45-50 Second Time Frame*

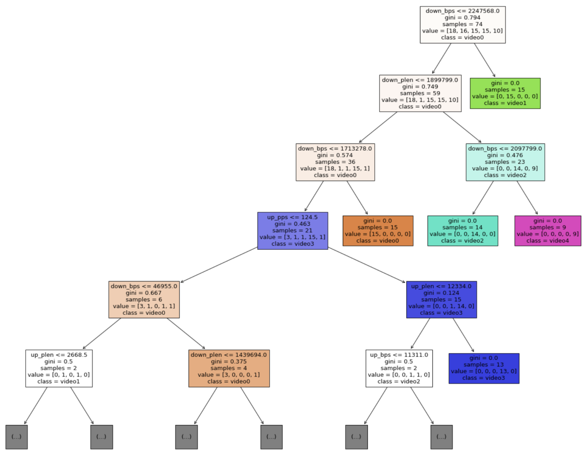

down_bps <= 2247568.0 gini = 0.794 samples = 74 value = [18, 16, 15, 15, 10] class = video0
down_plen <= 1899799.0 gini = 0.749 samples = 59 value = [18, 1, 15, 15, 10] class = video0
gini = 0.0 samples = 15 value = [0, 15, 0, 0, 0] class = video1
down_bps <= 1713278.0 gini = 0.574 samples = 36 value = [18, 1, 1, 15, 1] class = video0
down_bps <= 2097799.0 gini = 0.476 samples = 23 value = [0, 0, 14, 0, 9] class = video2
up_pps <= 124.5 gini = 0.463 samples = 21 value = [3, 1, 1, 15, 1] class = video3
gini = 0.0 samples = 15 value = [15, 0, 0, 0, 0] class = video0
gini = 0.0 samples = 14 value = [0, 0, 14, 0, 0] class = video2
gini = 0.0 samples = 9 value = [0, 0, 0, 0, 9] class = video4
down_bps <= 46955.0 gini = 0.667 samples = 6 value = [3, 1, 0, 1, 1] class = video0
up_plen <= 12334.0 gini = 0.124 samples = 15 value = [0, 0, 1, 14, 0] class = video3
up_plen <= 2668.5 gini = 0.5 samples = 2 value = [0, 1, 0, 1, 0] class = video1
down_plen <= 1439694.0 gini = 0.375 samples = 4 value = [3, 0, 0, 0, 1] class = video0
up_bps <= 11311.0 gini = 0.5 samples = 2 value = [0, 0, 1, 1, 0] class = video2
gini = 0.0 samples = 13 value = [0, 0, 0, 13, 0] class = video3
(...)
(...)
(...)
(...)
(...)
(...)

Figure 9: *Trustee [4] Decision Tree for Multiple Node Random Forest Classifier on 45-50 Second Time Frame*

Classifier 45-50 second time frame and see the choices that the model made. Most of the tree decisions were based on down bps, and down plen.

Trustee [4] was able to trace decisions that we believe were similar to what the neural network had learned. After reviewing them, it was clear that our single node dataset led to tighter decision boundaries than the multi node dataset, which leads to a model that is less generalizable to varying network conditions. Additionally, we could confirm that the black box model did not learn shortcuts in the training process.

5 Conclusion

When analyzing the results of our experiments, we see a series of learning points and stages that guided us towards creating a model that was not only comparable to the paper "Beauty and the Burst," [5] but also supported our hypothesis. Through a series of trial and error and reconfiguring our data collection strategy, data pre-processing steps, and model design, we were able to capture data that supported our claim that a model trained in one network environment would be very heavily susceptible to overfitting. Diverse network conditions are required in the data collection phase to train a model that is generalizable and can be adapted by other users. Beyond this, we emphasized the importance of utilizing a closed-loop machine learning pipeline when addressing learning problems in the network community as well as the overall field of technology.

References

[1] Beltiukov, R., Chandrasekaran, S., Gupta, A., and Willinger, W. Pinot: Programmable infrastructure for networking. In *Applied Networking Research Workshop (ANRW '23)* (New York, NY, USA, 2023), ACM, p. 3. (Cited on pages 1, 2, 3 and 4.)

[2] Beltiukov, R., Guo, W., Gupta, A., and Willinger, W. In search of netunicorn: A data-collection platform to develop generalizable ml models for network security problems. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)* (New York, NY, USA, 2023), ACM, p. 15. (Cited on pages 1, 2, 3 and 4.)

[3] Gupta, A. Lecture 1: Machine learning for networking. Lecture presented in CS 190N, 2023. (Cited on page 2.)

[4] Jacobs, A. S., Beltiukov, R., Willinger, W., Ferreira, R. A., Gupta, A., and Granville, L. Z. Ai/ml and network security: The emperor has no clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2022), CCS '22, Association for Computing Machinery. (Cited on pages 1, 2, 3, 5 and 6.)

[5] Schuster, R., Shmatikov, V., and Tromer, E. Beauty and the burst: Remote identification of encrypted video streams. 1357–1374. (Cited on pages 1, 3, 4, 5 and 6.)

A GitHub Repository

Access to the GitHub Repository with steps and documentation to recreate our experiments will be publicly available for use at the link mentioned above.