ERUDITE: a Deep Neural Network for Optimal Tuning of Adaptive Video Streaming Controllers

Luca De Cicco
luca.decicco@poliba.it
Politecnico di Bari
Bari, Italy

Giuseppe Cilli
g.cilli2@studenti.poliba.it
Politecnico di Bari
Bari, Italy

Saverio Mascolo
mascolo@poliba.it
Politecnico di Bari
Bari, Italy

ABSTRACT

Adaptive video streaming systems are expected to provide the best user experience to improve service engagement. To this purpose, the video player implements a controller to dynamically choose the most suitable video representation to be downloaded. It is well-known that finding one tuning of the controller's parameters which performs satisfactorily in a wide range of scenarios is very challenging. This paper studies the problem of providing users with (near) optimal Quality of Experience (QoE) for Dynamic Adaptive Streaming over HTTP (DASH) systems. We present ERUDITE, a closed-loop system to optimally tune – at run-time – the adaptive streaming controller's parameters to adapt to changing scenario's parameters. The proposed system is based on a Deep Neural Network (DNN) which continuously provides the streaming controller with estimates of optimal parameters based on measured metrics such as bandwidth samples and overall obtained QoE. The DNN is trained using a dataset that we have built by finding, for thousands of scenarios, the optimal adaptive streaming controller's parameters using a Bayesian optimization algorithm. Results, gathered considering a large number of diverse scenarios, show that ERUDITE is able to provide near optimal performances by reducing impairments due to rebuffering and video level switching.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → *Control methods*; • **Computer systems organization** → *Neural networks*.

KEYWORDS

Adaptive Video Streaming, Gaussian Optimization, Deep Neural Networks, Quality of Experience

ACM Reference Format:
Luca De Cicco, Giuseppe Cilli, and Saverio Mascolo. 2019. ERUDITE: a Deep Neural Network for Optimal Tuning of Adaptive Video Streaming Controllers. In *10th ACM Multimedia Systems Conference (MMSys '19), June 18–21, 2019, Amherst, MA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3304109.3306216

MMSys '19, June 18–21, 2019, Amherst, MA, USA

ACM ISBN 978-1-4503-6297-9/19/06…$15.00
https://doi.org/10.1145/3304109.3306216

1 INTRODUCTION

Video streaming is the primary source of Internet traffic globally and platforms such as YouTube, Netflix, and Facebook stream video contents to an ever-increasing audience worldwide. In order to ensure smooth reproduction of the video, streaming services must implement adaptivity to cope with the time-varying nature of the end-to-end available bandwidth. To this purpose, such systems employ *adaptive video streaming controllers* to dynamically select the video *representation*, or *level*, to download from a set of available bitrates $\mathcal{L} = \{l_1, l_2, \ldots, l_L\}$ with the ultimate goal of providing the best possible Quality of Experience (QoE) to the user. It is now well-established that the QoE is related to the following metrics, listed in decreased order of importance [3]: i) the *rebuffering ratio* and *frequency*, which should be kept as small as possible; ii) the *average video bitrate*, which should be as high as possible to improve visual quality; iii) the *video level switching* frequency and amplitude, which should be contained to provide a smooth user experience.

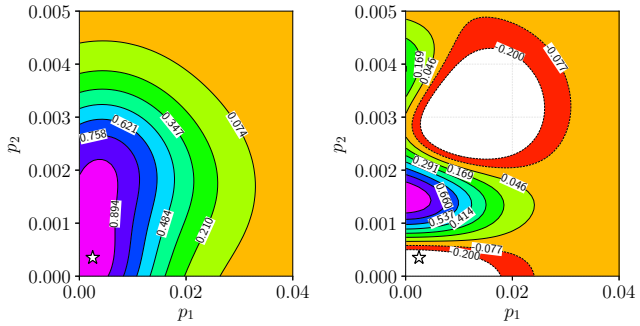
In this paper, we take an original approach which sits in between model-based and model-free methodologies. In particular, we propose ERUDITE: a *dEep neuRal network for optimal tUning of aDaptive vIdeo sTreaming controllErs*. ERUDITE is built around a controller whose parameters are dynamically tuned using a Deep Neural Network (DNN) to adapt to different video contents and bandwidth traces. The idea is to employ a controller whose theoretical properties have been analyzed and leave the optimal tuning of its parameters to a DNN which suggests the most appropriate set of controller's parameters to be set. The proposed design allows to separate the time-scales of the decisions of the closed-loop system: i) a faster inner loop – managed independently by the adaptive video streaming controller – reacts to sudden and unpredictable changes in scenario's parameters (i.e., the bandwidth); ii) a slower outer loop – governed by the DNN – acts as a supervisor and dynamically tunes the controller to adapt to long-term changes in scenario's parameters and achieve the best possible QoE.

ERUDITE differs from the approach proposed in [27] since it does not take decisions by directly considering bandwidth predictions, rather it reactively changes the controller's parameters when performance drops are measured. ERUDITE also differs significantly from Pensive [21], in that we do not invoke the DNN to select the video level at each chunk download. Instead, we leverage the DNN to find the best tuning to adapt to time-varying scenario's parameters.

The proposed methodology has been tested on a wide set of scenarios by considering a rich set of bandwidth traces and a large and diverse video catalog characterized by videos having different video chunk durations and video levels. Results show that ERUDITE is able to provide near optimal QoE by continuously refining

the set of controller's parameters in response to varying scenario conditions.

The rest of the paper is organized as follows: Section 2 reviews the existing control strategies for adaptive video streaming systems; Section 3 gives an overview of the proposed system; Section 4 describes the procedure employed to obtain the optimal controller parameters; Section 5 presents the scenario dataset employed to train ERUDITE's DNN; Section 6 presents the proposed DNNs architectures along with the procedure employed for training; Section 7 presents the results obtained; Section 8 discusses how to integrate ERUDITE into running streaming services and highlights future research directions, finally, Section 9 concludes the paper.



(a) Scenario *A* (b) Scenario *B*

Figure 1: QoE contour lines with respect to two of the four controller's parameters computed for two different scenarios. The other two parameters, i.e., p_3 and p_4, are set to the same value in both the scenarios. The star indicates the optimal parameters for Scenario *A*.

2 BACKGROUND

Several methodologies have been explored for designing adaptive video streaming controllers. Classical approaches base their decisions on the estimated available bandwidth (rate-based approaches) [16], on the buffer level (buffer-based) [14, 23], or on the combination of the two strategies [12, 17]. These algorithms do not directly optimize a particular QoE functional based on the metrics presented above, but rather strive to improve such metrics individually. Recently, a new class of algorithms has been developed taking decisions based on the optimization of QoE-related functionals. From the methodological point of view, control theoretical approaches, such as Model Predictive Control (MPC) [27], have been proven to be a viable solution to improve the QoE. The model-based approach proposed in [27] decides the video level to download based on a dynamical model of the system and on the available bandwidth predictions over a given time horizon. With this information, an optimization problem is solved at run-time at each decision step, i.e. every time a segment is downloaded. Although each of the aforementioned approaches provides a different and unique solution to ultimately improve the user's experience, the performance of all of them strongly depends on the setting of their parameters. For instance, in FESTIVE [16], the parameter p helps to tolerate the buffer fluctuation caused by variability in chunk sizes while the trade-off factor α influences the efficiency and stability of the controller. Moreover, similarly to MPC, it is necessary to set the width of the prediction horizon, i.e. the number of bandwidth samples that should be used to perform bandwidth estimation and, in this regards, in [21] the authors show that when throughput predictions are incorrect, MPC's performance can decrease significantly. Additionally, BOLA [23] requires to set the parameters V to manage the trade-off between the buffer size and the optimal utility and γ as to properly weight utility and smoothness. Finally, PANDA [19] has a total of six parameters, among which the probing convergence rate k, the smoothing convergence rate α, and the safety margin ε are those that affect instability the most.

PENSIEVE [21] takes a drastically different approach and designs a model-free reinforcement learning algorithm to dynamically select the video level bitrate. In particular, based on measured features, such as the estimated bandwidth samples and the video-related features including future video segment sizes and the current state of the buffer, the learning algorithm drives a neural network with the ultimate goal of maximizing the reward, namely the QoE measured *a-posteriori*.

Recently, a novel technique for auto-tuning ABR controller has been presented. Similarly to ERUDITE, Oboe [2] dynamically adapts the parameters allowing the controller to perform better across a wide range of network conditions. Compared to the system proposed in this paper, there exist noticeable differences we would like to point out. Firstly, Oboe clusters multiple bandwidth dynamics into network states that differ only by mean and variance. Next, for each of such clusters, it pre-computes the set of parameters maximizing the QoE. During the online stage, Oboe detects which of these states best approximate the experienced bandwidth evolution and updates the controller's parameters accordingly. In contrast, ERUDITE leverages bandwidth dynamics and video features and, therefore, it is able to capture subtle differences which might not be evident by only looking at mean and variance. Moreover, Oboe schedules a parameters update each time it detects a change in the network state, ignoring the buffer evolution. Instead, ERUDITE continuously monitors the system performance in order to provide updates only when the underlying ABR controller fails.

Finally, an interesting and different approach based on game theory is proposed in [4, 5]. Authors cast decisions to be taken by the adaptive streaming controller to a bargaining process and consensus problem. The consensus point is reached through optimal decisions considering several network conditions and QoE metrics. An extensive experimental evaluation shows that the proposed algorithm is able to obtain promising results in terms of QoE in a large set of scenarios.

3 SYSTEM OVERVIEW

In this section, we propose a methodology to provide (near) optimal QoE for adaptive video streaming delivery systems. Indeed, the setting of the controller's parameters defines the way the system dynamically reacts to time-varying scenario features. Notice that

classical adaptive streaming controllers typically keep these values constant during the entire streaming session. It follows that it is not possible to find a unique set of controller's parameters performing well in a wide number of scenarios. To give a concrete example, consider Figure 1 that shows the contour lines of the functions mapping the parameters' values to the obtained QoE for two different scenarios. These maps have been obtained by using the optimization process that will be described in details in Section 4 with respect to a controller with four parameters. The figure clearly shows that the optimal set of parameters obtained for the scenario A (marked with a star in Figure 1a), provides very poor performances if applied to the scenario B (Figure 1b). Moreover, the mapping for scenario A has low variance, meaning that parameters' values set in the neighborhood of the optimal lead to a small performance drop. On the other hand, scenario B is clearly more challenging and small perturbations in the optimal parameters lead to severe performance degradation.

In particular, we propose a system to optimally tune – in the sense that will be defined below – the adaptive streaming controller's parameters to adapt to varying scenario's parameters. In this paper, we employ the term *scenario* to refer to parameters influencing system performances which fall outside the *design parameter space* (the controller's parameters). In particular, we consider the features of the user selected video clip and the available bandwidth time-evolution as the key scenario parameters. In this paper, we do not consider other parameters such as the user device resolution and context features.

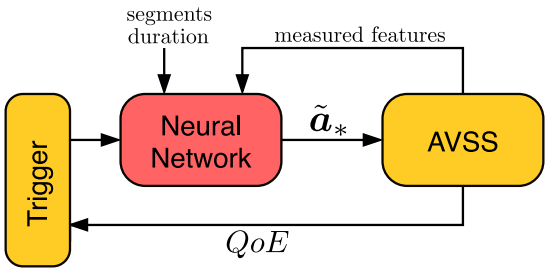


Figure 2: ERUDITE architecture. A neural network acts as a supervisor and dynamically tunes the parameters of the AVSS' controller to some values $\tilde{a}_*$ to improve the QoE.

Figure 2 shows the architecture of the proposed system. Firstly, the parameters of the *Adaptive Video Streaming System (AVSS) controller* are initialized to some default values that are known to ensure the stability of the system. Then, the streaming session starts and the client continuously fetches video segments (or *chunks*) from the server at a video level $l_i \in \mathcal{L}$ selected by the AVSS control algorithm. The download of new chunks proceeds until the *playout buffer* is full enough and the *player* can start to drain it mitigating the risk of incurring in rebuffering events. At this point, the player begins to render the downloaded frames and, simultaneously, evaluate the AVSS performances based on measured objective QoE-related metrics. A *performance drop* buffer measures the achieved performance in a time window of duration W and cumulatively stores the error committed with respect to the expected highest performance for the

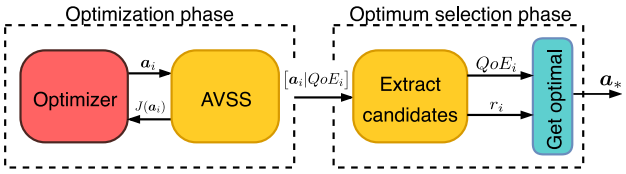


Figure 3: Optimal computation procedure employed to build the ground truth dataset.

experienced scenario. A *Trigger* module monitors the cumulative error and, if necessary, activates the *Neural Network* (NN) to update the controller's parameters to some values $\tilde{a}_*$ which approximate the optimal parameters.

Deciding when to activate the NN is not trivial. Obviously, the NN has to be activated only after the reception of new bandwidth measurements to provide it with different features to work on. However, we argue that it is not advisable to trigger the NN at each received bandwidth sample for two main reasons. First, the dynamics of the system takes some time to settle and, consequently, it is not appropriate to update the set of parameters before the system extinguishes its transient; this implies that the time interval between two consecutive updates must be larger than a fixed threshold, i.e., a minimum number of bandwidth samples should be collected before enabling an update. Second, too frequent updates might cause the system to overfit to bandwidth evolution.

Additionally, the selection of the NN input features has to be carried out carefully. To make an example, bandwidth samples are clearly important features having a remarkable impact on performances. However, the *number* of samples to feed the NN with is a non-trivial hyperparameter. Considering a small number of samples reduces the number of features and thus accelerates the forward propagation process. However, taking (too) few samples would make the system overfit to the bandwidth noise as fewer bandwidth samples would be evaluated to predict which set of parameters might improve the performances in the near future. Similarly, too many samples would fail in coping with scenarios in which sudden changes in the available bandwidth occur since a larger number of past samples would shadow recent bandwidth evolution.

Next, the training process of the NN requires to determine for each training sample, i.e., the scenario, the corresponding ground truth. In our case, we consider as the ground truth the optimal controller's parameters for a given scenario. As already shown in Figure 1, identifying the mapping between the controller's parameters and the corresponding QoE is generally a challenging task regardless of the employed AVSS controller. Therefore, we strive to obtain an approximation of such a mapping by using an optimizer. Then, we leverage this approximation to find the optimal parameters without performing a complete search in the parameters' space which is practically unfeasible in most cases. The process is summarized in Figure 3. In the *optimization phase*, the optimizer probes system's performance J on multiple sets of controller's parameters $\boldsymbol{a}_i$. At each iteration, the optimizer refines its understanding of the underlying mapping $\boldsymbol{a}_i \mapsto J(\boldsymbol{a}_i)$ by evaluating the system on a set

around which it has not enough information yet. At the end of this phase, a sequence of n sets of controller's parameters $\boldsymbol{a}_i$ is returned together with the corresponding performance $J(\boldsymbol{a}_i)$.

As the neural network expects a unique ground truth for each training sample, further processing is necessary to identify which one, among the n sets of parameters, achieves the best performance. We argue that this goal cannot simply be reached by naively choosing the parameter set with the highest reward (i.e., $\boldsymbol{a}_* = \text{argmax}\, J(\boldsymbol{a}_i)$). In fact, with such a choice, performances are likely to be very sensitive to small perturbations in the scenario parameters. Instead, in the *optimum selection phase*, we extract $n' < n$ candidates and, for each of them, we compute a robustness index r_i estimating the performance sensitivity to perturbations of the scenario's parameters. Then, we use as the ground truth the candidate $\boldsymbol{a}_*$ with the best compromise between *QoE* and robustness.

Finally, we split our generated ground truth dataset into train, test, and validation sets. Each of them is, furthermore, split into five subsets, each one holding scenarios where video traces have segments with the same duration. We consider and compare the performance obtainable with two NN architectures, a *Multilayer Perceptron* (MLP) and a *Convolutional Neural Network* (CNN). For each of these architectures, we train five neural networks, one for each considered segment duration (ranging from 2 s up to 10 s). We have decided to split the dataset in such a way since segment duration is constant throughout the duration of a session. Moreover, the segment duration is known to have a remarkable impact on performances.

4 BUILDING THE GROUND TRUTH

In this section, we discuss the methodology we propose to compute *off-line* the ground truths for each considered scenario. First, we briefly introduce the adaptive streaming controller (Section 4.1). We next present the adopted optimizer (Section 4.2) and the metrics that contribute to the definition of the optimal (Section 4.3). Finally, we describe how the optimal sets are selected to build the ground truth dataset (Section 4.4).

4.1 The AVSS controller

In this work, we have decided to use the hybrid buffer-based/rate-based adaptive streaming controller ELASTIC [12]. Notice that this choice does not affect the generality of the methodology proposed in this paper. In fact, as previously mentioned in Section 2, ERUDITE can be used to optimally tune any adaptive streaming controller compliant to the DASH standard.

ELASTIC control law is defined as follows:

$$l(t_k) = \begin{cases} l(t_{k-1}) & q_L \le q(t_k) \le q_H \quad (1) \\ \left[\!\!\left[\dfrac{b(t_k)}{1 - k_p e(t_k) - k_I e_I(t_k)} \right]\!\!\right] & \text{otherwise} \quad (2) \end{cases}$$

where $b(t_k)$ is the estimated available bandwidth at the end of the download of the k-th segment and $[\![\cdot]\!] : x \mapsto l_i$ is a quantizer function mapping x to the closest video level $l_i \in \mathcal{L} = \{l_1, \ldots, l_L\}$ ($l_i < l_{i+1}$) which is lower than x. The error $e(t_k)$ is given by

$$e(t_k) = \begin{cases} q_L - q(t_k) & q(t_k) < q_L, \\ q_H - q(t_k) & q(t_k) > q_H, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that $e_I(t_k)$ is the cumulative sum of the past values of the error $e(t_k)$ and defined as

$$e_I(t_k) = \begin{cases} 0 & q_L \le q(t_k) \le q_H, \\ \sum_k (t_k - t_{k-1}) e(t_k) & \text{otherwise.} \end{cases}$$

In a nutshell, the algorithm works as follows: as long as the playout buffer level stays inside the hysteresis ($q_L \le q(t_k) \le q_H$), the video level is kept constant (eq. (1)) to contain the amount of video level switches which is known to have an adverse effect on the QoE. When $q(t)$ gets outside the hysteresis, the controller sets the video level according to (2). Notice that (2) aims at steering $q(t)$ towards the hysteresis when the playout buffer length is outside of it. Thus, at "steady state", if the available bandwidth stays roughly constant, it turns out that the queue length is confined in the hysteresis and the video level switches between the two adjacent levels which are closer to the available bandwidth [10]. An important consequence of this property is that ELASTIC ensures that the average video level bitrate matches the average available bandwidth [12].

The resulting system dynamics depends on the settings of four non-negative parameters: the proportional and integral constants, k_p and k_I respectively, and the two hysteresis thresholds q_L and q_H. Intuitively, the higher q_L the less the chances to incur in rebuffering when abrupt bandwidth drops occur. The distance between the two thresholds, i.e., the hysteresis width, affects the responsiveness of the controller to bandwidth variations. Setting a large hysteresis width $\delta = q_H - q_L$ delays video level changes and contains the number of video switches. The setting of k_p and k_I affects the controller dynamics when the queue is outside the hysteresis (2). The larger k_p and k_I the faster the response to changes of the playout buffer length.

In order to ensure system stability, all the parameters defined above must be positive, i.e. the feasible parameter's domain set is the positive orthant of $\mathbb{R}^4$. However, regarding the low hysteresis threshold q_L we argue that this should be no smaller than one segment duration in order to ensure a minimum of one segment is stored in the buffer, i.e., $q_L \ge \tau$. Similarly, the hysteresis should not be smaller than one segment duration since lower values would rule out the effect of the hysteresis. It follows that the domain set where the optimization procedure shall find the optimal values of parameters is defined as:

$$\mathcal{D} \coloneqq \{(k_p, k_I, q_L, \delta) \in \mathbb{R}^4 | k_p \ge 0, k_I \ge 0, q_L \ge \tau, \delta \ge \tau\}. \quad (3)$$

with τ being the segment's duration of the scenario experienced.

4.2 The optimization phase

The optimizer we have used is the SAFEOPT implementation of the Bayesian Optimization Algorithm [6, 24], in its version named SWARM SAFEOPT [13] which further speeds up the entire optimization process while keeping RAM usage low. SAFEOPT requires the definition of a reward functional measuring the performance

obtained when tuning the controller with the chosen set of parameters. Additionally, the algorithm allows defining safety conditions which have to hold to consider a set of parameters *safe* to be used. Defining safety conditions restricts the search to those regions of the parameters' space that are considered safe.

In our case, the reward is defined as the normalized *QoE* (QoE_{norm}). We consider a set of parameters to be safe if the fraction of the streaming session during which the buffer length has been higher than the low threshold q_L is above 80%. This safety condition allows excluding cases exhibiting a high reward yet having very small buffer lengths for a large fraction of the streaming session.

Given a scenario, at the i-th iteration of the optimization process, SAFEOPT returns a set of safe parameters $\boldsymbol{a}_i = [k_p, k_I, q_L, \delta]^\top \in \mathcal{D}$, where $\mathcal{D}$ is the set defined by (3). Then, the streaming session is run with respect to the current scenario and the set of parameters $\boldsymbol{a}_i$. Finally, the evaluated QoE_{norm} is fed back to SAFEOPT which is used to update a Gaussian Process (GP), meant to be an estimation of the underlying relation between parameters and performance. At the end of this process, SAFEOPT produces a new set of parameters $\boldsymbol{a}_{i+1}$ to be used in the next iteration. During the optimization phase, we compute two metrics for each $\boldsymbol{a}_i$: i) a measure of how the system performed, QoE_{norm} (see Section 4.3), and ii) a measure quantifying the sensitivity of the obtained QoE performance with respect to variations of the scenario, i.e., the estimated *robustness* of $\boldsymbol{a}_i$ (see Section 4.4).

4.3 QoE definition

The QoE is estimated by using the functional defined in [8]

$$QoE(Q, F, S) = 4.85Q - 4.95F - 1.557S + 0.5, \tag{4}$$

where: i) Q is the average downloaded video bitrate normalized with respect to the video level l_L having the highest bitrate; ii) F is the penalty due to rebuffering episodes defined as in [8]; iii) S is the video level switching penalty. In particular, Q is computed as follows:

$$Q = \frac{\mathbb{E}[l(t_k)]}{l_L}.$$

F is defined as the linear combination of two terms: the first one depends on the frequency $b_{\texttt{freq}}$ of rebuffering events during the entire streaming session; the second one is based on the average duration $\overline{b}_{\text{time}}$ of such rebuffering events:

$$F = \frac{7}{8} \cdot \max\left(\frac{\log(b_{\texttt{freq}})}{6} + 1, 0\right) + \frac{1}{8} \cdot \frac{\min(\overline{b}_{\text{time}}, 15)}{15}$$

Finally, the switching cost S is computed as follows:

$$S = \frac{N_{\text{sw}}}{N_{\text{chunks}}} \cdot \frac{1}{N_{\text{sw}}} \sum_{i=1}^{N_{\text{sw}}} \frac{\Delta l_i}{l_L - l_0}$$

S is thus the product of two terms: the first one is the normalized number of switches, i.e. the ratio between the number of video level switches N_{sw} and the downloaded chunks N_{chunks}; the second one is the normalized average bitrate excursion covered at each switch.

While (4) is suitable for evaluating how different sets of parameters perform with respect to a given scenario, normalization is needed to compare the performance across different scenarios. In fact, Q depends on the scenario's average available bandwidth $\mathbb{E}[b(t_k)]$: a large value of $\mathbb{E}[b(t_k)]$ is likely to lead to a higher value of Q. However, it is straightforward to show that it is not possible for the controller to obtain a Q larger than $Q_{\max} = \mathbb{E}[b(t_k)]/l_L$ without eventually incurring in rebuffering episodes. Thus, the maximum achievable *QoE* for a given scenario can be obtained by substituting in (4) $Q = Q_{\max}$, $S = 0$, and $F = 0$ (no penalty due to switching and rebuffering):

$$QoE_{\max} = 4.85Q_{\max} + 0.5 \tag{5}$$

Finally, we define the *normalized QoE* as

$$QoE_{\text{norm}} = \frac{QoE}{QoE_{\max}}. \tag{6}$$

From now on, we will refer to QoE_{norm} as *QoE* for the sake of notational conciseness.

4.4 The optimum selection phase

The measured QoE alone is not sufficient to properly determine the ground truth. In fact, the real system will have to deal with a time-varying scenario, therefore, the sensitivity of the controller's parameters towards variations of bandwidth traces and video features must be as small as possible. To compute the robustness r_i associated to each parameter set $\boldsymbol{a}_i$, we have adopted the following approach.

Let us consider the j-th scenario s_j for which SAFEOPT has explored a number of parameters $\boldsymbol{a}_i^{[j]}, i = 1, 2 \ldots, N$. In the following discussion, we drop the apex $[j]$ for notation brevity. Let QoE_i, Q_i, F_i, S_i be the performance metrics corresponding to the parameter $\boldsymbol{a}_i$. Then, for each considered metric we construct the following sets, denoted as *metrics sets,* to discard points not achieving sufficiently high performances in any of the aforementioned metrics: $\mathcal{S}_{QoE} = \{\boldsymbol{a}_i \mid QoE_i \geq P_{90,QoE}\}$, $\mathcal{S}_Q = \{\boldsymbol{a}_i \mid Q_i \geq P_{90,Q}\}$, $\mathcal{S}_F = \{\boldsymbol{a}_i \mid F_i \leq P_{10,F}\}$, $\mathcal{S}_S = \{\boldsymbol{a}_i \mid S_i \leq P_{10,S}\}$ where $P_{x,S}$ denotes the x-percentile computed over the metric M.

Next, we compute the robustness of $\boldsymbol{a}_i$. Strictly speaking, in order to evaluate how $\boldsymbol{a}_i$ performs with respect to variations of the scenario it has been computed for, K noisy versions of the scenario should be computed, with noise affecting both the bandwidth trace and the video segments sizes. However, such an approach is extremely time-consuming since, for each $\boldsymbol{a}_i$, K new streaming sessions should be run and evaluated. We found that, instead of keeping $\boldsymbol{a}_i$ fixed and perturbing the scenario, comparable results can be achieved by tackling the dual problem. For each $\boldsymbol{a}_i$ we randomly generate 1000 new points $\boldsymbol{a}_{i,k} = \mathcal{N}(\boldsymbol{a}_i, \sigma_i^2)$ and evaluate performance on those while keeping the scenario unchanged[1]. The advantage of this approach is that we do not need to run any simulation. Instead, we leverage the Gaussian Process $\mathcal{G}$ iteratively refined by SAFEOPT, to get an estimate of the performance of $\boldsymbol{a}_{i,k}$. The robustness of each set of parameters is then given by:

$$r_i = c_i^2 \cdot \underset{k}{\text{MSE}}\big(QoE_i, \mathcal{G}(\boldsymbol{a}_{i,k})\big)$$

where $c_i \in \{0, \ldots, 4\}$ is the number of metric sets where $\boldsymbol{a}_i$ lives in, formally defined as

[1]Since each point is 4-dimensional, generating 1000 new random points means exploring the parameters' domain $\mathcal{D}$ by taking approximately 5 new values in each coordinate's direction.

$$c_i = \left| \left\{ \mathcal{A} \in \left\{ \mathcal{S}_{QoE}, \mathcal{S}_Q, \mathcal{S}_F, \mathcal{S}_S \right\} \,\middle|\, \boldsymbol{a}_i \in \mathcal{A} \right\} \right|.$$

Notice that c_i acts as a regularizer preventing points having a high score in only one or two of these metrics from being chosen as the ground truth. Finally, we promote to ground truth the point with the highest QoE among those whose robustness was higher than that of the 90-percentile:

$$\boldsymbol{a}_* = \text{argmax}\, QoE(\boldsymbol{a}), \;\; \text{with}\, \boldsymbol{a} \in \{\boldsymbol{a}_i \mid r_i \geq P_{90,r}\}$$

We do not follow the naive approach of choosing as optimum $\boldsymbol{a}_*$ the set of controller parameters with the highest robustness to exclude corner cases (that indeed do exist) in which points with very high robustness result in a poor QoE.

5 SCENARIOS DATASET CONSTRUCTION

In this section, we describe the scenario datasets. We define a scenario as the tuple $s_j = (T_j, \tau_j, v_j)$, where T_j is the bandwidth trace, τ_j denotes the video segments duration in seconds, and v_j is the considered video. The bandwidth traces are picked from a dataset that we have built by merging two publicly available datasets. The first dataset[2] contains 3G bandwidth traces collected in Sydney under vehicular driving conditions [7]. Bandwidth samples have an average of $\mu = 1518.35$ kbps and a standard deviation equal to $\sigma = 503.10$ kbps. We have also considered the bandwidth traces made publicly available in [25] which consider six means of transportation: foot, bicycle, car, tram, train, and bus. Average and standard deviation of this dataset are respectively $\mu = 3118.2$ kbps and $\sigma = 1464.0$ kbps.

Regarding the videos catalog, we downloaded the 23 video clips released in [22]. For each clip, five versions corresponding to five different segment durations are provided: 2 s, 4 s, 6 s, 8 s, 10 s. Each video is available at ten bitrates $\mathcal{L} = \{l_1, \ldots, l_{10}\}$ ranging from ~ 185 kbps up to ~ 4215.1 kbps. Then, to generate feature diversity, for each of these videos we have generated new videos by selecting five out of the ten available representations (i.e., $\mathcal{L}_{even} = \{l_2, l_4, l_6, l_8, l_{10}\}$ and $\mathcal{L}_{odd} = \{l_1, l_3, l_5, l_7, l_9\}$). At the end of this operation, we have obtained 345 videos using the dataset [22].

Moreover, we have built a dataset by fetching ~ 200 YouTube 4K videos with the same segment durations and 8 available levels (minimum video levels ~ 317.75 kbps, maximum video bitrates ~ 17322 kbps). This way, we ended up with a very diverse video catalog of roughly 1000 videos having different segment sizes and video level sets.

We have generated ~ 4000 scenarios by randomly picking videos and traces from the corresponding datasets. Finally, for each scenario s_j we have computed the optimal set of controller's parameters $\boldsymbol{a}_*^{[j]}$ by using the methodology presented in Section 4. Thus, a sample of our ground truth dataset is given by the tuple $x_j = (s_j, \boldsymbol{a}_*^{[j]})$. Finally, we split the ground truth dataset into training, test, and validation sets holding respectively ~ 2600, ~ 700, and ~ 700 samples.

[2] https://github.com/aubokani/Bandwidth-Dataset.git

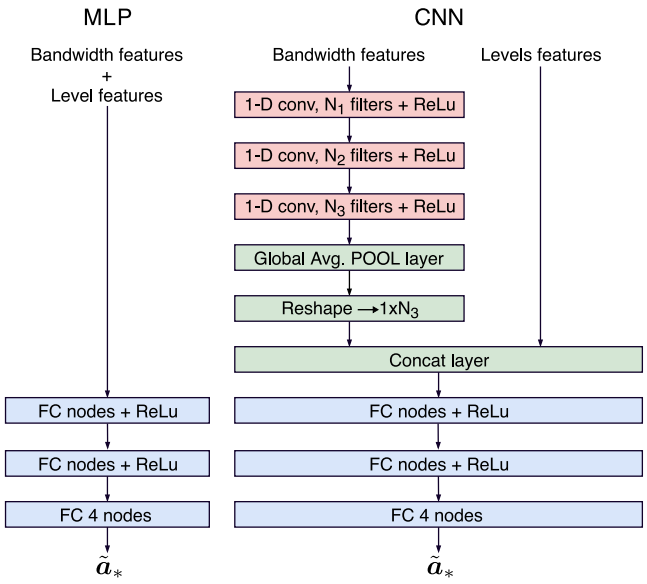


Figure 4: Architecture templates for MLP and CNN neural networks.

6 NEURAL NETWORKS TRAINING

In this section, we describe the dataset employed to train the neural networks. We have divided the input features into two categories: bandwidth samples features and video features. Concerning bandwidth features, we have considered the last 80 consecutive bandwidth estimates provided by the AVSS as inputs to the NN (see Figure 2). Regarding the video, we have considered the following features: (f_1) the nominal bitrate of the highest level l_L; (f_2) the nominal bitrate of the lowest level l_1; (f_3) the average of the relative distances (in bitrate) between nominal level bitrates, i.e. $f_3 = \frac{1}{L} \cdot \sum_{k=1}^{L-1} (l_k - l_{k-1})/l_k$; (f_4) the minimum, (f_5) the average and (f_6) the maximum of the coefficient of variation $c_{v,k} = \sigma_k / \mu_k$ per level bitrate, where μ_k and σ_k are the mean and variance of the video segments bitrate for the k-th video level, respectively. Therefore, the total number of features is 86. Finally, we implemented dataset normalization. Input features were normalized by Z-score. Normalization was done independently for the level-dependent features and as a whole for the bandwidth samples as they represent different outcomes of the same statistical process. Ground truths were normalized with the min-max normalization. We have employed TensorFlow [1] to implement and train the neural networks. The following NN architectures have been considered and compared in our work.

Multilayer Perceptron (MLP). For each layer, the Rectified Linear Unit (ReLu) activation function was used with the exception of the last layer where a linear activation was preferred to assess the regression task. We have also applied weight regularization using L_2-norm and a regularization strength equal to 0.001.

Convolutional Neural Network (CNN). We have modified the baseline neural network presented in [26] to fit our problem. Firstly, we have removed the Batch Normalization layers [15] since we have

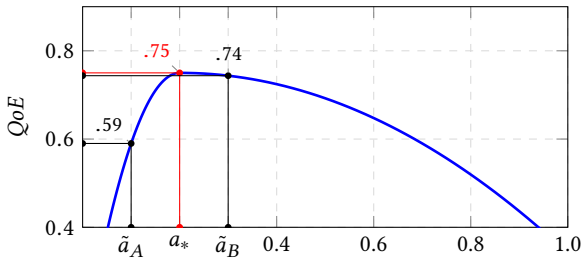


Figure 5: Example of a one-dimensional reward mapping.

found they did not help generalization in our specific case. This way, each *convolutional block* is a 1-D convolutional filter followed by the ReLu activation function as to introduce non-linearity. Secondly, the features extracted from the global average pool layer [20] were concatenated with the six level-dependent features (Figure 4). The remaining part of the network's architecture is a simple Multi Layer Perceptron.

On both network's architectures, we adopted the Adaptive Moment Estimation (ADAM) [18] as optimizer with a learning rate of 10^{-5} in the MLPs and 10^{-4} in the CNNs, $\beta_1 = 0.99$ and $\beta_2 = 0.999$, and – at each iteration – we shuffled the training set to improve generalization. We tested 300 network's hyperparameters configurations, 50 for the MLP and 250 for the CNN, for a total of 60 NN architectures for each of the five segment durations we have considered. Hyperparameters include the number of layers, the number of nodes per layers, the number of filters and filter sizes in the CNN, and the percentage of training samples given as batch input during training. Moreover, for some configurations, we have applied the *Savitzky–Golay* filter to bandwidth traces as to remove the intrinsic noise. To avoid overfitting, every 100 iterations of the training process, we estimated the generalization performance measuring the overall loss achieved on the test set.

6.1 Loss function

In the following we propose the Weighted Mean Squared Error (WMSE) that we employ to define the loss function. For the sake of illustration, let us consider the scalar case in which a function $QoE : \mathbb{R} \rightarrow \mathbb{R}$ maps the unique controller parameter a to the corresponding system performance $QoE(a)$ as shown in Figure 5. Suppose that two different predictions, $\tilde{a}_A$ and $\tilde{a}_B$, have been made with respect to the optimal value a_*. The prediction error exhibited by the NN can be expressed by the distance between the prediction and the optimal, i.e. $e(\tilde{a}, a_*) = |\tilde{a} - a_*|$. However, although the prediction errors are identical, i.e. $e(\tilde{a}_A, a_*) = e(\tilde{a}_B, a_*)$, the corresponding performances $QoE(\tilde{a}_A)$ and $QoE(\tilde{a}_B)$ are extremely different. Consequently, in this example, the performance drop would be much higher if the NN would have predicted $\tilde{a}_A$ instead of $\tilde{a}_B$ since the slope of the function in the left neighborhood of a_* is much higher than in the right counterpart. It follows that employing a simple MSE to evaluate the distance between the optimal sets and the predicted ones is not suitable to evaluate network performance. Thus, in order to get a better measure of the NN performance, information on the underlying reward function must be encoded so that each committed error is weighted proportionally to the estimated loss of system performance. To this purpose, we use the estimate of the reward function returned by SAFEOPT which has the form of a Gaussian Process $\mathcal{G}$. Let $\boldsymbol{a}_*$ be an optimal set of the controller parameters and a_i its i-th parameter. Suppose that $\boldsymbol{a}_*$ is normalized, i.e. $a_i \in [0, 1]$. We approximate the left and right partial derivatives of QoE at $\boldsymbol{a}_*$ with respect to a_i as follows:

$$w_i^L = \frac{\mathcal{G}(a_1, \ldots, a_i - h, \ldots) - \mathcal{G}(\boldsymbol{a}_*)}{h},$$
$$w_i^R = \frac{\mathcal{G}(a_1, \ldots, a_i + h, \ldots) - \mathcal{G}(\boldsymbol{a}_*)}{h}.$$

where $h = 5 \cdot 10^{-3}$. Thus, for each component a_i, we obtain one left weight w_i^L and one right weight w_i^R which quantify the loss of performance with respect to predictions $\tilde{a}_i$ having values lower or higher than a_i, respectively. With this setting, the loss function takes the following shape:

$$\mathcal{L}(\tilde{\boldsymbol{a}}_*, \boldsymbol{a}_*) = \sqrt{\left\| \lfloor \boldsymbol{a}_* - \tilde{\boldsymbol{a}}_* \rfloor \circ \boldsymbol{w}^L \right\|^2 + \left\| \lfloor \tilde{\boldsymbol{a}}_* - \boldsymbol{a}_* \rfloor \circ \boldsymbol{w}^R \right\|^2}$$

where $\lfloor \boldsymbol{x} \rfloor = [\max(0, x_i)]_{i=1,\ldots,4}$, $\|\cdot\|$ denotes the Euclidean norm, and $\boldsymbol{x} \circ \boldsymbol{y}$ denotes the element-wise product between two vectors of the same length, namely the *Hadamard product*.

7 RESULTS

This section investigates the performance achieved by the proposed approach. We divide this investigation into two parts. We start in Section 7.1 by evaluating the *open-loop case* to quantify how the proposed neural networks, trained with the procedure presented in Section 6, approximate optimal performances. Next, in Section 7.2 we discuss the results obtained by ERUDITE, i.e., when we close the loop with the NN which continuously provides estimates of the optimal parameters to the ELASTIC controller (see Figure 2).

7.1 The open-loop case

We compare the performance of the 300 NN architectures described in Section 6 with those achieved with two baseline approaches. The first one, named *random*, randomly picks a set of parameters from those contained in the train set of the corresponding segment duration; the second one, named *centroid*, assigns a set of parameters $\boldsymbol{a}_c^\tau$ for each segment duration τ. Each component of $\boldsymbol{a}_c^\tau$, i.e., $a_{c,i}^\tau$ with $i = \{1, \ldots, 4\}$, is the median computed over the i-th parameter in the optimal parameters' sets corresponding to that segment duration. We also assessed the *optimal* performances by evaluating, for each scenario s_j, the corresponding QoE (see eq. (6)) when setting ELASTIC with the optimal parameters $\boldsymbol{a}_*^{[j]}$ for that scenario.

For each strategy and each segment duration $\tau \in \{2, 4, 6, 8, 10\}$ s, we ran ~120 simulations using the scenarios of the validation set. The simulations have been carried out by implementing the ELASTIC controller (see Section 4.1) using the hybrid modeling framework presented in [9] that we have shown to provide very precise results compared to experiments with considerably shorter execution times.

Table 1 summarizes the hyperparameters' values which we have found to achieve the best performance. Unsurprisingly, we found that MLP models benefit from the use of the *Savitzky–Golay* filter,

Table 1: Best neural networks architectures.

Arch.	Hyper-param.	Segment duration τ				
		2 s	4 s	6 s	8 s	10 s
MLP	*Layers*	[48, 24, 16, 4]	[128, 64, 32, 16, 4]	[96, 48, 24, 8, 4]	[96, 48, 24, 8, 4]	[64, 32, 16, 8, 4]
CNN	*Filters size*	{8, 5, 3}	{12, 7, 5}	{12, 7, 5}	{8, 5, 3}	{12, 7, 5}
	Num. of filters	[32, 64, 32]	[32, 64, 32]	[32, 64, 32]	[32, 64, 32]	[32, 64, 32]
	Layers	[64, 32, 16, 8, 4]	[96, 48, 24, 4]	[48, 24, 16, 4]	[48, 24, 16, 4]	[128, 64, 32, 16]

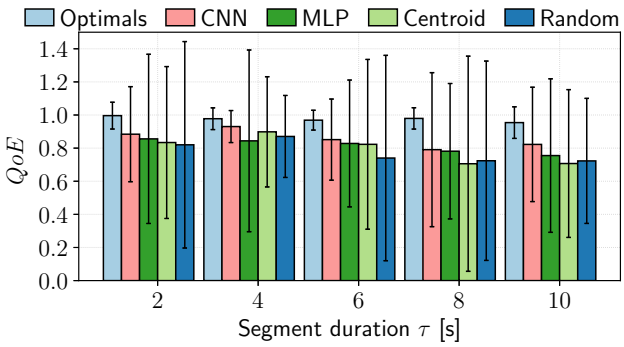


Figure 6: Average QoE as function of the segment duration

while most of CNN's best architectures perform a better bandwidth filtering through the 1-D convolutional layers.

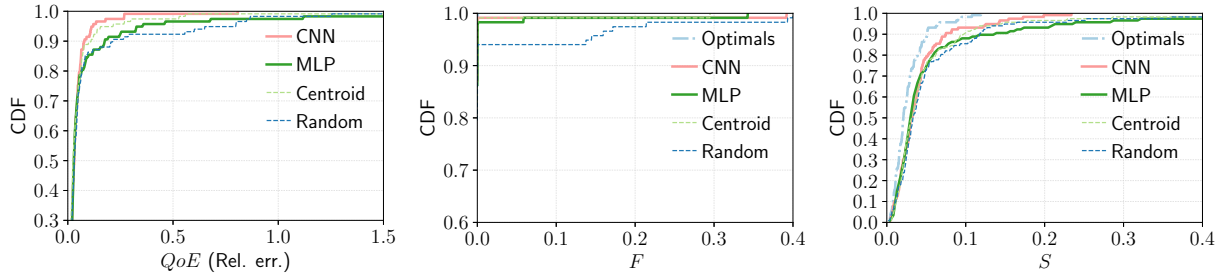
Figure 6 shows an overview of the obtained results in terms of the average QoE achieved by each strategy as a function of the considered segment durations. Confidence intervals are equal to one standard deviation. Notice that smaller confidence intervals indicate that the algorithm performances are more predictable and insensitive to different scenarios. Thus, if two algorithms would have comparable averages, the one with a lower confidence interval should be preferred. The figure illustrates that the performances achieved with the NNs outperform those obtained with *random* and *centroid* strategies. In particular, CNNs show the highest average QoEs for each segment duration together with the lowest standard deviations, which means that convolutional architectures are capable of guaranteeing sufficiently high performance when they fail to predict the correct set of parameters, more than centroid and random strategies. We have found that such a behavior is mostly due to the WMSE which pushes predictions away from regions where low performances are more likely to occur. Moreover, the figure clearly shows that – as expected – the average QoE decreases with increased segment durations. In fact, the lower the segment duration, the smaller the segment's size and, consequently, the more frequently the controller selects the video level for the next chunk, thus reacting to performance drops as soon as they occur. This means that an incorrect video bitrate selection due to an inefficient parameters' setting, can be managed better when segment duration is small, which makes such scenarios easier to deal with. We found that both *random* and *centroid*'s performances drop from approximately 0.85 to 0.70 as we move from segment durations of 4 s to 10 s. A similar behavior is experienced by MLPs and CNNs. However, the performance drop is less remarkable for NNs, so that, even for the toughest scenarios (i.e. those having segment duration $\tau = 10$ s), the average *QoE* does not drop below ~ 0.8.

Figure 7 shows the cumulative distribution functions (CDF) of the relative error on the average QoEs achieved from the considered strategies with respect to the optimal ones. Furthermore, we show the CDFs of the performance-related metrics, i.e., the rebuffering cost F and the switching cost S. Let us consider first Figure 7a and Figure 7d. The CNN achieves the lowest error on the average QoE in more than 10% of the scenarios when $\tau = 4$ s and in more than 30% in the case of $\tau = 10$ s so that CNN is able to provide satisfactory results even in challenging scenarios. Moreover, CNN rebuffering costs are comparable to those provided using *optimal* tuning in both cases (Figure 7b and Figure 7e), while *centroid* and *random* strategies perform poorly with videos having segment durations of 10 s reaching a cost of ~ 0.1 in 15% of the scenarios. MLP sits in between centroid and CNN resulting in rebuffering costs of ~ 0.1 in around 10% of the scenarios. Finally, CNN achieves the lowest switching penalty in more than 20% of the scenarios with segment durations of 4 s (Figure 7c) while, in the case $\tau = 10$ s, it outperforms the other strategies in virtually the entire validation set (Figure 7f).

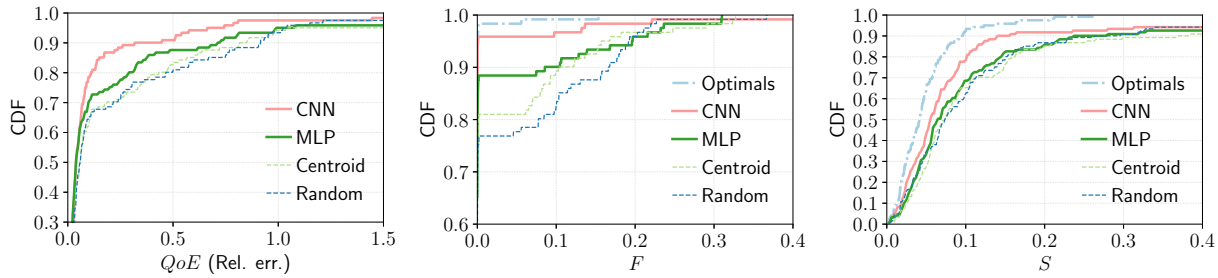
To summarize, this investigation has shown that among the considered strategies, the CNN is the one providing the best results in approximating the optimal performances.

7.2 The closed-loop case

7.2.1 Overall performances. We now investigate the performance of the closed-loop system shown in Figure 2. For this evaluation, we generated 40 new scenarios by uniformly sampling traces and videos from the datasets described in Section. 5. Each scenario has now a longer duration of 4000 s in order to assess the NN performances in the long term. Due to lack of space, we only report the results obtained on the evaluations carried out for segment durations equal to 4 s and 10 s in order to explore the performance bounds relatively to both fairly simple scenarios ($\tau = 4$ s) and more complex ones ($\tau = 10$ s). The *Trigger* module shown in Figure 2 evaluates the overall system performances at regular time intervals of length W, which we call *window width*. We have considered three window widths: 25 s, 50 s, and 100 s. At each evaluation, the performance error computed as $1 - QoE$ is cumulatively stored

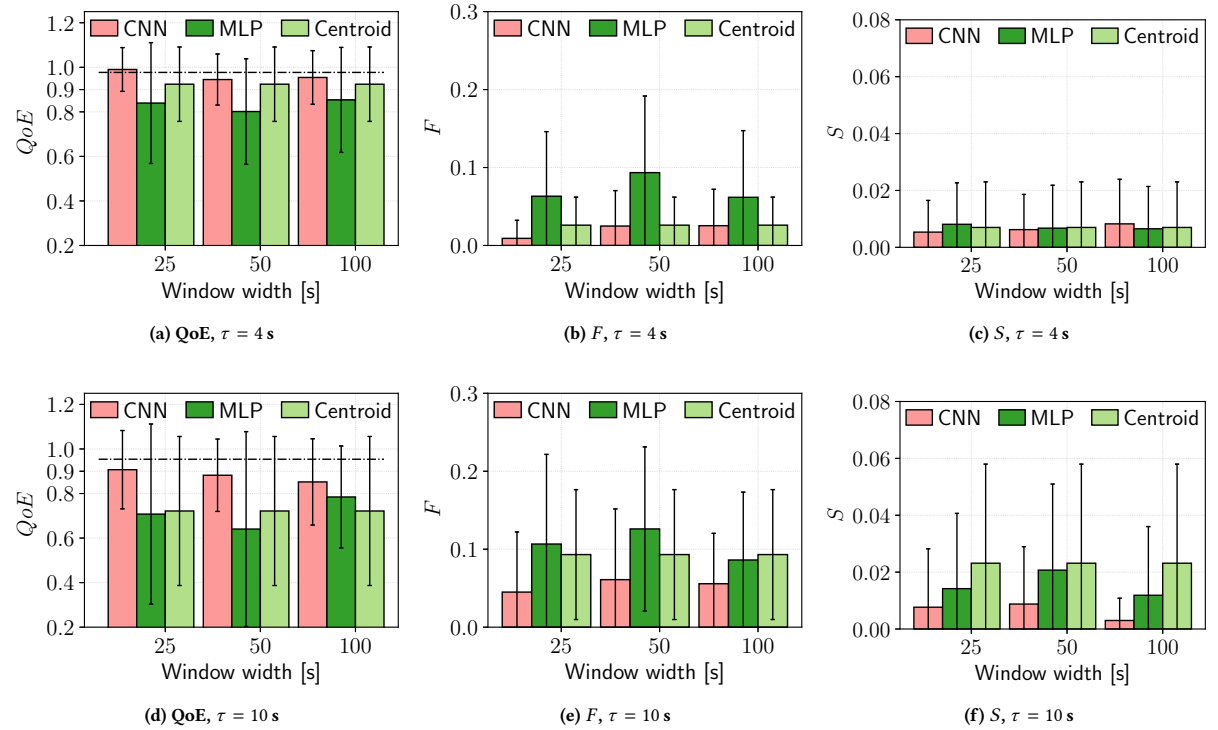


(a) QoE relative error: τ = 4 s (b) Rebuffering penalty: τ =4 s (c) Switching penalty: τ =4 s



(d) QoE relative error: τ =10 s (e) Rebuffering penalty: τ =10 s (f) Switching penalty: τ =10 s

Figure 7: Cumulative distribution functions of the QoE-related metrics in the case of videos with segment durations τ equal to 4 s or 10 s



(a) QoE, τ = 4 s (b) F, τ = 4 s (c) S, τ = 4 s

(d) QoE, τ = 10 s (e) F, τ = 10 s (f) S, τ = 10 s

Figure 8: Closed-loop performance function of the NN activation window width W and segment duration τ

in a performance drop buffer, so that, when this exceeds 0.1, the Trigger module activates the NN to provide the AVSS with a new set of parameters. At each evaluation, the performance drop buffer length is discounted by a discount factor $\lambda = 0.95$. This way, old performance fluctuations are discarded in favor of more recent feedbacks.

Figure 8 shows the obtained results[3] comparing CNN, MLP, and *centroid* performances. We have not evaluated the *random* strategy since we have already shown that it performs poorly even in the open-loop case. Let us consider the QoE results shown in Figure 8a and Figure 8d. In both the figures, the dash-dotted black lines mark the average performances obtained when setting ELASTIC with *optimal* parameters in the open-loop case. The figures show that, when the CNN is used with a window width equal to 25 s, it clearly outperforms the other strategies in the case of both small and large segments. In particular, when $\tau = 4$ s the CNN equates the results obtained by the optimal strategy in the open-loop case and improves its performances compared to the open-loop case (see Figure 6). This indicates that the CNN refines the parameters throughout the duration of the video session with the beneficial effect of improving the QoE. The CNN achieves remarkably good performances in terms of rebuffering, with a negligible measured rebuffering cost F. In the most challenging case of large segments ($\tau = 10$ s), the CNN achieves an average QoE of around 0.88, improving the performance obtained in the open-loop case, and getting very close to the optimal performance (the black dash-dotted line in Fig 8d).

Differently from the CNN, the best performing setting for the MLP has been obtained for W =100 s. We argue that in the case of the MLP when using small window widths overfitting occurs, whereas in the case of CNN overfitting is prevented by the convolutional filters that are applied to bandwidth features. In particular, MLP outperforms *centroid* in the case of large segments, but it performs slightly worse then *centroid* in the small segment case. Generally, MLP performance is affected by the higher rebuffering (see Figure 8b and Figure 8e) and level switching costs (see Figure 8c and Figure 8f) measured for both the cases of $\tau = 4$ s and $\tau = 10$ s.

7.2.2 Closed-loop system dynamics. We conclude our investigation by comparing the dynamics obtained in the case of segment size $\tau =$ 10 s with the CNN (window width equal to 25 s) with those achieved by *centroid* in the case of a YouTube video. Figure 9 compares the selected video level and the resulting playout buffer dynamics and shows the way the CNN dynamically selects the gains k_p and k_I. Let us start by analyzing the dynamics associated with the CNN (Figure 9a and Figure 9d). After a transient, the CNN starts to adapt the four ELASTIC's parameters (Figure 9b) and the selected video levels smoothly adapt to the time-varying available bandwidth as shown in Figure 9a. Notice that during the 4000 s of the simulation, only three very short rebuffering episodes occurred (see Figure 9d) resulting in a negligible rebuffering cost ($F = 10^{-5}$) and an overall QoE roughly equal to 1.0. The reason for the remarkably good performance obtained when using the CNN is the way parameters are dynamically changed. When the simulation starts, ELASTIC's parameters are initialized with *centroid* values. Then, at $t = 400$ s the CNN is activated (see Figure 9e) and new parameters are computed. In particular, as shown in Figure 9b, both k_p and k_I are set to very low values until $t \approx 1200$ s. This means that during the time interval [400, 1200] s the video level is computed only based on the estimated bandwidth (see eq. (2)). Notice that, due to this setting, during this time interval the queue gets large. Then, the CNN refines the setting by increasing only the value of k_p. This has an important consequence: ELASTIC now computes the video level by taking into account both the estimated bandwidth and the current value of the playout buffer level $q(t)$ (see eq. (2)). This reflects on the queue evolution that exhibits smaller oscillations above q_H. Compared to *centroid*'s setting, the CNN sets at steady-state the gain k_p to a higher value, whereas it turns off the integral action ($k_I \approx 0$).

On the other hand, by employing fixed parameters, *centroid* is not able to adapt satisfactorily to changed scenario's parameters. In fact, Figure 9c shows that the video level is changed too aggressively, overshooting the available bandwidth. Consequently, as shown in Figure 9f, a large number of rebuffering events occurs (rebuffering penalty $F = 0.14$) remarkably affecting performances, resulting in a measured QoE equal to 0.54.

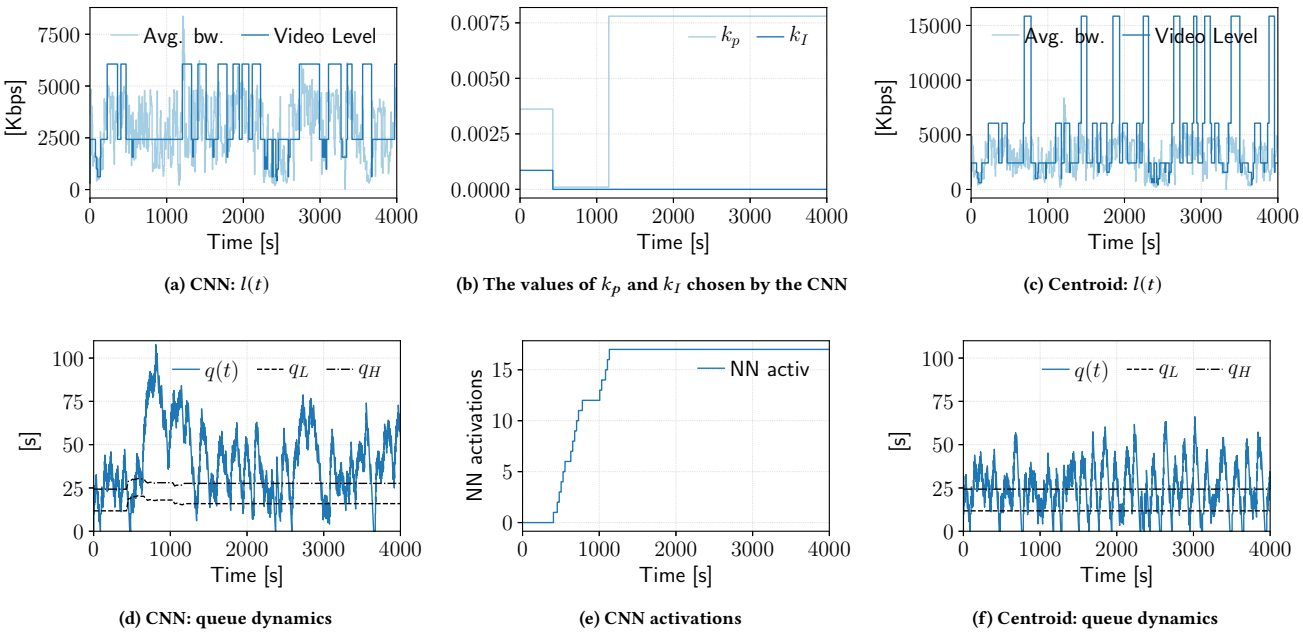
8 DISCUSSION

This section briefly discusses the way ERUDITE can be integrated in running streaming services.

Ground truth and (re-)training. The first step, which is executed off-line, for the integration of ERUDITE is to build the ground truth used for training the NN (Section 4). This entails finding optimal settings for a large number of scenarios. Scenarios could be built by either employing open datasets providing bandwidth traces and video traces or by leveraging, if available, the bandwidth traces and video clip features of already logged video sessions. Next, the optimization phase can be carried out by either running experiments in a controlled network environment (i.e., using network emulators and a real video catalog), or by employing accurate simulators such as the one used in this paper to speed-up such process. It is well-known that a large training dataset is preferable to ensure that the NN generalizes well across a wide range of scenarios. However, in practice, optimizing over a large number of scenarios might be too time-consuming, even in a simulated environment. Hence, we suggest to train ERUDITE on a reasonably sized dataset in the first place. Then, after deployment, ERUDITE's performance can be recorded and stored together with the scenarios experienced by users for future re-training. This approach allows to both expand the training set while the system is in production and to tackle those scenarios for which ERUDITE exhibited poor performance. Finally, the re-training process becomes necessary when the video clips features drastically change or when the QoE definition varies to emphasize different metrics.

Video client implementation. ERUDITE is fully compliant to DASH and does not require any server-side modification (i.e., standard HTTP servers can be used). In fact, ERUDITE acts externally to the adaptive video streaming algorithm placed at the client as it only updates at run time the controller's parameters (see Figure 2). This brings an important advantage: the control algorithm itself does not require any modification; however, an API function should be implemented to allow changing the parameters of the controller when requested. Furthermore, the video client should implement a module to measure and log all the variables that are needed to

[3]Confidence intervals are equal to one standard deviation.



(a) CNN: $l(t)$

(b) The values of k_p and k_I chosen by the CNN

(c) Centroid: $l(t)$

(d) CNN: queue dynamics

(e) CNN activations

(f) Centroid: queue dynamics

Figure 9: Closed-loop dynamics ($\tau = 10$ s). (a), (b), (d), (e): CNN dynamics. (c), (f): Centroid dynamics

compute the QoE metric and the features listed in Section 6 regarding the video clip and the estimated network bandwidth dynamics. Notice that this module is in practice already implemented by any well-engineered adaptive video streaming system.

ERUDITE is based on a NN to dynamically update the adaptive streaming controller's parameters when activated by the Trigger module. To this purpose, based on the pre-trained neural network and the measured features, the video client runs the inference phase that only requires to execute the forward propagation step. It is important to stress that, compared to black-box algorithms such as PENSIEVE which rely on a NN to run the adaptation algorithm at each segment download, ERUDITE has a much lower computational footprint since it activates the NN to compute the new controller's parameters only when the Trigger module has sensed a decreased QoE. To make an example, consider Figure 9e which shows that ERUDITE activated the NN only 17 times during a video session of 4000 s. Considering a segment duration of 10 s, PENSIEVE would have activated its NN 400 times. Finally, from the implementation point of view, several efficient javascript libraries exist to implement the forward propagation phase required by ERUDITE.

Future research directions. This work opens a number of future directions, which we briefly discuss here. First, it would be interesting to evaluate the performance improvements obtainable using ERUDITE in conjunction with other control algorithms available in the literature such as f.i., PANDA [19], BOLA [23], BBA [14]. Moreover, comparing ERUDITE with other adaptive streaming algorithms based on NNs such as PENSIEVE would also be interesting.

Network-assisted algorithms are known to bring performance improvements in terms of fairness and obtainable video quality [11]. As such, another orthogonal line of work that in our opinion is worth pursuing is to investigate to what extent including explicit feedback provided by network operators (such as 5G telcos) to the NN features would improve the accuracy of the NN in estimating the optimal parameters. Finally, while the optimization of a scenario can be executed offline as soon as all its parameters are collected, the training of the NN requires a higher computational effort. Regarding this issue, we plan to investigate on when re-training the NN is necessary as well as on how much additional data is required to effectively improve ERUDITE's performance.

9 CONCLUSIONS

This paper proposes ERUDITE, a system to provide users with near-optimal QoE for adaptive streaming systems. ERUDITE employs a Deep Neural Network (DNN) continuously providing the streaming controller with estimates of optimal parameters based on measured metrics such as bandwidth samples and the measured QoE. First, we evaluated the system in the open-loop case to quantify the accuracy of the proposed DNNs in providing the optimal performances. Results show that, between the proposed DNNs, the CNN is the best performing one and obtains QoEs differing from the optimal ones from 5% up 15% in the most challenging scenarios. We next closed the loop and measured the performances obtained when the DNNs provide the adaptive streaming controller with updated parameters to react to changed scenario's parameters. Again, CNN was the best performing strategy. Furthermore, results show that by closing the loop performances improve compared to the open-loop case. The CNN provides near optimal performances both in the case of small segments and in the more challenging case of large segment sizes.

ACKNOWLEDGMENTS

The authors would like to thank our shepherd Wei Wei and the anonymous reviewers for their valuable comments and helpful suggestions. This work has been partially supported by the Italian Ministry of Economic Development (MISE) through the CLIPS project (no. F/050136/01/X32). Any opinions, findings, conclusions or recommendations expressed in this work are the authors' and do not necessarily reflect the views of the funding agency.

REFERENCES

[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.
[2] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: Auto-tuning Video ABR Algorithms to Network Conditions. In *Proc. ACM SIGCOMM '18*.
[3] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a predictive model of quality of experience for internet video. In *Proc. of ACM SIGCOMM '13*.
[4] Abdelhak Bentaleb, Ali C. Begen, Saad Harous, and Roger Zimmermann. 2018. A Distributed Approach for Bitrate Selection in HTTP Adaptive Streaming. In *Proc. ACM International Conference on Multimedia (MM '18)*. 573–581.
[5] Abdelhak Bentaleb, Ali C Begen, Saad Harous, and Roger Zimmermann. 2018. Want to play DASH?: a game theoretic approach for adaptive streaming over HTTP. In *Proc. of ACM Multimedia Systems Conference (MMSys '18)*. 13–26.
[6] Felix Berkenkamp, Andreas Krause, and Angela P. Schoellig. 2016. Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics. *CoRR* abs/1602.04450 (2016). arXiv:1602.04450
[7] Ayub Bokani, Mahbub Hassan, Salil S. Kanhere, Jun Yao, and Garson Zhong. 2016. Comprehensive Mobile Bandwidth Traces from Vehicular Networks. In *Proc. ACM Multimedia Systems Conference '16 (MMSys '16)*.
[8] Maxim Claeys et al. 2013. Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming. In *Proc. Adaptive and Learning Agents Workshop*. 30–37.
[9] Giuseppe Cofano, Luca De Cicco, and Saverio Mascolo. 2016. A hybrid model of adaptive video streaming control systems. In *Proc. IEEE 55th Conference on Decision and Control (CDC)*. 6601–6606. https://doi.org/10.1109/CDC.2016.7799285
[10] Giuseppe Cofano, Luca De Cicco, and Saverio Mascolo. 2018. Modeling and Design of Adaptive Video Streaming Control Systems. *IEEE Transactions on Control of Network Systems* 5, 1 (March 2018), 548–559.
[11] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo. 2016. Design and Experimental Evaluation of Network-assisted Strategies for HTTP Adaptive Streaming. In *Proc. of ACM Multimedia Systems Conference (MMSys '16)*. 3:1–3:12.
[12] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. 2013. ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *Proc. 20th Packet Video Workshop (PV)*.
[13] Rikky R.P.R. Duivenvoorden, Felix Berkenkamp, Nicolas Carion, Andreas Krause, and Angela P. Schoellig. 2017. Constrained Bayesian Optimization with Particle Swarms for Safe Adaptive Controller Tuning. *IFAC-PapersOnLine* 50, 1 (2017), 11800 – 11807. 20th IFAC World Congress.
[14] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proc. ACM SIGCOMM '14*.
[15] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. International Conference on Machine Learning*. 448–456.
[16] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2014. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (TON)* 22, 1 (2014), 326–340.
[17] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi. 2015. SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *Proc. IEEE ICCW 2015*. 1765–1770.
[18] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980
[19] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 719–733.
[20] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network In Network. *CoRR* abs/1312.4400 (2013). arXiv:1312.4400 http://arxiv.org/abs/1312.4400
[21] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proc. ACM SIGCOMM '17*.
[22] Jason J. Quinlan, Ahmed H. Zahran, and Cormac J. Sreenan. 2016. Datasets for AVC (H.264) and HEVC (H.265) Evaluation of Dynamic Adaptive Streaming over HTTP (DASH). In *Proc. ACM Multimedia Systems Conference '16 (MMSys '16)*.
[23] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *Proc. IEEE INFOCOM '16*.
[24] Yanan Sui, Alkis Gotovos, Joel W. Burdick, and Andreas Krause. 2015. Safe Exploration for Optimization with Gaussian Processes. In *Proc. International Conference on Machine Learning*. JMLR.org, 997–1005.
[25] J. van der Hooft et al. 2016. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Comm. Letters* 20, 11 (2016), 2177–2180.
[26] Z. Wang, W. Yan, and T. Oates. 2017. Time series classification from scratch with deep neural networks: A strong baseline. In *Proc. International Joint Conference on Neural Networks (IJCNN)*. https://doi.org/10.1109/IJCNN.2017.7966039
[27] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proc. of ACM SIGCOMM '15*.

A SYMBOLS TABLE

Table 2: Symbols table

Symbol	Term	Meaning
$\mathcal{L}$	Video level set	Bitrates l_i of available representations of a given video clip.
T	Bandwidth trace	Time evolution of a bandwidth trace.
v	Video clip	Represents a video clip described by the segment sizes for each of the video levels.
τ	Segment duration	Duration of each video clip segment expressed in seconds.
s	Scenario	Defined by the tuple (T, τ, v).
$\boldsymbol{a}_*$	Optimal parameters	The set of the AVSS optimal parameters for a given scenario.
QoE	Estimated QoE	A functional estimating the QoE of a given video session.
Q	Average downloaded video bitrate	
F	Rebuffering penalty	Depends on both the rebuffering frequency and the cumulative rebuffering time
S	Switching penalty	
$\mathscr{L}(\tilde{\boldsymbol{a}}_*, \boldsymbol{a}_*)$	Loss function	It measures the drop of performance when $\tilde{\boldsymbol{a}}_*$is used instead of the optimal $\boldsymbol{a}_*$.
W	Window width	Time interval at which the Trigger module evaluates system's performance.