# WebQMon.ai: Gateway-based Web QoE Assessment Using Lightweight Neural Networks

Enge Song[1], Tian Pan[1,2], Qiang Fu[3], Chenhao Jia[1],
Jiao Zhang[1,2], Tao Huang[1,2], and Yunjie Liu[1,2]

[1] State Key Laboratory of Networking and Switching Technology, BUPT, China
[2] Purple Mountain Laboratories, China
[3] Royal Melbourne Institute of Technology, Australia

**Abstract.** Users' perception of their experience accessing web pages greatly affects users' willingness to continue browsing the website. However, it is difficult to assess user perception through a generic Quality of Experience (QoE) model. Web content consists of a large variety of static as well as dynamic objects, with some of them coming from the remote sites. This makes QoE assessment a challenge for the traditional methods. To build a generic QoE model, we introduce WebQMon.ai, a lightweight Web QoE assessment architecture using machine learning methods without setting any specific formula or threshold. WebQMon.ai can evaluate web-browsing QoE using mostly network-layer data with only one piece of application-layer information, the *referer* in the HTTP header, which is used to aggregate the packets associated with the same web page. The distribution of the arriving packets requested by the web page is used to construct WebQMon.ai. WebQMon.ai requires little storage space (80KB˜6MB). More importantly it can be deployed directly at edge routers/gateways, due to the weak dependence on the application-layer payload. We further improved our algorithm by *ensemble learning* combining multiple orthogonal features, to generate a stronger classifier. We evaluated WebQMon.ai on three popular websites. It shows that the QoE assessment results for *4,800* unknown samples can be obtained within just *0.07s* and reach an average accuracy of *97%*.

**Keywords:** Web-browsing QoE · Neural networks.

## 1 Introduction

The web-based Internet activities produce a large amount of HTTP traffic [15]. Internet users often visit a variety of websites to search for information, watch video clips or socialise with someone. The page loading latency is critical to user experience (UE). A study shows that there is a strong correlation between the performance of e-commerce websites and online shoppers' behavior [1]. The observation from the experiment with 1,048 online shoppers indicates that *two seconds* is the critical threshold for page loading latency, after which consumers

This work was supported by National Key Research and Development Program of China (2019YFB1802600). Co-corresponding authors: Tian Pan (pan@bupt.edu.cn) and Qiang Fu (qiang.fu@rmit.edu.au).

may become impatient if the page is still not loaded, and 40% of the consumers will abandon the site if it goes beyond three seconds. Research also shows that slow page loading damages consumers' loyalty to an e-commerce site, especially for high spenders [3]. Up to 79% of online shoppers who experienced a dissatisfying visit would never visit the site again while 27% of them would not visit its physical store either. QoS has been widely used to measure network performance. But, it is not a direct reflection of QoE [5]. ISPs and equipment vendors can however leverage their knowledge of the traffic going through their networks and create hypothetical QoE prediction models to estimate UE or QoE anywhere in the network. They can then refine the network settings or give feedback and recommendations to website owners to improve QoE.

However, real-time QoE assessment is a challenge. Casas et al. [4] propose YOUQMON, which can predict in real-time the stalling events of YouTube videos with network-layer data. Unfortunately, it is not a generic QoE assessment tool, as the thresholds and formulas are made specific to YouTube. The web-browsing QoE is mostly affected by the loading time of the last visible object shown on the screen, that is, above-the-fold (ATF) time. The study in [7] shows that among a variety of QoE metrics including page load time (PLT), ATF is most correlated to user experience. Many studies use ATF or its variants to capture how users perceive web-browsing experience [7, 9, 11, 12, 14]. Although predicting QoE by ATF works well, it is not a trivial task to obtain ATF. Most of the existing methods obtain ATF or its variants with client-side support by analyzing the video recordings of the web page rendering process [9, 11, 17] or tracing loading time of different resource types [6, 7, 12, 14, 16]. However, these analysis methods have a common issue: relying on client-side support to install software or hardware plugins and thus obtain or infer ATF or its variants. In contrast, we use machine learning methods to predict ATF.

To this end, we design WebQMon.ai, a generic real-time tool for assessing web-browsing QoE without requiring client-side support. It uses network-layer data and a single piece of application-layer data (the *referer* field in HTTP header) to predict ATF. As the *referer* is in the first few bytes of the HTTP request packet, no packet reassembly is needed. The *referer* is not even required, if the content from the third-party web site is not a concern. Hence, WebQMon.ai can be deployed at access routers or gateways without additional security or privacy concerns, which may be an issue for those requiring client-side support.

As ATF increases QoE deteriorates. We can classify ATF into multiple categories. Each category corresponds to a certain level of QoE, *e.g.*, an ATF below two seconds indicating good user experience [1]. We gather TCP traffic generated from web browsing, and then characterize the traffic by the two proposed traffic metrics: *Traffic Volume per Second* (TVS) and *Cumulative Traffic Volume* (CTV). TVS and CTV in time series exhibit different patterns under different network conditions, which is a good indication of distinct QoE. WebQMon.ai can predict QoE by distinguishing these patterns. Based on this architecture, we propose five supervised learning-based models to classify the patterns. The collected samples of the traffic metrics in time series are labeled by the esti-

mated ATF. The labeled data is then used as training data to train the machine learning model. The trained model takes up a small amount of storage space, from tens of KBs to a few MBs. WebQMon.ai is capable of real-time and accurate prediction. For instance, more than four thousand samples can be predicted within *0.07* seconds, with an average accuracy of *97%*.

Our major contributions are summarized as follows:

- We propose WebQMon.ai, a generic web-browsing QoE assessment system, capable of real-time prediction with high accuracy. It can be deployed on gateways instead of end hosts, with little storage space required. WebQMon.ai is powered by a data-driven model. It can be easily updated with new data to adapt to new types of web content in a timely manner, without the complexity of mathematical formulations or threshold settings (§ 3).
- To realize the data-driven model, we develop five models based on machine learning algorithms and the WebQMon.ai architecture to predict ATF. These methods are lightweight, easy to train and enable real-time and accurate prediction without being limited to a certain type of content (§ 4).
- We implement the five models with 1,876 lines of code, available at our github repository [2] (§ 5). The extensive evaluation shows that WebQMon.ai works very well on QoE prediction (§ 6).

## 2 Related Work

The current work requires client-side support for web QoE assessment. Some of the approaches rely on models specifically designed for a particular web site such as YouTube. WebQMon.ai is a generic architecture that can be easily applied to different types of web content. The other adopt generic models using formulas or machine learning models. These solutions acquire ATF and/or its variants with client-side support, and then perform QoE mapping through formulas or machine learning models. These solution are conceptually different to WebQMon.ai, which aims to estimate ATF without client-side support.

**Non-generic models.** The solutions in [4, 10] predict QoE through mathematical formulations and empirical threshold settings. This is a complex procedure, and more importantly, limits its applicability to other types of web content. For example in [4], YouTube stalling events are predicted using two thresholds $\alpha_{sta}$ and $\alpha_{play}$, which are estimated from large measurement campaigns. This is a complex procedure, and more importantly, limits its applicability to other types of web content [4]. For a different application, it may require to re-estimate the threshold from another large measurement campaign. WebQMon.ai is data-driven, which does not require threshold settings or mathematical formulations.

**Formula-based QoE mapping.** Given the user-perceived metrics, the solutions in [11, 14] uses formulas for QoE mapping. For example, the work in [11] analyzes the video frames of the recorded web page rendering process and then obtains ATF related metrics such as SpeedIndex. The work in [14] has a customized application embedded into the user devices to get the relevant information for QoE mapping. These systems rely on client-side plugins to obtain relevant metrics to do QoE mapping and have to be deployed on the client-side.

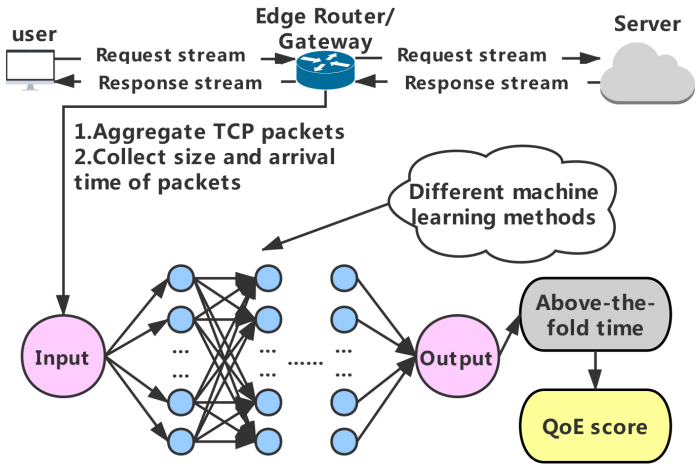



**Fig. 1.** WebQMon.ai architecture.

**Machine learning-based QoE mapping.** Some solutions use machine learning models for fine-grained QoE mapping, after acquiring ATF or relevant metrics from the client-side [7, 9, 12, 13]. The studies in [7, 12, 13] get the metrics through the browser's API. Similar to [11], the work in [9] obtains ATF through analyzing the recorded web page rendering process. These solutions are conceptually different to our approach, as WebQMon.ai uses machine learning models to estimate ATF based on traffic characteristics. However, we did get inspired by these studies and adopted the machine learning-based approach.

## 3 WebQMon.ai Architecture

### 3.1 System Architecture

ATF determines user's web-browsing experience. We propose WebQMon.ai to predict ATF and thus user experience. We collect network-level data from the Gateway and transform raw data into a useful format (Fig. 1). Then, our model is trained with the processed data. After that, WebQMon.ai can predict ATF when the user visits the web pages. In § 4, based on the architecture, we propose five models using different input metrics and machine learning algorithms. These models can be trained and/or updated with the diverse types of content or web sites, giving it the ability to learn and adapt to the new context.

The access routers or gateways all perform DPI (Deep Packet Inspection) for a variety of reasons. Raw data is readily available without needing additional resources. No packet reassembly is even needed as packets are processed as they need to be. We only need to get the statistics and construct the proposed traffic metrics. As to be discussed in § 3.2, the overhead can be managed by following vendor's restrictions on sampling intervals. The test time of the models is instant as shown in § 6, predicting over 4,000 samples within 0.07 seconds. The storage requirement is in the order of tens of KBs to a few MBs. All these indicate that WebQMon.ai can be conveniently deployed at gateways.

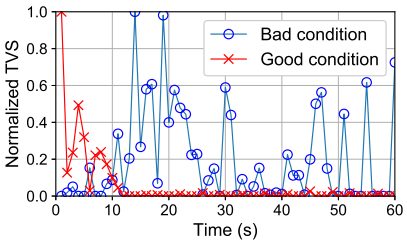



**Fig. 2.** Traffic Volume per Second.

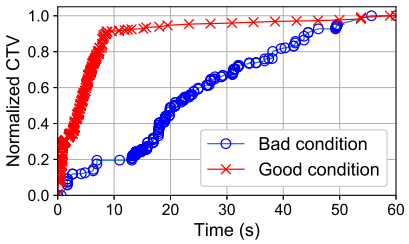



**Fig. 3.** Cumulative Traffic Volume.

### 3.2 Dataset and Data Preprocessing

The training and test dataset are derived from the TCP streams appearing when the user visits a web page. In Fig. 1, we can easily get all TCP traffic at the Edge Router or Gateway. The *referer*, a field in HTTP header, is used to identify and aggregate the traffic requested by the visited web page from other sources such as a third-party website. The arrival pattern of TCP packets is highly correlated to network conditions. The arriving packets are from different TCP flows. We will first need to aggregate the packets into their individual flows to establish the correlation. To extract meaningful flow features that can reflect network dynamics, we propose two *traffic metrics* to be defined below.

*Traffic Volume per Second* ($TVS$) measures the instantaneous throughput of a flow. The challenge is the sampling granularity. A fine granularity may cause a high-level of measurement overhead, which the gateway may struggle to handle. A coarse granularity may result in the loss of detailed flow features, which will decrease classification accuracy. The sampling interval is set to one second, the finest sampling granularity limited by the switch without overloading the gateway. Fig. 2 shows the normalized $TVS$ in time series with good and bad network conditions, respectively. Good network conditions result in satisfactory ATF while bad network conditions result in unsatisfactory ATF, as defined in Eq.(1). We will elaborate on this in § 5. It shows that when the network condition is good, a large amount of content arrives quickly and the peak rate appears at the early stage of the transmission. In contrast, when the network condition is bad, the content is loaded slowly and there is no clear peak rate.

*Cumulative Traffic Volume* ($CTV$) measures the total amount of traffic received over a flow at a time point. Fig. 3 illustrates the normalized $CTV$ in time series for different network conditions, corresponding to satisfactory and unsatisfactory ATF, respectively. With the good network condition, the curve shows a steep slope, or otherwise a shallow slope.

Both $TVS$ and $CTV$ in time series exhibit a pattern clearly correlated to network conditions. This is a strong indication that these two traffic metrics may be used to reflect web-browsing QoE.

### 3.3 Training and Prediction

The raw TCP data is processed to generate the proposed traffic metrics. Each of them is marked with a unique label for supervised learning and the ATF can be predicted by exploring the time series patterns of the metrics. When using

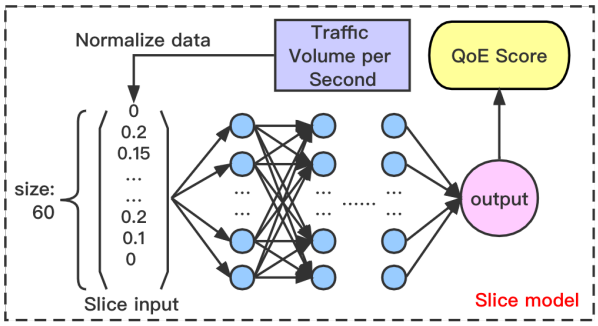



**Fig. 4.** Architecture of the Slice model.

machine learning methods, *TVS* and *CTV* need to be processed to generate input variables. We will elaborate on this in a later section. In the training stage, the difference between the predicted value and the label is reduced iteration by iteration. In order to get the prediction results (*i.e.*, ATF), we conduct simple matrix calculations. The ATF can then be mapped to the QoE score by a mapping function [7]. This enables real-time QoE assessment.

## 4 WebQMon.ai Algorithm

To explore *TVS* and *CTV* for QoE assessment, we design five classification models based on machine learning methods, namely, Slice, NN, LSTM, R-LSTM and Combine. All the models use the WebQMon.ai architecture, with selected machine learning algorithms and feature variables, to be discussed in § 4.3 . Slice classifies *TVS* using the fully connected neural networks. NN is based on the maximum slope and time domain features of *CTV*. LSTM relies on the *linear interpolation* data of *CTV*. R-LSTM improves LSTM by reversing the input variables. The fifth method, Combine, uses the idea of *ensemble learning*, which can subtly combine the predictions from multiple learning models to achieve more accurate, stable, and robust results. It is particularly suitable in our case as the features of Slice, NN and R-LSTM are distinct to each other.

### 4.1 Basic Classification Models

The packets that arrive within 60 seconds after the HTTP request are collected and preprocessed. We calculate and normalize *TVS* and *CTV*, which are then used as the input of the first four models. The output is the label of ATF or QoE score, which is determined by the probability of the estimated ATF, for example, being below or above 2 seconds.

**Slice** takes *TVS* as input. The different forms of *TVS* correspond to different ATF. Slice uses fully connected neural networks for fast training and testing and good performance. Since packets are collected for 60 seconds, the data format of *TVS* is a 60-dimensional vector as shown in Fig. 4. The input variable is the normalized data.

**NN** takes *CTV* as input and uses fully connected neural networks. Fig. 3 shows that the *maximum slope* of *CTV* can reflect the network condition — the

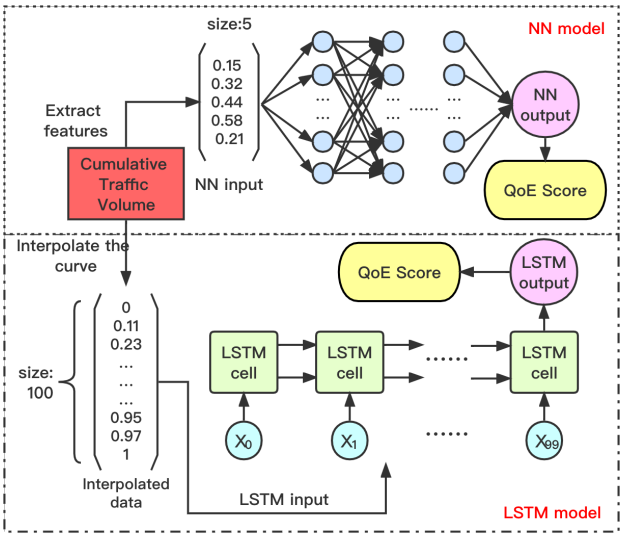



**Fig. 5.** Architecture of the NN model and the LSTM model.

deeper the slope is, the better the condition is. This makes *maximum slope* one of the classification features. The time when $CTV$ reaches x% is denoted as $t_{x\%}$. It makes sense to use $t_{25\%}$, $t_{50\%}$, $t_{75\%}$ and $t_{90\%}$ as inputs, to capture $CTV$ in time domain while making them relatively independent. Together with *maximum slope*, we have a five-dimensional input, that is, $t_{25\%}$, $t_{50\%}$, $t_{75\%}$, $t_{90\%}$, *maximum slope*. Note that the time domain inputs $t_{x\%}$ do have some correlation. This violates the assumption to use NN and explains its less impressive performance on ternary classification in § 6. LSTM is more suitable to use $CTV$.

**LSTM** (*Long Short-Term Memory Neural Networks*), a variant of *Recurrent Neural Networks* (RNN), is often used to process the time series data. Compared to the simple RNN, LSTM can keep the *Long-Term Memory* feature of the sequence. Therefore, LSTM is particularly suitable to address the long delays of $CTV$ and capture the dependency between data points. As shown in Fig. 5, to construct the input, *linear interpolation* is used to create approximate 100 points of $CTV$ for curve fitting.

**R-LSTM.** The packets that arrive earlier within the 60 seconds represent the initial response of the loading process, and thus may have a greater impact on user experience. However, in LSTM the early input has less impact on the output. This is not desirable. Therefore, we *reverse* the interpolated data, that is, the early data points of $CTV$ are processed last, giving them more influence on the output. We call this model R-LSTM. Note that LSTM and R-LSTM differ only in the input vector. We shall demonstrate in § 6 that R-LSTM indeed performs better than LSTM.

### 4.2 Combine Classification Model

**Combine** leverages *ensemble learning*. The idea is to first generate multiple learners, then combine them with some integration strategies, and finally gener-

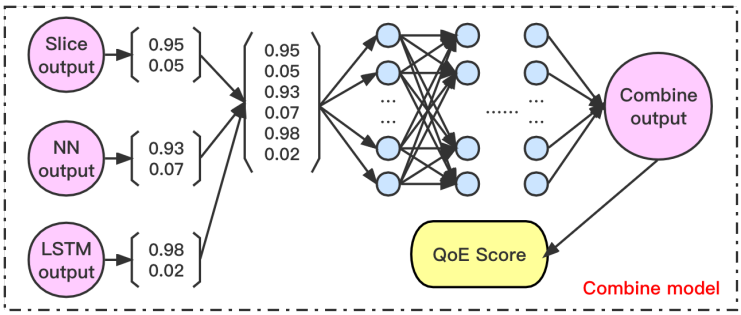



**Fig. 6.** Architecture of the Combine model.

ate the output. The theoretical basis of *ensemble learning* is that strong learners and weak learners are equivalent, so we can find ways to convert weak learners into strong learners instead of having to directly search for strong learners that are hard to find. Take the binary classification problem as an example. Assume that there are $N$ independent classifiers with an error rate of $p$. Using a simple voting method to combine all the classifiers, the error rate of the integrated classifier is $P_{error} = \sum_{k=0}^{N/2} C_N^k (1-p)^k p^{N-k}$. It can be seen from the equation that when $p < 0.5$, the error rate $P_{error}$ decreases as $N$ increases. If the error rate of each classifier is less than 0.5 and they are independent of each other, the more the number of classifiers is, the smaller the $P_{error}$ will be. When $N$ is infinite, the $P_{error}$ is 0. In addition, the ensemble model works well when these weak classifiers perform well individually and have different features.

Since R-LSTM performs better than LSTM, we decide to combine R-LSTM, Slice, and NN through *ensemble learning*. As the features of these three classifiers are distinct to each other, the ensemble model may work well. After completing the training of the three basic models, we combine them using fully connected neural networks. As shown in Fig. 6, based on binary classification, the predicted values of the three models are combined into a six-dimensional vector as the input variable of the fully connected neural networks.

### 4.3 Feature and Algorithm Selection

For the fully connected neural networks, we assume that all inputs are relatively independent of each other. In contrast, the idea behind LSTM neural networks is to make use of sequential correlated information. The models are designed to take advantage of the different input variables. For example, $TVS$ is used for Slice while $CTV$ is used for NN. The input variables for LSTM and R-LSTM are derived from $CTV$, with the earlier input related to the latter input. The other three models does not have this property and thus can take a simple approach using the fully connected neural networks. Combine aims to take advantage of the other models for better performance.

All these models are data-driven methods, which means that we can update our models with new data in a timely manner on a regular basis. The real-time ability of these methods allows ISPs and equipment vendors to assess user experience on the fly and take actions if necessary.

# 5 Implementation and Experimental Settings

## 5.1 Dataset collection

**Data Collection.** Our experiments took place from September 2020 to February 2021 in a laboratory. The hardware was equipped with i5-8600K CPU and GTX 1070Ti. We selected three websites to visit, that is, "amazon.com", "sina.com.cn", ranked the eighth and the nineteenth, respectively, in Alexa Traffic Rank, and "youku.com.cn," which represent widely used *shopping*, *news* and *video* sites. For simplicity, we will use Amazon, Sina and Youku to refer to these websites. Our data was obtained by visiting the homepage of the website. Note that the homepage is the most popular, diverse and dynamic page and its content changes over the months of the sampling period. Being able to classify the homepage of the three different types of websites is a challenge. We collected the packets arriving within 60 seconds after the user visited the website. Then, we got the $TVS$ and the $CTV$ from these packets and labeled this sample of $TVS$ or $CTV$ in time series according to the estimated ATF. We used *Dummynet* to create a bottleneck to control the network condition and construct samples with different labels accordingly. There were already mature plugins for the aggregation of packets belonging to the same visit. Traffic aggregation was implemented by the Firefox browser.

**Data Labeling.** We evaluate the performance of Slice, NN, LSTM, R-LSTM and Combine to predict ATF using the collected dataset. Our experiments have two parts: binary classification and ternary classification. We adjust the network condition through *Dummynet* to ensure that the ATF meets our requirements, measured by the chrome plugins released in [6]. We then label the data according to the estimated ATF using the following rules based on Akamai's research on customer behaviours, 2 seconds being the psychological threshold [1]:

$$Binary : label = \begin{cases} 0,\ ATF \leq 2s \\ 1,\ ATF > 2s \end{cases}; \quad Ternary : label \quad = \begin{cases} 0,\ ATF \leq 2s \\ 1,\ 2s < ATF \leq 5s \\ 2,\ ATF > 5s \end{cases} \quad (1)$$

We use a simple one-to-one correspondence to map ATF to QoE, which could be implemented through a mapping function [7]. For binary classification, "0" represents good UE, and "1" represents poor UE. For ternary classification, "0" means good UE, "1" means poor UE, and "2" means terrible UE.

**Dataset.** For binary classification, we collected about *16,000, 16,000, 8,000* samples from Sina, Amazon and Youku, respectively, of which the proportion of positive and negative samples was 1:1. This balanced split is for the purpose of learning, to ensure that the models learn both positive and negative cases. In reality the occurrence of positive cases is much less common than that of negative cases. However, only positive cases are of interest to ISPs and vendors. For ternary classification, we only got extra 6,000 samples from Sina due to being blocked later on. The proportion of "0", "1" and "2" samples was 4:4:3. 70% of the samples are used as the training dataset with the rest as the test dataset. Since most websites have a mechanism against crawling, this prevents us from

**Table 1. Other Model Parameters.**

| | No. of input units | No. of hidden units | No. of hidden layers |
|---|---|---|---|
| Slice | 60 | 480 | 2 |
| NN | 5 | 40 | 2 |
| R-/LSTM | 100 | 256 | 1 |
| Combine | 6 | 48 | 2 |

**Table 2. Dropout effect.**

| | No Dropout | $P_{keep}$ of 80% |
|---|---|---|
| *Accuracy* | 0.9425 | 0.9495 |
| *Precision* | 0.9569 | 0.97 |
| *Recall* | 0.9512 | 0.9489 |
| *F1 score* | 0.954 | 0.9593 |

frequently refreshing the same web page. For example, when we collected data from Amazon, it always required to provide a verification code. This made it extremely difficult to create a larger dataset. We believe that the amount of available data can demonstrate the feasibility of the model to a certain extent, and the three websites represent the typical scenarios of web browsing activities. For a certain website, different models share the same training and test dataset, ensuring that the results are not affected by how the dataset is split. We use the training dataset to train each model separately, that is, we train five models for each of the test websites, for a total of 15 models.

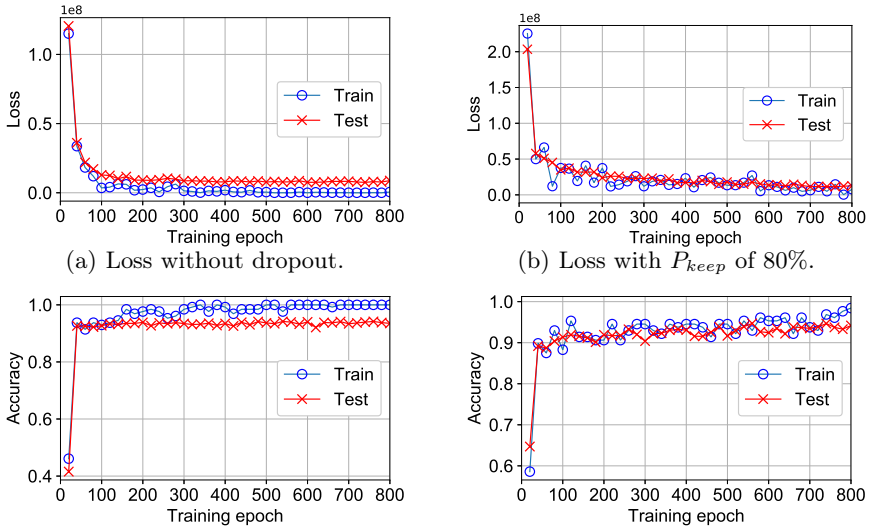
### 5.2 Model Parameters

**Parameter Setting.** The parameters common to all models are: *number of iterations=100,000*, *learning rate=0.003*, and *batch size=128*. These parameters were chosen to achieve the highest accuracy on the validation set. Other parameters are shown in the Table 1. The *Number of input units* is the dimension of the input variable, while the *Number of hidden units* is the number of neurons in the hidden layer. The *Number of hidden layers* is the number of the network layers between input and output layers. The classic "trial-and-error" method was used for creating neural network layers. It was a simple process with several iterations. There was no need to readjust the parameters and it worked well across all the three web sites, indicating the ease of parameter selection. It also shows that all the five models have a simple lightweight architecture.

**Overfitting and Dropout.** In machine learning, overfitting may occur when a model corresponds too closely or exactly to the training data and thus may fail to fit the test data or predict reliably. Dropout regularization is one of the popular techniques to avoid overfitting, which helps make the model globally fit. As shown in § 6, the Slice model is overfitting. Dropout is applied by deactivating a portion of neurons at the training time. $P_{keep}$ represents the portion of active neurons. After tests on the validation set, we found that the Slice model had the best performance with $P_{keep}$ set to 80%.

## 6 Experimental Results

*Accuracy*, *precision*, *recall* and *F1 score* are commonly used performance metrics. In a real-world scenario, the chance to have an unsatisfactory ATF (positive) is small. *Accuracy* can be misleading for imbalanced data sets, *e.g.*, small portion of positives vs large portion of negatives. *Precision* represents true positives per predicted positive while *recall* represents true positives per real positive. As *recall* increases, *precision* may drop, and vice versa. *F1 score* is the harmonic




(a) Loss without dropout. (b) Loss with $P_{keep}$ of 80%.

(c) Accuracy without dropout. (d) Accuracy with $P_{keep}$ of 80%.

**Fig. 7.** Without dropout regularization vs. $P_{keep}$ of 80%.

mean of *precision* and *recall*. A high *F1 score* indicates a good balance between the two. As we have a focus on positive cases and the number of negatives is unknown and large, this makes the latter three metrics particularly suitable in our evaluation. Eq.(1) shows the label of our data. For binary classification, label with "1" is the positive instance and label with "0" is the negative instance. For ternary classification, each category is treated as a positive class for calculating the values of the metrics. The test dataset is used to predict labels. The predicted label is compared to the actual label, which serves as the ground truth.

### 6.1 Basic Models

**Dropout Effect on Slice.** Fig. 7 compares the results with and without dropout regularization ($P_{keep}$ set to 80%). It shows that the Slice model is indeed overfitting. Fig. 7(a) and 7(b) shows the loss functions measuring the inconsistency between the predicted value and the actual label against the number of training epochs. The loss functions decrease as the number of epochs increases. Without dropout (Fig. 7(a)), the loss function of the test dataset is not as small as that of the training dataset, and this gap does not decrease as the number of epochs increases. This is an indication of overfitting. With dropout (Fig. 7(b)), the loss function of the training and test datasets matches each other very well. Similarly, Fig. 7(c) and 7(d) shows the accuracy performance of the model against the number of epochs. The accuracy improves as the number of epochs increases. Without dropout (Fig. 7(c)), the accuracy of the test dataset does not match the accuracy of the training dataset, which reaches 100%, and the gap is not narrowed as the number of epochs increases. The model fits the training dataset well but fails to fit the test dataset. This indicates that the Slice model is indeed overfitting. With dropout (Fig. 7(d)), the accuracy of the training and the test

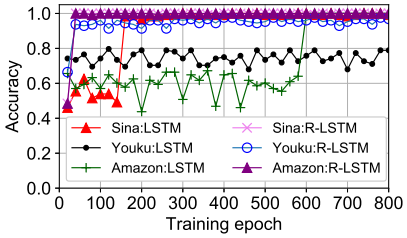



**Fig. 8.** Accuracy comparison (LSTM vs R-LSTM).

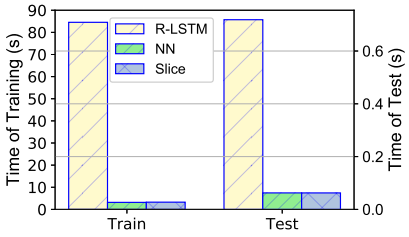



**Fig. 9.** Training and test time comparison (Amazon).

datasets matches each other closely. Although dropout reduces the accuracy of the training dataset, it helps improve the accuracy of the test dataset.

Table 2 shows that, with dropout regularization, the accuracy, precision and F1 score of the model slightly increase but the recall decreases a little bit. Note that precision and recall are mutually influential (true positives per predicted positive vs per real positive). True positives are usually achieved at the cost of false positives. A high recall may come with a low precision, and vice versa [8]. Ideally, we want to keep both precision and recall high, to ensure that the positives are true positives without any missing positives. A high F1 score (as shown in Table 2) indicates a good balance between the two. Based on these results, we can conclude that dropout regularization improves the performance of Slice. Dropout is applied to Slice in the later experiments.

**LSTM vs R-LSTM.** We will now demonstrate that the improved LSTM model, R-LSTM, has better performance. Fig. 8 shows the prediction accuracy by LSTM and R-LSTM against the number of training epochs for the three test websites. It shows that the model converges much faster with R-LSTM at roughly epoch 50 for both Sina and Amazon. With LSTM, the model converges at roughly epochs 150 and 600, for Sina and Amazon, respectively. For Youku, R-LSTM has much higher accuracy than LSTM, roughly 97% vs. 75%. Based on these results, we believe that reversing the input variables can significantly improve convergence time as well as accuracy, depending on the type of web content. In later experiments, we will use R-LSTM instead of LSTM.

**Performance.** Table 3 shows the performance of the models for binary classification on accuracy, precision, recall, and F1 score. The data volumes of the test datasets for Amazon, Sina, and Youku are approximately 4,800, 4,800, 2,400 samples (30% of the dataset), respectively. The results suggest that for Amazon and Sina, the three models work remarkably well on ATF prediction. The performance on all the metrics is close to 1. There are fewer than five prediction

**Table 3. ATF Prediction performance comparison.**

| | **Amazon** | | | **Sina** | | | **Youku** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Slice | NN | R-LSTM | Slice | NN | R-LSTM | Slice | NN | R-LSTM |
| *Accuracy* | 0.9991 | 0.9991 | **0.9995** | **0.9988** | 0.9986 | 0.9983 | 0.9466 | 0.9536 | **0.9684** |
| *Precision* | **1** | 0.9996 | **1** | **0.9992** | 0.9987 | 0.9987 | 0.9642 | 0.9674 | **0.9789** |
| *Recall* | 0.9984 | 0.9988 | **0.9992** | **0.9983** | **0.9983** | 0.9979 | 0.9501 | 0.9583 | **0.9690** |
| *F1 score* | 0.9992 | 0.9992 | **0.9996** | **0.9987** | 0.9985 | 0.9983 | 0.9571 | 0.9628 | **0.9739** |

**Table 4. Performance comparison with Combine on Youku.**

| | R-LSTM | Slice | NN | Combine |
|---|---|---|---|---|
| *Accuracy* | 0.9546 | 0.9546 | 0.9546 | **0.9693** |
| *Precision* | **0.9899** | 0.9728 | 0.9765 | 0.9826 |
| *Recall* | 0.939 | 0.9561 | 0.9523 | **0.9695** |
| *F*1 *score* | 0.9638 | 0.9644 | 0.9643 | **0.976** |

**Table 5. Performance for ternary classification.**

| | R-LSTM | Slice | NN | Combine |
|---|---|---|---|---|
| ... | 0.9439 | 0.9109 | 0.8607 | **0.9518** |
| ... | **0.941** | 0.891 | 0.828 | **0.941** |
| ... | 0.9192 | 0.8907 | 0.8299 | **0.9452** |
| ... | 0.9281 | 0.8908 | 0.8289 | **0.943** |

errors for Amazon and Sina. For Youku, the performance is less impressive. The values of the performance metrics vary between 0.94 and 0.98. Since Youku is a video site, there are a lot of dynamically loaded content on its home page. Therefore, regardless of the network conditions, the diversity of the content (or the dataset) on its home page is a challenge to classification and thus reduces the accuracy. Within the 2,400 samples, there are about 100 prediction errors, an error rate roughly between 2% to 6%. Among the three models, R-LSTM appears to be the best performer on Youku.

**Training and Test Time.** Fig. 9 shows the training and test time of the three models on Amazon. The data volume of the training and test datasets are about 11,200 and 4,800 samples, respectively. It shows that the training time of R-LSTM is much higher than that of the other two models. The training time of LSTM depends on the number of iterations, which is 100 in our model. Therefore, *backpropagation* of LSTM needs to be performed 100 times per training batch. In contrast, Slice and NN use fully connected neural networks as the classifier, which requires only one *backpropagation* per training batch. As a result, the training time of LSTM is much longer than that of Slice and NN. For test time, it is a similar situation. The *forward propagation* of LSTM requires 100 executions to generate an output, but the *forward propagation* of the fully connected neural network only needs to be performed once to generate an output. The time it takes R-LSTM to complete 4,800 predictions is much longer than it takes Slice and NN. However, as *backpropagation* is only needed for training, not for prediction, the gap between LSTM and NN/Slice on test time is much smaller than on training time. The time required for the three models to predict 4,800 samples is about 0.7s, 0.08s, and 0.07s, respectively. It confirms the possibility of using our model to assess the user's QoE in real-time.

### 6.2 Combine

**Ensemble Learning.** We combine the trained Slice, NN, and R-LSTM through a fully connected neural network to generate a new model, Combine. The three basic models were trained using the 70% of the total dataset as the training dataset and were able to predict with few errors. If we reuse the 70% for Combine, the inputs and the labels are likely to be the same or similar, making the training no longer meaningful. Therefore, Combine uses the remaining 30% as its training and test datasets. We still use 70:30 split of the dataset for training and testing. It is observed that all models perform very well on Amazon and Sina with no much difference, although Combine performs the best with no prediction errors at all. The advantage of Combine is getting clearer on Youku. Table 4 shows Combine's performance on Youku in comparison with the other models. The

performance of all models is not as great, because of the highly dynamic content on Youku's home page. However, Combine still performs the best on accuracy, recall and F1 score, and only behind R-LSTM on precision. This motivates us to further explore the effectiveness of ensemble learning on ternary classification.

**Ternary Classification.** Table 5 shows the performance on the test dataset. The precision, recall, and F1 score here are established as follows: an initial value of the metrics is obtained for each of the three QoE categories as a binary classification problem, and then a weighted average across the three QoE categories is calculated, which becomes the value of the metrics shown in Table 5. It shows that the performance of the models is substantially degraded for ternary classification. This is expected because of the finer granularity of the QoE. R-LSTM is the best performer among the three basic models, with the values of the metrics ranging from 0.919 to 0.941. NN is the worst, with F1 score of only 0.8289. The features used by NN do not describe well the differences between categories "1" and "2". Also as stated in § 4, NN's inputs are not as independent as assumed. In contrast, R-LSTM performs well because the accumulated data makes the input statistically significant, which facilitates classification. Slice sits in the middle, with the values of the metrics varying slightly around 0.90.

Combine performs the best across all the metrics, with their values all greater than 0.94. Through ensemble learning, we use three weak classifiers to form a strong classifier, making the model well suited for ternary classification.

### 6.3 Summary

WebQMon.ai can predict ATF well when users visit the websites, whether it is a binary or ternary classification problem. R-LSTM performs the best among the three basic models, but it takes the longest time to train and predict. Slice is more balanced, having a reasonable performance with the shortest training and prediction time. NN requires a short time to train and predict but has the worst performance. Combine performs the best through ensemble learning. However, since the model needs to use the results of the three basic models, it has the longest training and prediction time. Furthermore, we collected data for four months, during which the content of the websites changed greatly. Nonetheless, our model still predicts ATF well, which proves that the model can be updated by new data to accommodate changes in website content. It would be difficult to do this through threshold settings or mathematical formulations. In addition, the trained model takes up very little storage space, a minimum of *80KB* and a maximum of *6MB*. WebQMon.ai only requires the *referer* from the application layer, making it possible to deploy on edge routers.

## 7 Discussion

**HTTP vs HTTPS.** In contrast to current work, WebQMon.ai minimizes the use of application layer data — it only needs the *referer* to aggregate the traffic associated with a page. This is under the assumption that the content associated with a page may be sourced from other sites. Nevertheless, if we are only interested in the content from a particular site, WebQMon.ai does not need access to

application data at all. The limitation is that WebQMon.ai would not be able to predict page-based QoE, as the objects of the page may be from different sites. However, it can still provide a QoE assessment for the content from a particular site. This is of great value to content providers as well as ISPs.

**Placement.** Predicting web-browsing QoE on the client-side usually occupies user network bandwidth and needs the user to cooperate and install specific software. It may provide the best possible prediction accuracy, but it is also the most costly choice. If deployed on the server-side, it may work for the websites who have the budget to do so. However, from the client's point of view, this may result in the least accurate prediction, and the prediction is limited to a specific site. In contrast, WebQMon.ai can be deployed on edge routers/gateways, and thus is transparent to the server and the client. In some scenarios, WebQMon.ai may not be able to achieve the prediction as accurate as some client-based solutions. But it can still achieve a high level of accuracy, and more importantly, without the client-side constraints. WebQMon.ai has access to all the websites that ISP's clients are interested in. This enables close collaboration between ISPs and content providers to serve their clients.

**Versatility and real-time.** WebQMon.ai is powered by a data-driven model, which is easy to update and apply for all websites with different types of content. It can be updated on a regular basis as long as there is new data, that is, WebQMon.ai has the ability to learn and adjust to the new context. Current solutions that rely on empirical threshold settings or mathematical formulations are usually designed specifically to a certain site, limiting their applicability for other sites. In addition, WebQMon.ai only needs to use lightweight neural networks to achieve a high level of accuracy. This demonstrates not only that WeQMon.ai can get updated quickly and work in a real-time fashion, but also the practical applicability of machine learning in this field.

**Fine-grained ATF prediction.** WebQMon.ai can handle very well binary and ternary classifications, which are common cases for QoE prediction. However, as the granularity of QoE classification increases, the performance of WebQMon.ai deteriorates. At some stage, finer-grained ATF prediction may be required, which can be done through addressing the regression problem. The mapping from ATF to QoE can then be done through Mean Opinion Score (MOS) [7]. We can imagine this would improve the performance at the cost of training and test latency, due to the complexity of the model.

## 8 Conclusion

In this paper, we present WebQMon.ai to predict web-browsing QoE. WebQMon.ai relies on packet-level measurements without deeply parsing the packet payload, and thus can be deployed on edge routers/gateways instead of end hosts. WebQMon.ai is data-driven, empowered by lightweight supervised learning methods, which enables the system to learn and adapt to new contents in a timely manner. WebQMon.ai works very well for binary and ternary classification based QoE prediction, achieving a high level of accuracy in real-time. Furthermore, we demonstrate the potential and feasibility of machine learning methods in web-browsing QoE assessment.

# Bibliography


[1] Akamai research. https://www.akamai.com/us/en/about/news/press/2009-press/akamai-reveals-2-seconds-as-the-new-threshold-of-acceptability-for-ecommerce-web-page-response-times.jsp/, accessed 2020

[2] Webqmon.ai. https://github.com/songng/WebQMon.ai/, accessed 2022

[3] Caruana, A.: Service loyalty: The effects of service quality and the mediating role of customer satisfaction. European Journal of Marketing **36**(7/8), 811–828 (2002)

[4] Casas, P., Seufert, M., Schatz, R.: Youqmon:a system for on-line monitoring of youtube qoe in operational 3g networks. ACM SIGMETRICS Performance Evaluation Review **41**(2), 44–46 (2013)

[5] Chen, X., Li, Z., Zhang, Y., Long, R., Yu, H., Du, X., Mohsen, G.: Reinforcement learning based qos/qoe-aware service function chaining in software-driven 5g slices. Transactions on Emerging Telecommunications Technologies pp. e3477– (2018)

[6] Da Hora, D., Rossi, D., Christophides, V., Teixeira, R.: A practical method for measuring web above-the-fold time. In: Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos. pp. 105–107. ACM (2018)

[7] Da Hora, D.N., Asrese, A.S., Christophides, V., Teixeira, R., Rossi, D.: Narrowing the gap between qos metrics and web qoe using above-the-fold metrics. In: International Conference on Passive and Active Network Measurement. pp. 31–43. Springer (2018)

[8] Davis, J., Goadrich, M.: The relationship between precision-recall and roc curves. In: Proceedings of the ICML 2006. pp. 233–240. ACM (2006)

[9] Dey, P., Ahammad, P., et al.: Perceived performance of top retail webpages in the wild. ACM SIGCOMM Computer Communication Review **47**(5), 42–47 (2017)

[10] Gutterman, C., Guo, K., Arora, S., Wang, X., Wu, L., Katz-Bassett, E., Zussman, G.: Requet: Real-time qoe detection for encrypted youtube traffic. In: Proceedings of the 10th ACM Multimedia Systems Conference. pp. 48–59 (2019)

[11] Hoßfeld, T., Metzger, F., Rossi, D.: Speed index: Relating the industrial standard for user perceived web performance to web qoe. In: 2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX). pp. 1–6. IEEE (2018)

[12] Huet, A., Houidi, Z.B., Mathieu, B., Rossi, D.: Detecting degradation of web browsing quality of experience. In: CNSM 2020. pp. 1–7. IEEE (2020)

[13] Saverimoutou, A., Mathieu, B., Vaton, S.: A 6-month analysis of factors impacting web browsing quality for qoe prediction. Computer Networks **164**, 106905 (2019)

[14] Seufert, M., Wehner, N., Wieser, V., Casas, P., Capdehourat, G.: Mind the (qoe) gap: On the incompatibility of web and video qoe models in the wild. In: CNSM 2020. pp. 1–5. IEEE (2020)

[15] Singh, V., Bharti, S., Pathak, V., Sengar, A., Singh, T., Goswami, M.: On dominant characteristics of residential broadband internet traffic. In: ACM SIGCOMM Conference on Internet Measurement (2009)

[16] Subramanian, M., Ye, E., Korlipara, R., Smith, F.: Techniques for measuring above-the-fold page rendering (Aug 19 2014), uS Patent 8,812,648

[17] Varvello, M., Blackburn, J., Naylor, D., Papagiannaki, K.: Eyeorg: A platform for crowdsourcing web quality of experience measurements. In: Proceedings of the CoNEXT 2016. pp. 399–412. ACM (2016)