Inferring Quality of Experience of Video Streams with Low Level Network Data

Madison Long, Kaitlyn Tom, Dan Le

Abstract

Video streaming services such as Netflix, Youtube, and Twitch have grown exponentially over the past decade. With their growth, predicting Quality of Experience (QoE) has become increasingly more critical for developing adaptive bitrate (ABR) algorithms that ensure smooth playback and satisfactory user experience under changing network conditions. Network conditions have been shown to have a large impact on user-perceived QoE metrics like video resolution and rebuffering events. However, deploying machine learning for QoE prediction has been an ongoing challenge because of the complex nature of real-world network traffic. Existing models often rely on curated datasets, which leads to limitations in model generalizability. These challenges can lead to weak ABR algorithms that fail under real-world conditions and worsen user experience. We present a practical pipeline that integrates raw packet data with video session logs to predict QoE features such as video resolution and rebuffering events. Our system leverages supervised ML techniques to infer quality features from encrypted traffic patterns, bridging the gap between packet-level data and user-perceived video quality. Our results show that for some QoE features, such as SSIM, our model achieves high accuracy in prediction. This shows great promise in the feasibility of our approach for real-world network environments. We should note that many existing models tend to focus on multiple streaming platforms at once in an attempt to achieve generalizability. While we maintain an ML model should have a more fine grained focus on a specific platform (in our case, Puffer).

1 Introduction

With the growth of video streaming traffic, ensuring high Quality of Experience (QoE) for users has become a central challenge in modern networking. Adaptive Bitrate (ABR) algorithms, which are used by video streaming platforms such as Youtube and Netflix, must dynamically respond to changing network conditions in order to maintain smooth playback, high resolution, and minimal buffering. However, the increasing use of encryption protocols (e.g. HTTPS) on network traffic has restricted access to application-layer information, thus making predicting user perceived QoE in real time more difficult. The precise problem we address is how to infer video QoE metrics such as resolution and rebuffering events using only low-level network data (i.e. raw packets) and video session logs. Current approaches often rely on synthetic network conditions, which limits their generalizability. There is a large gap in practical methods for predicting QoE features from encrypted traffic in real world settings without having access to the application layer features of streaming platforms. In this paper, we show that it is possible to accurately infer QoE metrics from raw packet data using a data-driven pipeline. Our main contributions are:

- First, we design and implement a modular pipeline that takes raw packet capture files and video session logs as input, segments them into ten second sliding windows, and extracts meaningful features for QoE prediction.
- Second, we train supervised machine learning models to predict key QoE metrics such as video resolution and rebuffering events.
- Third, we validate our approach on real network traces and demonstrate its effectiveness across a variety of conditions.
- Fourth, we use an interpretability tool called TrusteeML, which uses decision trees to diagnose the decision making of our models.

Unlike prior work, which often assumes access to application-layer QoE labels or constructs synthetic datasets, our work operates in a realistic setting with encrypted traffic and noisy data. Our pipeline emphasizes practical deployment by working solely with network-layer features and generalizable video session metadata. The remainder of this paper is structured as follows: Section 2 reviews related work in QoE inference and ML-based traffic analysis. Section 3 describes our system design and data pipeline. Section 4 presents the implementation of our ML models, training setup, and TrusteeML explanations. Section 5 provides experimental results and evaluation. Section 6 concludes the paper and discusses limitations as well as potential extensions.

2 Background

Our work builds upon NetMicroscope [1], which developed machine learning models that extracted features from the network, transport, and application layers to infer two key metrics for QoE: startup delay and video resolution. NetMicroscope was deployed in real-world deployment environments with mixed traffic and was able to generalize across four streaming services: Netflix, Youtube, Amazon Prime Video, and Twitch. Features from the network and application layers were fed into random forest models for both regression (for startup delay) and classification (for video resolution). While NetMicroscope demonstrated promising results in calculating QoE metrics, it still had two major limitations. First, NetMicroscope only calculated two metrics for QoE, so we experimented with different QoE metrics such as rebuffering, average chunk size, and SSIM in addition to video resolution. Second, NetMicroscope did not provide any explanations for any of their models. Explainability of the models could provide important insight such as which features from the network data are most important in the model's decision making or if the model is taking advantage of any problematic patterns such as shortcuts or spurious correlations.

To address the first limitation of NetMicroscope, we trained and evaluated our models on a curated dataset based on Puffer [3], a streaming service that collects streaming session data from six television channels and is accessible to the public. Puffer was originally used to evaluate the performance of several different ABR algorithms that employed ML control schemes. A large-scale, publicly available dataset with detailed logs of real streaming sessions is provided by Puffer, with a total of 314,577 streaming sessions and over 1.9 million individual streams recorded. The dataset we obtained from Puffer recorded the average SSIM of each video session. Puffer had a broader range of network conditions and can generalize beyond specific services like Twitch or YouTube.

For the second limitation, machine learning models can be difficult to understand, especially when working with complex network data. Many ML models even outside of the networking field suffer from underspecification, where they rely on inductive biases instead of actually learning the essential structure of the data. Inductive biases can include:

- *shortcuts*: easy-to-learn patterns from the training data that do not necessarily correlate to the given task.
- *spurious correlations*: false associations that arise from coincidences in the training data.
- *failure on out of distribution samples*: samples that differ significantly from training data.

For our project, we decided to use TrusteeML [2], an interpretability tool that represents complex black-box models as simple, high fidelity decision trees. With TrusteeML, we can diagnose any inductive biases in the models we train.

3 Methodology

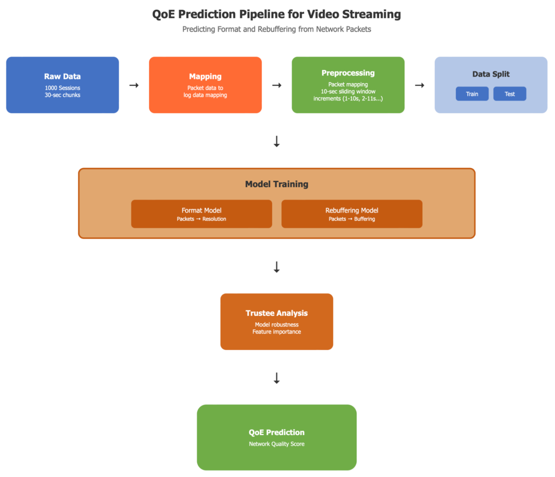


Figure 1: *Overview of the data prepocessing and model training pipeline.*

To accomplish our goal of using network data to predict QoE, the first and most important aspect of any model is its input data. Our dataset includes data from a range of users and multiple streaming sources, such as major broadcasters including NBC, CBS, and FOX. The dataset consists of two main components:

- Raw PCAP Traces: 1000 .pcap files. Each file represents a unique streaming session. We filtered these traces such that only PCAP's from a specific source IP were included. This aims to reduce background noise, filtering out PCAPS unrelated to the streaming session.
- Application Layer logs: During the streaming sessions, Puffer recorded 33,218 video transmission logs, containing data such as:
  - Sent and Acknowledged Timestamps
  - User and session identifiers
  - Video resolution (avg_format)
  - Chunk size
  - Structural Similarity index (SSIM)
  - Congestion window, RTT, in-flight bytes
  - Buffer size

Puffer's ability to produce detailed data logs allowed us to adopt a strategy similar to the netMicroscope paper [1]. This requires a complex preprocessing step in which raw PCAP data is sorted into ten second window bins based on the timestamp and user_id and mapped in an object to the corresponding Puffer log data from the same time period. Figure 1 shows the visual diagram for the pipeline for preprocessing the data and training our models. A full breakdown is shown below:

1. Logs are processed into a single dataframe.
2. PCAP features are extracted via TShark.
    - Tshark allows extraction of many features, including unix timestamp, source and destination IP addresses and packet size. The result is stored in a CSV file.
3. Log and PCAP timestamps are converted to human readable time.
4. Temporal Binning
    - Most streaming sessions in the dataset were just under a minute long, we broke them down further into ten second bins using a sliding window technique with a stride of one second. This creates overlapping time bins. Any bins that do not contain both packet and video data are discarded. This reduces noise.
5. Label Engineering
    - Each bin is labeled with several target QoE outcomes, such as resolution class, chunk size and rebuffering events. The resulting data is stored in a singular massive file, modeling_dataset.csv. Each row represents a time bin and its corresponding input features and targets.
6. Model Feeding
    - modeling_dataset.csv is used as an input for Random Forest Classifiers and regressors for the various targets.
7. TRUSTEE Explanations
    - Trustee was utilized for further analysis of the results.

4 Implementation

We developed a prediction pipeline to estimate Quality of Experience (QoE) metrics, specifically video format and chunk size, using only packet-level data. Our work was based on the Puffer dataset [3], from which we extracted and processed two key data sources: the `video_sent.1.log` file and the corresponding packet captures. We began by TShark to extract metadata from the packet captures, including source and destination IPs and timestamps. We then aligned these packet-level records with the video chunk metadata in the log files by matching on temporal proximity and identifying features such as session or channel information. After establishing this mapping, we segmented the combined data into overlapping 10-second windows using a sliding window approach. For instance, one chunk would span seconds 0 to 10, the next from 1 to 11, and so forth. This approach allowed us to retain temporal structure while increasing the number of training examples.

With the processed data, we trained two machine learning models: a random forest classifier to predict the video format as well as another classifier for rebuffering events and a random forest regressor to estimate chunk size as well as SSIM. We split our dataset into 80 percent for training and 20 percent for testing. The classification models were evaluated using F1 scores, accuracy, and classification reports, while the regression model was assessed using R^2 values, Mean Squared Error (MSE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE). Model performance metrics are detailed in the Evaluation section. To ensure our models were making decisions for meaningful reasons and not simply exploiting spurious patterns in the data, we used the Trustee library to generate TrustReport analyses. These reports provided interpretable explanations and helped validate that our models were not shortcutting.

The primary goal of our implementation was to demonstrate that QoE metrics could be predicted accurately using only packet-level data, without access to application-level or video content information. One major challenge we faced was aligning the video logs with packet-level data due to differences in granularity and data structure. We addressed this by carefully extracting packet timestamps with TShark and designing custom logic to map them to corresponding video chunks.

We implemented the entire pipeline in Python using a Jupyter Notebook hosted on Google Colab, which allowed us to prototype, visualize, and run everything in a cloud-based environment without needing to set up locally on everyone's computer. For preprocessing and file handling, we used the os library to download files. Pandas was used to load and manipulate structured data in table-like data frames, making it easy to group, filter, and process both the video and packet information. We relied on NumPy for fast numerical operations, especially for working with arrays and performing calculations on time and size data.

For model training and evaluation, we used several modules from the `scikit-learn` library. `RandomForestClassifier` was used to predict the video format and rebuffering events as a classification task, while `RandomForestRegressor` was used to estimate chunk size and SSIM as a regressor task. We used `train_test_split` to divide the dataset into training and testing sets, and `LabelEncoder` to convert categorical labels into a numerical format that the models could work with. For evaluation, we computed F1 scores, accuracy, and other classification metrics using `classification_report` and `accuracy_score`, and used `unique_labels` to extract and verify the class labels in the dataset. For the regression tasks, we evaluated performance using metrics such as MSE, RMSE, MAPE, R^2 score.

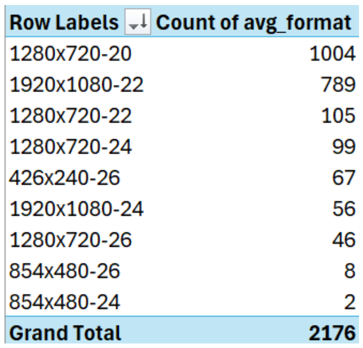
To understand and validate the model's reasoning, we used the Trustee library for interpretability. We applied the `ClassificationTrustee` and `RegressionTrustee` class to generate feature-based explanations of the classifiers' decisions and the regressors' decisions respectfully, and used `TrustReport` to summarize how the model made predictions and which features were most influential. This helped ensure that the model's behavior aligned with our expectations and wasn't relying on artifacts in the data.

5 Evaluation

In this section, we aim to answer the following key question: *Can specific QoE metrics be accurately predicted using only network metadata?* Our results show that machine learning models can successfully infer resolution and detect rebuffering events based solely on packet-level features. We should note all trials focused exclusively on a dataset generated by Puffer in an attempt to achieve generalizability with this specific domain. After completing the pipeline, we produced a dataset containing 2176 session bins. These were used to compute the following results:

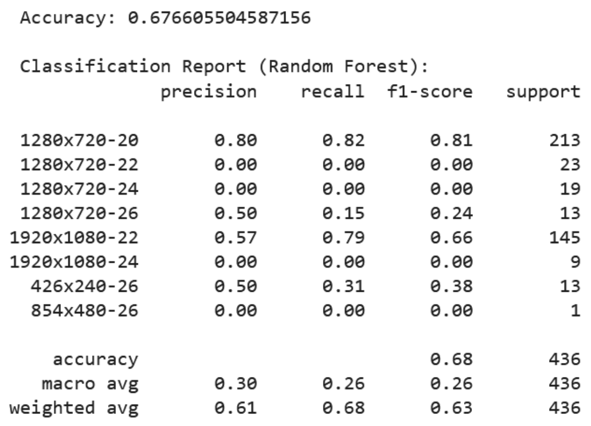
5.1 Video Resolution

We framed this as a multi-class classification task, (e.g, 1920x1080-22, 1280x720-24 etc). There is a finite number of fixed streaming qualities puffer will attempt to achieve making for an excellent classification problem. As shown in the distribution summary in Figure 2, the refined data was heavily skewed towards a few specific classes. This proved to be a major challenge as there were insufficient examples of rarer streaming formats in the session bins collected. Notice the extremely heavy skew towards popular formats such as 1920x1080-22 and 1280x720-20, while 854x480-26 only appeared in two total 10s bins.



Row Labels	Count of avg_format
1280x720-20	1004
1920x1080-22	789
1280x720-22	105
1280x720-24	99
426x240-26	67
1920x1080-24	56
1280x720-26	46
854x480-26	8
854x480-24	2
Grand Total	**2176**

Figure 2: *Class distribution of preprocessed resolution classes .*



```
Accuracy: 0.676605504587156

Classification Report (Random Forest):
              precision    recall  f1-score   support

 1280x720-20       0.80      0.82      0.81       213
 1280x720-22       0.00      0.00      0.00        23
 1280x720-24       0.00      0.00      0.00        19
 1280x720-26       0.50      0.15      0.24        13
1920x1080-22       0.57      0.79      0.66       145
1920x1080-24       0.00      0.00      0.00         9
  426x240-26       0.50      0.31      0.38        13
  854x480-26       0.00      0.00      0.00         1

    accuracy                           0.68       436
   macro avg       0.30      0.26      0.26       436
weighted avg       0.61      0.68      0.63       436
```

Figure 3: *Raw Model Output .*

The raw classifier achieved an f1 score of 0.6766 which is mediocre and not quite ideal. To combat this and provide some explanation, we incorporated TRUSTEE, which highlighted some of the issues with the dataset, notice how rarer classes have an f1 score of 0 while more common classes such as 1280x720-20 had an f1 score of 0.81. Implying the existence of shortcuts/bias toward overrepresented classes.

A higher score would imply the existence of a rule based structure in how packet features relate to resolution changes. Features such as throughput, packet count, and inter-packet delays may well be effective predictors of bitrate-adaptive decisions. An augmented study with a more robust dataset has the potential to highlight such a possibility if it exists.

5.2 Chunk Size

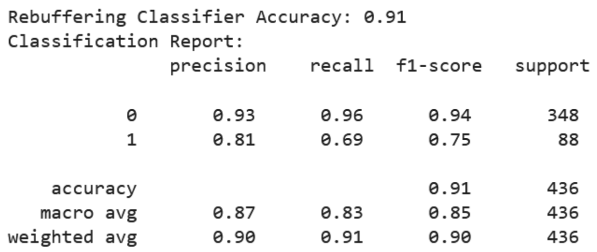
Predicting the average chunk size was framed as a regression problem, where we wanted to predict the size of each chunk, which is given in bytes. To evaluate the regression model, we used the Mean Absolute Percentage Error (MAPE), which is given by:

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

where n is the number of samples, $\hat{y}_i$ is the predicted value, and y_i is the actual value. MAPE gives the error as a percentage without units and scale, making it easy to interpret. The MAPE of the random forest model used to predict the average chunk size was 36.51%. This indicates that the model struggles to predict the average chunk size. There is most likely significant variation in chunk sizes, so the model poorly generalizes to the testing data. This suggests that the features used (packet count, byte size, packet size, and inter packet delay) are not enough for the model to accurately predict the average chunk size. Fortunately, average chunk size is relatively unimportant for calculating QoE.

5.3 Rebuffering

We define rebuffering as any 10s window in which fewer than 4 chunks are delivered, this functions on the base assumption that in Puffer, each chunk ideally carries 2.15s of video. Ideally, this means that 4.65 chunks would be needed for this window size. Should the number of chunks dip below 4, this implies the client was either paused (very rare) or stalled due to a buffer depletion. That is, a rebuffering event occurred, often to a lower resolution. However, as demonstrated in Figure 4 below, the data is highly imbalanced. Notice how only 6 positive results occurred in the dataset. This makes the impressive Rebuffering Classifier accuracy of 0.98, an invalid statistic.



```
Rebuffering Classifier Accuracy: 0.91
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.94       348
           1       0.81      0.69      0.75        88

    accuracy                           0.91       436
   macro avg       0.87      0.83      0.85       436
weighted avg       0.90      0.91      0.90       436
```

Figure 4: *Rebuffering Results*

The experiment could be validated if the window technique was modified to check for rebuffering since the beginning of the session. Future experiements could incorporate this improvement, as there is much rebuffering relevant data available in PCAPS, as rebuffering is typically preceded by:

- Reduced packet throughput, resulting in lower packet_count and total_bytes
- Increased inter packet delay variance, reflected in std_inter_packet_delay
- More sporadic and bursty packet arrivals, leading to 'jitter' and inconsistent delivery.

These behaviors disrupt the client's ability to fetch chunks at a consistent rate, resulting in playback pauses. We believe this may be a valid pursuit in future studies, but in our case, the data is insufficient to provide a valid trial.

5.4 SSIM

To assess SSIM, we first converted the SSIM values to decibel units. This is because our data only ranged from 0.88 to 0.99 out of the possible SSIM values of 0 - 1. After converting, we fitted a regressor model for our data as SSIM and SSIM decibel values are continuous. The following lists our results:

- *Mean Square Error (MSE)*: 6.18 dB^2
  - On average, the squared difference between our model's predicted and actual SSIM_dB values is 6.18 dB^2. This indicates that the model is somewhat sensitive to larger errors.
- *Root Mean Square Error (RMSE)*: 2.49 dB
  - Our model's predictions deviate from the actual values by about 2.49 decibels on average. This gives us a clear sense of the typical error magnitude in meaningful units.
- *Mean Absolute Percentage Error (MAPE)*: 9.03%
  - Our predictions are, on average, 9% off from the true SSIM_dB values. This relative error helps us assess the model's accuracy across different scales.
- *R^2 Score*: 0.695
  - Our model explains about 69.5% of the variance in the actual SSIM_dB values. This suggests that it captures most of the underlying signal, though there's still room for improvement.

Trustee's evaluation showed that the surrogate decision tree stayed fairly aligned with the original random forest. Although its predictive performance dropped compared to the forest, with an MSE of 8.23, RMSE of 2.86 dB, MAPE of 10.12%, and an R^2 of 0.59 against the ground truth, it has a fidelity R^2 of 0.75 relative to the forest. All five input features were used throughout the tree, and there was no sign of shortcutting or over reliance on any single feature. This suggests that our random forest learned a meaningful, robust structure rather than cheating.

6 Conclusion

In this project, we built machine learning models to predict different QoE metrics: resolution, chunk size, rebuffering events, and SSIM, using only network metadata from Stanford's Puffer streaming platform. Our results show that packet-level features can offer some insight into user experience, but the accuracy really depended on the task. Resolution prediction was impacted by class imbalance, chunk size regression didn't generalize well, and rebuffering detection was limited by how few positive cases we had. SSIM regression stood out with more promising results. We used the Trustee framework to interpret all our models, and while it helped with transparency, the decision tree surrogates didn't always match the original model performance. Overall, this partially supports our hypothesis that lightweight, privacy-preserving inputs can be used for QoE estimation, though success depends a lot on data quality and how the problem is framed. Future work should explore broader and more robust datasets, tracking patterns over entire sessions, and designing models with more generalizable architectures.

References

[1] Bronzino, F., Schmitt, P., Ayoubi, S., Feamster, N., Teixeira, R., Wassermann, S., and Sundaresan, S. Lightweight, general inference of streaming video quality from encrypted traffic. *CoRR abs/1901.05800* (2019). (Cited on page 2.)

[2] Jacobs, A. S., Beltiukov, R., Willinger, W., Ferreira, R. A., Gupta, A., and Granville, L. Z. Ai/ml for network security: The emperor has no clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (2022), pp. 1537–1551. (Cited on page 2.)

[3] Yan, F., Netravali, R., Narayan, K., Ratnasamy, S., Shenker, S., and Winstein, K. Puffer: Streaming video with randomized buffering for increased performance. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (2020), pp. 101–115. (Cited on pages 2 and 3.)