PERCEIVE: Deep Learning-based Cellular Uplink Prediction Using Real-time Scheduling Patterns

Jinsung Lee
University of Colorado Boulder
jinsung.lee@colorado.edu

Sungyong Lee
UNIST
sungyonglee@unist.ac.kr

Jongyun Lee
UNIST
jongyunlee@unist.ac.kr

Sandesh Dhawaskar Sathyanarayana
University of Colorado Boulder
sadh0344@colorado.edu

Hyoyoung Lim
University of Colorado Boulder
hyoyoung.lim@colorado.edu

Jihoon Lee
University of Colorado Boulder
jihoon.lee-1@colorado.edu

Xiaoqing Zhu
Cisco Systems
xiaoqzhu@cisco.com

Sangeeta Ramakrishnan
Cisco Systems
rsangeet@cisco.com

Dirk Grunwald
University of Colorado Boulder
dirk.grunwald@colorado.edu

Kyunghan Lee
Seoul National University
kyunghanlee@snu.ac.kr

Sangtae Ha
University of Colorado Boulder
sangtae.ha@colorado.edu

ABSTRACT

As video calls and personal broadcasting become popular, the demand for mobile live streaming over cellular uplink channels is growing fast. However, current live streaming solutions are known to suffer from frequent uplink throughput fluctuations causing unnecessary video stalls and quality drops. As a remedy to this problem, we propose PERCEIVE, a deep learning-based uplink throughput prediction framework. PERCEIVE exploits a 2-stage LSTM (Long Short Term Memory) design and makes throughput predictions for the next 100ms. Our extensive evaluations show that PERCEIVE, trained with LTE network traces from three major operators in the U.S., achieves high accuracy in the uplink throughput prediction with only 7.67% mean absolute error and outperforms existing prediction techniques. We integrate PERCEIVE with WebRTC, a popular video streaming platform from Google, as a rate adaptation module. Our implementation on the Android phone demonstrates that it can improve PSNR by up to 6dB (4×) over the default WebRTC while providing less streaming latency.

CCS CONCEPTS

• **Networks** → **Mobile networks**; **Cross-layer protocols**; **Network measurement**.

KEYWORDS

LTE; Cellular Uplink; Deep Learning; Live Video

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MobiSys '20, June 15–19, 2020, Toronto, ON, Canada
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7954-0/20/06…$15.00
https://doi.org/10.1145/3386901.3388911

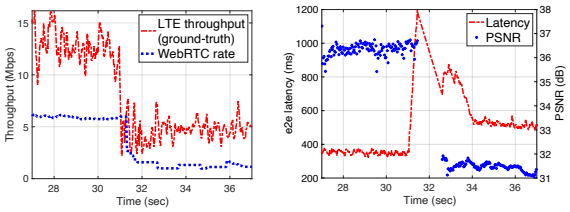
ACM Reference Format:
Jinsung Lee, Sungyong Lee, Jongyun Lee, Sandesh Dhawaskar Sathyanarayana, Hyoyoung Lim, Jihoon Lee, Xiaoqing Zhu, Sangeeta Ramakrishnan, Dirk Grunwald, Kyunghan Lee, and Sangtae Ha. 2020. PERCEIVE: Deep Learning-based Cellular Uplink Prediction Using Real-time Scheduling Patterns. In *The 18th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '20), June 15–19, 2020, Toronto, ON, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3386901.3388911

1 INTRODUCTION

A recent study unveiled that live video has reached near-universal adoption for social use as it becomes as pervasive as voice calls and easy to use with smartphones [59]. We can quickly realize it from the fact that there are a lot of mobile applications for video conferencing and personal live broadcasting in either Google's Play Store or Apple's App Store. As mobile live video streaming has grown substantially [16], the traffic demand over cellular uplink channels has been growing fast.

However, unpredictable throughput fluctuation in cellular uplink channels affects user experience significantly. Figure 1 illustrates such a problematic case from our measurement in WebRTC, which is the de-facto standard platform for web-based real-time streaming applications [29, 72]. We observe that abrupt throughput drop within a short time interval can lead to a surge of end-to-end delay along with sluggish rate adaptation in the transport layer, which, in turn, causes severe degradation of user experience (i.e., *video stall* and *quality degradation*). This example explains why small mistake in throughput estimation (at the moment around 31s) is crucial in practice, which motivates our work.

In this paper, we explore the potential of incorporating accurate uplink throughput prediction in cellular networks to improve user experience in the live video streaming. To this end, we leverage microscopic behaviors such as millisecond-level resource allocation patterns extracted from LTE chipset rather than tracking longer-term throughput observations in upper layers, which is typically



(a) LTE and WebRTC bitrate (b) WebRTC performance

Figure 1: WebRTC performance degradation due to abrupt throughput drop in LTE uplink.

used by existing prediction algorithms. Our approach is based on the fact that in LTE systems, the base station (BS) scheduler assigns available resources called RB (Resource Block) to multiple mobile devices with 1ms granularity in the time domain (detailed in §2).

Through extensive measurement in commercial LTE networks, we observe that there are several unrevealed correlations between short-term resource allocation patterns in a smartphone and cell load conditions in uplink channels of the serving BS. Our novel finding indicates that the history of the RB allocation patterns that is the consequence of cellular scheduling algorithms can be used for inferring up-to-date cell load status such as the amount of background traffic and the number of competing devices. By learning the above relationship with a neural network, we can accurately predict cellular uplink throughput even in highly dynamic network conditions. Besides, for the accurate short-term uplink throughput prediction, we let the neural network learn how to choose the best length of historical LTE-layer information from the degree of cell load dynamics (i.e., the speed of change in available throughput).

In particular, our deep learning-based framework PERCEIVE (Predicted pERformance by CEllular Inferring deVicE) utilizes Long Short-Term Memory (LSTM)-based neural network [32]. It is capable of handling long-term dependencies for time series prediction by maintaining past hidden information with memory. The characteristics of LSTM can let the neural network understand complex, relatively long RB allocation sequences, which have been set by the cellular scheduling algorithm that uses the input of active users with traffic demand and their historical service rates. As a result, PERCEIVE learns such implicit dynamics between network load and local RB assignment patterns over time, which cannot be successfully realized by other simple learning algorithms. We provide the detailed evaluations in §6.

The advantages of PERCEIVE are three-fold. First, it only exploits local information from the phone's cellular modem without any assistance from the BS and other mobile devices, i.e., *client-only approach*. Second, it is inherently *device-agnostic*, since the BS scheduler decides resource scheduling patterns for all types of served devices. Third, it is *applicable to different (even untrained) cellular networks*, since the networks employing Proportional Fair (PF) scheduling policy [10, 34, 69] are expected to reveal similar resource allocation patterns in uplink channels.

In summary, our paper makes the following contributions:

- We discover that there exist strong correlations between RB allocation patterns and network load conditions in cellular uplink channels (§3). Further, we find that the normalized variation in RB allocation can be used to infer the degree of cell load dynamics.
- We develop PERCEIVE that accurately predicts cellular uplink bandwidth using LSTM neural networks (§4). Our method can predict short-term throughput using the best length of historical LTE-layer information according to cell load dynamics. We demonstrate that PERCEIVE can predict the uplink bandwidth with the mean absolute error (MAE) of 7.67% over the whole dataset from LTE networks in the U.S.
- We design PERCEIVE-based rate adaptation for an open-sourced WebRTC platform (§5). We demonstrate via trace-driven simulations that PERCEIVE outperforms existing learning-based algorithms, and via implementation on the off-the-shelf phone that it can improve PSNR by up to 6dB (4×) over the default WebRTC algorithm while keeping smaller streaming delay in commercial LTE networks (§6).

2 BACKGROUND

2.1 Cellular network scheduling

LTE scheduling primer. In the LTE system, for either downlink or uplink, each mobile device is assigned by BS scheduler with RB, which is the smallest resource unit that spans across seven time-domain symbols (one slot, 0.5ms) and 12 frequency-domain subcarriers with 15kHz each. In the time domain, an LTE frame has ten subframes, each of which consists of two slots. In the frequency domain, there exist RBs whose total number depends on channel bandwidth (e.g., 50 RBs for 10MHz).

It is known that most existing LTE schedulers follow PF algorithm to fairly allocate radio resources among mobile devices while maximizing network resource utilization [10, 42]. The PF scheduler aims to maximize the utility function $\sum_i \log(R_i)$ over all feasible scheduling rules, where R_i is the long-term service rate of user i, and is updated as follows: $R_i(t+1) = (1-\epsilon)R_i(t) + \epsilon r_i(t)$ if user i is scheduled; $R_i(t+1) = (1-\epsilon)R_i(t)$ otherwise. $r_i(t)$ is the instant service rate for user i and ϵ is a weight factor reflecting history on the order of 1000 subframes ($\approx$ 1sec). Ideally, to maximize log utility sum, the scheduler should serve the user who maximizes $r_i^c(t)/R_i(t)$, say *PF metric*, on each RB c at each time t. Thus, $r_i(t) = \sum_c r_i^c(t), \forall i$.

Let's suppose that multiple devices are attaching to a BS with similar channel conditions. In this case, the PF metric can be dominated by $1/R_i$ rather than $r_i^c(t)$. This implies that the scheduler would prioritize resource allocations to either a newly arrived user (i.e., a user with $R_i = 0$) or existing user having intermittent traffic pattern due to its relatively small R_i value compared to users having constant traffic (e.g., streaming or bulk transfer). In other words, the device with continuous traffic would obtain remaining resources after the BS schedules other prioritized devices. Thus, a mobile device that knows its RB allocation patterns can infer the network condition that reflects the up-to-date traffic loads imposed by others. We investigate whether this intuition is valid in practice in §3.

Uplink scheduling procedure. Uplink scheduling differs from downlink counterpart since the scheduler is located in the BS, while the uplink data traffic is generated at mobile clients. If one mobile device has data to send, it sends a scheduling request (SR) at a

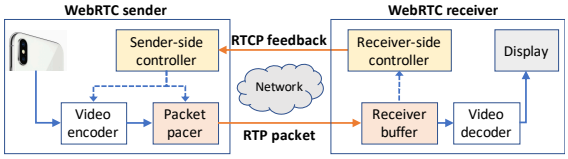


Figure 2: WebRTC system architecture.

specific interval, which is called SR periodicity [4], assigned by the BS. When the BS has available resources for the device's request, it allocates a scheduling grant to it. Upon receiving the grant from the BS, the client sends the granted amount of packets to the BS and then reports the amount of uplink data to be transmitted via a buffer status report (BSR).

2.2 WebRTC architecture

WebRTC uses UDP, over which it uses RTP (Real-time Transport Protocol) to send media packets of video and voice. It gets feedback from the receiver in the form of RTCP (RTP Control Protocol). For rate control, WebRTC uses the Google Congestion Control (GCC) algorithm by default in two ways: delay-based control at the receiver and loss-based control at the sender [33, 35], as shown in Figure 2. Specifically, the receiver calculates the one-way delay variation from the difference between inter-arrival time and inter-departure time using the transmission of two consecutive packets. Then, the receiver-side controller decides whether to increase, decrease, or hold the target receiving rate depending on the one-way delay variation. On the other hand, the sender-side controller computes the target sending rate based on the fraction of lost RTP packets whenever it receives RTCP feedback from the receiver [33].

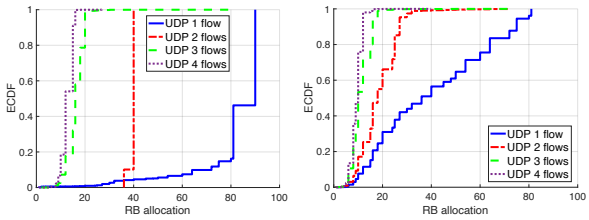
Upon detecting the bitrate change from the RTP layer, WebRTC rate control performs two tasks: packet pacing at the RTP layer and source rate adaptation at the application layer. That is, the sender adjusts the interval of enqueued packets for pacing to the network and triggers to update encoding parameters on incoming frames from the video source to the encoder. It is worth noting that it takes a long time for GCC to track available bandwidth due to its slow adaptation [35].

3 OBSERVATIONS

3.1 Measurement setup

To generate persistent uplink traffic, we run `iperf3` [1] at smartphones in LTE networks of three major U.S. operators. We use an SDR-based LTE sniffer, which consists of a high-end laptop, USRP B210 [20], and antennas supporting LTE frequencies, and run open-source OWL (online watcher for LTE) [11] to collect and decode cell-wide scheduling information. The LTE sniffer gathers device ID, number of RB allocation, TBS (transport block size)[1], and MCS (modulation and coding scheme) for each of devices attaching to the observed cell. At the same time, we run an online cellular monitoring tool at the phone, which we enhanced from the open-source analyzer MobileInsight [44] (detailed in §6.2). Thus, we can independently record client-side information such as RSRP (reference signal received power), cell ID, number of RB, and TBS.

[1]TBS is decided by the number of allocated RBs and MCS as specified in [4], which is the amount of data size that can be transmitted during one slot (0.5ms).



(a) Idle network **(b) Congested network**

Figure 3: RB allocation for one device with varying number of competing flows (one flow per device) over AT&T LTE network with 20MHz cell.

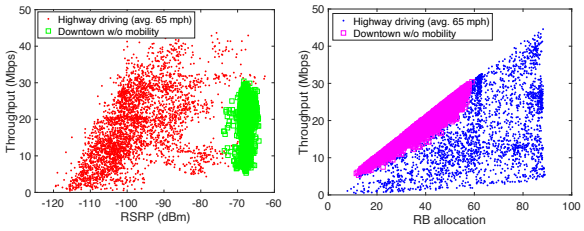
Also, LTE-layer uplink throughput is calculated by dividing the summation of TBS values over the time window of 20ms by the duration of that window.

3.2 Uplink throughput fluctuation

Impact of cell load. We vary the number of UDP flows (one per device) from one to four, and inject them to the network in the uplink direction. We then measure RB allocation over two different cell load scenarios: idle and congested, whose statistics are analyzed by LTE sniffer. Figure 3 shows ECDF (empirical cumulative distribution function) of RB allocations for one device with varying number of competing UDP flows. Depending on the cell load, we observe a clear difference in RB allocation patterns: when idle, it has small variation, while when congested, it has larger variation, irrespective of competing flows. Further, we find that different cell loads affect the throughput variation significantly. To quantify such throughput diversity, we apply the coefficient of variation (CV) metric that is defined as the ratio of STDEV to mean of samples [61]. The congested case shows much severe diversity ranging from 1.53× to 11.8×, compared to the idle one. We did not see any noticeable disparity between TCP and UDP.

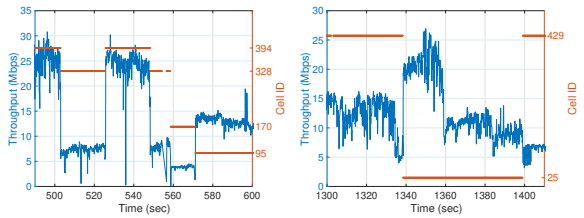
Impact of signal strength. The signal strength between the mobile device and the BS is one of the critical factors deciding MCS, thus affecting achievable cellular throughput [41, 48, 67]. To see the impact of the signal strength, we measure RSRP and throughput together with and without mobility. As shown in Figure 4, RSRP is correlated with throughput in the uplink channel, especially when the device is moving. Frequent handovers (every 26s on average) lead to dispersed patterns in RSRP and RB allocation values. However, more interestingly, we observe that there exists a stronger correlation between RB allocation and achieved uplink throughput. That is, while the device is stationary (i.e., with similar RSRP values), the uplink throughput varies significantly due to the variation of RB allocation in a short timescale. Note that as RSRQ (reference signal received quality) can be used for inferring downlink load utilization [41, 67], we cannot see any visible connection to uplink throughput.

Impact of cell switching. Besides, we observe that abrupt throughput variations frequently happen due to the change of serving cell during continuous upload while being stationary. Figure 5 shows several switching events during upload at the same location, which results in abrupt throughput changes (up to 10×) in AT&T and T-Mobile LTE networks, respectively. If the cell switching happens



(a) RSRP vs. Tput (b) RB vs. Tput

Figure 4: Throughput variation w.r.t. RSRP and RB allocation with and without mobility over AT&T LTE network.



(a) AT&T network (b) T-Mobile network

Figure 5: Throughput change by cell switching during uplink transmission at the same location.

between adjacent cells with different channel bandwidths, the variation is intensified. We conjecture that this operation is likely to happen for load balancing, which is hidden to mobile devices. For accurate throughput prediction against such unexposed events, we have to borrow the unmatched (or matchless) pattern recognition ability of a deep learning, detailed in §4. Lastly, we have not seen any carrier aggregation in uplink scheduling across LTE networks, compared to downlink counterpart [18].

3.3 Correlation between scheduled resource and cell-wide load in uplink

We study the correlation between RB allocation patterns and cell-wide load status in uplink channels. For the load status, we consider two metrics: (i) the amount of competing traffic in terms of RB utilization and (ii) the number of scheduled devices, which are measured as the ground-truth by our LTE sniffer. In particular, we obtain the cell load of competing traffic by summing RB values allocated to all active devices (except ours) in the same subframe from the stored trace, and the number of scheduled devices by counting the number of unique device IDs for the same duration. The information of our device can be identified in the cell-wide trace from the LTE sniffer via the device ID acquired in the online monitoring tool.

Our hypothesis is that since the BS scheduler distributes uplink resources among granted devices considering their BSR, each device can infer the cell load status from its own historical RB allocation patterns. In other words, the RB allocation trace gathered at each device contains a clue about the BS scheduling decisions based on up-to-date cell load (discussed in §2). As estimating the maximum available throughput of cellular links is challenging [64], we verify the hypothesis using long-lived (≥ 60s) UDP traffic to saturate the cellular uplink. The measurement of each scenario is performed in

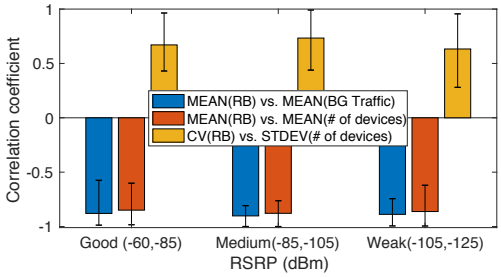


Figure 6: Correlation coefficient between RB allocation and cell load metrics w.r.t. RSRP values over AT&T LTE network; T-Mobile and Verizon show similar trends.

at least ten different locations across one metropolis and several downtown areas. During the data transfer, we collected the client's uplink RB allocation trace using the online monitoring tool as well as cell-wide scheduling information using the LTE sniffer, so that we could keep track of RB allocation patterns of our device in the cell-wide traces.

RB allocation and competing traffic. We investigate the correlation between allocated resources for our device and cell load utilization. To analyze this, we use RB traces of the phone and cell-level resource utilization from the LTE sniffer. Figure 6 shows a correlation result for the aggregation window of 1s over AT&T's LTE network. To remove the effect of the signal strength, we classify the results into three categories based on RSRP measurement. We found that there is a strong negative correlation, which indicates that the phone can infer the amount of resource utilization by competing traffic, given that it keeps injecting enough traffic.

RB allocation and the number of active devices. We study the correlation between allocated resources for the device and the number of actively contending devices in the same cell. In particular, we keep track of the trace of allocated RBs at our phone and the number of scheduled devices together. In all scenarios, we can see that there is a strong negative correlation between the two, as depicted in Figure 6. It is natural because competing traffic is generated by granted devices and the uplink scheduler needs to distribute resources to them in a fair manner to timely serve those clients, which is different from downlink scheduling that reveal bursty allocation patterns to a few clients [8].

RB allocation and cell load dynamics. To understand cell-wide load dynamics by uplink scheduling, we investigate the variation of RB patterns. We use an aggregated time window, during which the standard deviation (STDEV) of allocated RBs for our phone is calculated. As discussed in §2, we expect that such RB variations originate from the variation of active devices that affects scheduling decisions done by BS. However, as different cells can have different bandwidths, we apply the CV metric to the RB allocation.

Figure 6 shows the correlation results between the CV of RB allocation in our device and the STDEV of the total number of scheduled devices in the uplink. We can see that there is a reasonably strong positive correlation. More importantly, we found that the CV value varies due to the variation of scheduled devices that can represent the degree of variation in the cell load status. That is, when contending devices served by the same cell join and leave more frequently, it leads to an increase of STDEVs of both

Scenario	Location	Time(hour)	Traffic(GB)	Cell IDs(#)
Stationary	Downtown	16.8	98.3	24
Stationary	Entertainment	7.8	49.3	34
Stationary	Office	3.7	16.4	26
Stationary	Residence	13.0	33.7	27
Low mobility	Downtown	5.6	28.5	208
High mobility	Highway	1.5	8.0	117

Table 1: Dataset from commercial LTE networks

scheduled devices and RB allocation for our device. Thus, if the cell experiences more number of arrivals and departures of scheduled devices with more traffic, the CV in our device can be large.

4 PERCEIVE DESIGN

In this section, we describe the system design of PERCEIVE that predicts cellular uplink throughput using a 2-stage neural network considering cell load dynamics.

4.1 Data collection

We performed extensive measurements on LTE-layer information during four months in 2019 over commercial LTE networks. We obtained the low-level data using our online monitoring tool while sending UDP traffic at the phone enough to saturate the network. The rationale of using saturated traffic is that our goal is to predict maximum available throughput in uplink channels, which is known to be challenging due to the closed architecture in cellular networks [8, 64]. Also, the network saturation has been used for throughput prediction in other works [69, 70]. Table 1 lists the details of our dataset. The whole dataset covers a wide range of scenarios (from stationary to highway driving), including different phones (Pixel XL and Nexus 6P), cellular operators (AT&T, T-Mobile, and Verizon), times and locations, and corresponds to the duration of 48.5 hours and the data upload of 234.2 GB. The total number of observed cell IDs is 436.

4.2 Prediction framework

Input and output. PERCEIVE predicts future uplink throughput during next 100ms with past LTE-layer information such as RB allocation, TBS, and RSRP. Instead of the original RB_{sample}, we exploit the RB allocation ratio defined by $\frac{RB_{sample}}{RB_{\max}}$ for normalization between BSs with different channel bandwidths. Based on the reported channel bandwidth, we can figure out $RB_{\max}$ in advance. The length of input features is set to 1s because the correlation coefficients become rather decreasing beyond 1s, after which it may contain more stale data (§3.3). We implement a data structure using a circular buffer so that the model can always access up-to-date data during past 1s, and each array is updated when the LTE information is received every 20ms. Finally, the predictor infers uplink throughput for the next 100ms.

2-stage inference model. We design our deep learning inference model as a 2-staged architecture, which consists of cell load dynamics recognition and uplink bandwidth prediction. The up-to-date degree of cell load dynamics is inferred using the whole input data gathered in the past 1000ms. Based on the classified cell load dynamics, the ITW (input time window) is selected among 100, 300, and 1000ms. For the uplink bandwidth predictor, we use the most

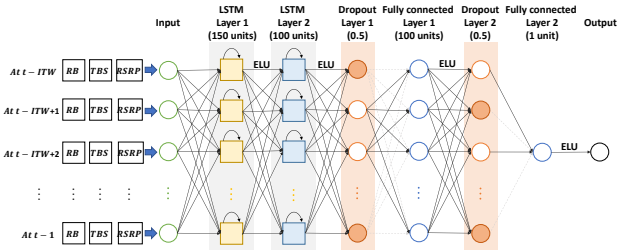


Figure 7: Neural network for uplink throughput prediction in PERCEIVE.

recent input data in the length of ITW so that the potentially stale values can be excluded for better prediction.

Neural network architecture. Each inference model has its neural network. (i) *Cell load dynamics recognition:* the structure of our neural network for this inference is similar to that of our neural network for the uplink throughput prediction (explained below), but it has only a single LSTM layer with 150 hidden units. This simplification is possible since in recognizing cell load dynamics, the model only needs a few number of labels (e.g., three classes in our setting) for effective characterization of cell load variations at the reduced computation cost. (ii) *Uplink throughput prediction:* the neural network of this inference model contains two LSTM layers, where the numbers of hidden units are 150 and 100, respectively. Figure 7 depicts the overall network architecture including two dropout layers with the dropout ratio of 0.5 and two fully connected layers with the hidden units of 100 and 1. We also implement Exponential Linear Units (ELU) [17] as an activation function, where ELU has the same property of ReLU [53] while overcoming the gradient vanishing problem of ReLU at negative values [17].

Training PERCEIVE. We use 80% of the traces for training, and the remaining 20% for test. In the training phase, we use the Adam optimizer with an initial learning rate of 0.001 [36], which is known to be appropriate for non-stationary problems (like our case). For robustness against outliers in the dataset and model optimization by various target quantile values, we adopt the *Pinball-Loss* [58], which is defined as

$$\text{Pinball-Loss}(y, \hat{y}) = \begin{cases} (y - \hat{y})\tau, & \text{if } y \geq \hat{y}, \\ (\hat{y} - y)(1 - \tau), & \text{if } y < \hat{y}, \end{cases} \quad (1)$$

where y is a real value, $\hat{y}$ is a predicted value, and τ is a target quantile. Pinball-loss estimates a conditional target quantile. For example, if $\tau = 0.5$, the loss function optimizes the model by median, which shows more robustness against outliers than MSE [23]. It is worth noting that overestimation of the throughput can affect user experience more seriously due to buffering compared to underestimation that can lead to the degraded video quality. To reflect such asymmetry, we set the target quantile of $\tau = 0.45$. Thus, the loss function is optimized with a value less than the median, which makes the model prefer underestimation to overestimation.

Input time window choice. If the chosen ITW is too small, the model may not fully reflect network conditions. This, in turn, can lead to either overestimation or underestimation of the uplink throughput. On the other hand, if the chosen ITW is too large, it may be disturbed by the stale information. Thus, it is essential to control the ITW size adaptively according to the cell load dynamics. For this, we consider the selection of the proper model based on the

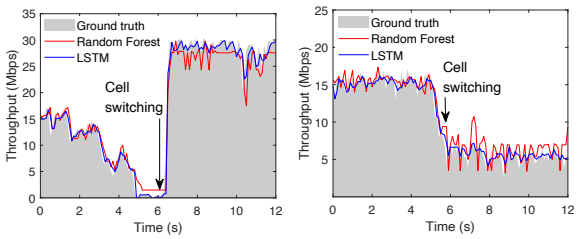
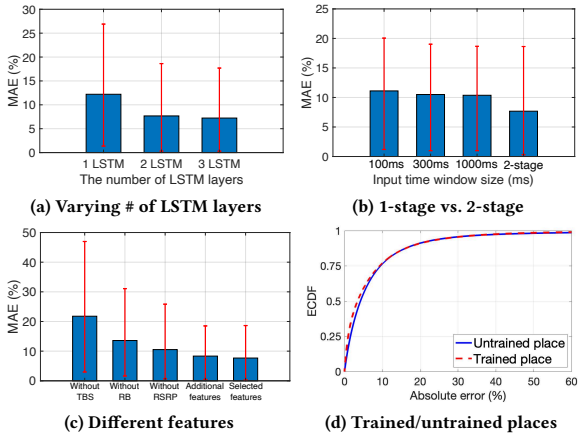


Figure 8: Predicted throughput traces of LSTM and Random Forest in case of cell switching.

cell load dynamics, each of which is trained with a different ITW size. The upper bound of ITW is 1s because we observe that historical data beyond 1s degrades the correlations (§3), while the lower bound is chosen to be 100ms considering prediction accuracy and inference runtime. The middle value is chosen to be 300ms since it is the smallest interval to achieve the best prediction in-between from our dataset. The probability distribution that each of three ITWs achieves the best performance is 0.27, 0.32, and 0.41.

Labeling in cell load dynamics recognition. We need to quantitatively measure the current cell load dynamics, but it is very hard to express the variation of the load by numeric values. Thus, we utilize deep learning to derive qualitative mapping between the patterns of cell load dynamics and the best ITW. For a simple yet effective classification, we categorize the degree of cell load dynamics into three states: high, medium, and low. Then, our problem can be transformed into the mapping of the ITW size among 100ms, 300ms and 1000ms for a given set of traces such that the highest accuracy is achieved empirically, depending on the degree of cell load dynamics. One issue to train this model is the difficulty of labeling the dataset with the degree of cell load dynamics. To produce the training dataset, we compare inference results achieved by each of three ITW values. That is, after conducting uplink throughput prediction using the three ITWs, we label the best window that has the minimum error for the dataset.

Justification of LSTM. In PERCEIVE, we exploit an LSTM-based deep learning approach due to the following reasons: *First*, as discussed in §2, RB traces are the consequence of resource allocation made by BS scheduling algorithm using the input of active user distribution with traffic demand and their service rate over a certain time window. In order to capture such historical trajectory which is hidden to the mobile client, the learning model needs to maintain internal states to pass through the neural network between consecutive time steps. This is exactly what LSTM can do by maintaining memory blocks called *memory cells* [32]. Thus, LSTM can extract implicit characteristic between network load and local RB patterns over time. Note that the correlation analysis in §3.3 confirms overall trend between the cell load and RB pattern and does not explain time-varying relation between them. *Second*, LTE-layer information itself can have a great variety of patterns during 1s window, each of which corresponds to a 1D array sequence with 1000 entries. The huge number of combinations in the input data makes it very difficult for simple learning mechanisms to learn with a proper label during training. It is worth noting that Random Forest [45] does not scale well for time-series prediction with untrained dataset while working well with trained one due to its



(a) Varying # of LSTM layers **(b) 1-stage vs. 2-stage**

(c) Different features **(d) Trained/untrained places**

Figure 9: Prediction performance of PERCEIVE.

intrinsic extrapolation [5, 22], which can be observed in §6.3. *Third*, simple learning algorithms cannot predict throughput variations due to radical cell switching (observed in §3.2). For example, the prediction by decision trees using multiple conditions can exhibit substantial and prolonged inaccuracy. However, LSTM can show immediate adaptation from the resulting decision of high load dynamics in such events. Figure 8 illustrates how LSTM and Random Forest handle the cell switching cases after training them using the two-stage inference model with the same inputs.

4.3 Prediction accuracy

Neural network selection. PERCEIVE uses 2 LSTM layers for the uplink bandwidth predictor. To see the impact of LSTM layers, we compare performance with varying number of LSTM layers. Figure 9a shows the prediction performance attained by different neural networks: 1 LSTM layer with 150 hidden units, 2 LSTM layer with 150 and 100 hidden units each, and 3 LSTM layer with 150, 100, and 50 each. Other layers remain the same as described in Figure 7. As shown, the neural network with 2 LSTM layers achieves the MAE of 7.67%, which provides the best trade-off considering both accuracy and computation complexity.

Inference architecture selection. Our deep learning model is designed to have a 2-stage inference, which controls the ITW size by cell load dynamics. Figure 9b compares the performance between 1-stage inference with constant ITW sizes and 2-stage inference with the most suitable ITW over the dataset collected at trained places. As shown, the 2-stage inference outperforms 1-stage models using a constant ITW: MAE of 2-stage inference is 7.67% while MAEs of 100ms, 300ms and 1000ms are 11.11%, 10.50%, and 10.37%, respectively. Its gain stems from the fact that the ITW size showing the best prediction varies according to cell load dynamics (§4.2).

Feature selection. We choose RB, TBS and RSRP as the input of our deep learning model. Figure 9c shows performance achieved by our model with different combinations of features: our selection (RB allocation, TBS and RSRP) and those features in our selection with the addition of RSRQ and cell ID. Performance with the selected features is slightly better than that with additional features (7.67% vs. 8.33% in MAE) due to inclusion of unnecessary information as observed in §3.2. Moreover, performance without either RSRP,

RB allocation or TBS is degraded compared to that with selected features, implying that the selected features jointly improve the performance.

Untrained places. By testing dataset gathered at untrained places, we verify the generality of our inference model. Figure 9d compares prediction results with data collected at trained places (but collected at different moments than the trained data) and untrained places from data collected in both stationary and mobile scenarios. As shown, the performance of untrained places is not much deviated from that of trained places (8.51% vs. 7.67% in MAE). We conjecture that such good prediction performance attributes to common scheduling strategies (e.g., PF policy) adopted by cellular carriers, as discussed in the literature [10, 34, 69]. We leave the in-depth investigation as our future work.

4.4 Compression for mobile deep learning

Our target platform is a mobile device (e.g., off-the-shelf smartphone), but it is challenging for a mobile device to run the original deep learning algorithm due to heavy computation. Our measurement shows that the latency for a single inference is 224ms on Pixel 3 XL (with Snapdragon 845), which is unacceptable to be used for live streaming. Thus, for the latency reduction, we adopt model compression techniques to our deep learning model. Among various methods [14, 24, 31, 57], we select *pruning* and *quantization* based compression techniques. Pruning reduces the number of weights of a model by eliminating less important weights, while quantization reduces the size of a model and computation by lightening the data type of weights. For pruning, we use the gradual pruning algorithm [73], which gradually increases sparsity of a model until reaching a target sparsity by setting a mask of the smallest weight to zero. For quantization, we convert the data type of weight and activation from `float64` to `float16`. Our compression efforts are evaluated in §6.4.

5 LIVE VIDEO STREAMING SYSTEM

5.1 Design

The design goal is to build a live video streaming system that aims at providing best video quality with managed latency, irrespective of uplink throughput fluctuation. The proposed system consists of three components: online cellular monitoring module, LSTM-based bandwidth predictor, and prediction-based encoding bitrate adaptation algorithm in WebRTC, as depicted in Figure 10.

Challenges. The major challenges to realize our system in mobile phones are three-fold: *First*, WebRTC's adaptation module needs to exploit the predicted bandwidth information in a timely manner considering both incoming rate of raw frames and video encoding time. *Second*, the system should be robust to sporadic mispredictions by smoothly handling the gap between source injection rate (video encoding rate) and actual service rate (LTE uplink throughput). *Third*, the deep learning based inference calculation should be feasible in the mobile phone to instantly provide the prediction of network performance for real-time video streaming.

Our approach. To address the first challenge, we measure video encoding time using several smartphones with different resolutions and take the measured encoding time (T_{enc}) and inter-frame time ($\frac{1}{fps}$) into account to set the prediction window (PW), i.e., $PW > \max(T_{enc}, \frac{1}{fps})$. Then, the predicted throughput can be used for actual transmission of packets after encoding in a synchronized manner. Against the second challenge, we monitor LTE's buffer status and provide a feedback signal to the video encoder at a shorter timescale than the prediction interval to complement the gap due to inaccurate prediction. To overcome the last, we adopt model compression techniques for mobile deep learning so that the minimum processing time could be achieved while preserving high accuracy (§4.4).

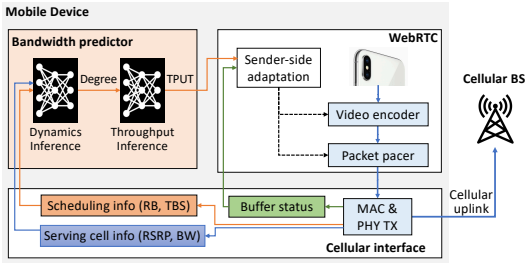


Figure 10: PERCEIVE-based real-time video streaming system architecture.

5.2 Prediction-based rate adaptation

Since cellular uplink is a typical bottleneck for live streaming applications [30, 65], we present a novel adaptation algorithm based on the uplink throughput predictor. In our algorithm, the primary adaptation is performed during encoding process for compressing video frames generated at the video source, as controlling source injection rate is the most effective way to adapt to varying network condition. Meanwhile, the secondary adaptation exploits the buffer status of LTE firmware to handle the discrepancy between predicted encoding rate at upper layer and actual throughput at lower layer.

Video encoding rate control. We set the video encoding rate with the output of the LSTM-based predictor for $\lfloor PW/\frac{1}{fps} \rfloor$ amount of next frames by

$$r_{encoder}(t) = \Delta(t) \times \widehat{BW}(t), \tag{2}$$

where $\widehat{BW}(t)$ is the predicted throughput during the time period of PW from time t. In our design, we set the PW to 100ms. This value corresponds to three consecutive frames in case of 30fps and is large enough to be used for accurate and synchronous prediction compared to the average encoding time ($< 20ms$) observed in our measurements. $\Delta(t)$ is an adaptation parameter to handle the prediction error caused by the nature of cellular load dynamics, which is explained below.

Queueing feedback to handle mismatch. In order to compensate for the sporadic error (i.e., misprediction) by our LSTM-based predictor, we observe queueing delay at LTE uplink firmware buffer, where the queueing delay $d(t)$ at time t is inferred from the actual throughput $BW(t)$ and the firmware buffer occupancy $B(t)$, both of which can be gathered from PHY-layer statistics (i.e., $d(t) = \frac{B(t)}{BW(t)}$).

If the queueing delay increases due to the increased buffer occupancy, it implies that the predictor overestimates the uplink throughput. To quickly respond to the overbuffering case, our adaptation module sends the signal to WebRTC encoder to reduce the

encoding rate. This not only directly adapts video quality within the overestimated interval (100ms), but also prevents the increased latency from a queue buildup. In contrast, if the queueing delay is managed within the threshold, it sends the signal to increase the encoding rate to follow the predicted throughput. In sum, the adaptation parameter Δ is continuously updated by those signals over time as follows:

$$\Delta(t+1) = \begin{cases} \alpha \times \Delta(t), & \text{if } d(t) < d^*, \\ \beta \times \Delta(t), & \text{if } d(t) \geq d^*, \end{cases} \quad (3)$$

where d^* is the queueing delay threshold (20ms in our setting) and $\alpha = 1.05$, $\beta = 0.9$ and $\Delta_{init} = 1$. The bound of $\Delta(t)$ can be chosen by considering upper-layer header overhead and maximum prediction error. Updating Δ is performed every buffer status report.

Moreover, we add a safety mechanism to cope with a long-term throughput overestimation from the deep learning predictor. For the detection, we observe the moving average of queueing delay ($d_{avg}(t)$) at the LTE firmware buffer. If $d_{avg}(t) \geq d^*$, then we drop a frame.

RTP sending rate control. In parallel to the video encoding adaptation, a sending rate is controlled at the transport layer. To avoid unnecessary queueing at RTP layer, we set the pacing rate r_{RTP} to always exceed the video encoding rate $r_{encoder}$, i.e., $r_{RTP}(t) = \gamma \times r_{encoder}(t)$, where $\gamma > 1$. We tested different γ's, but did not observe any difference between the values. In our evaluation, we use $\gamma = 2.5$ (by default in WebRTC).

5.3 Fallback to GCC

Even if the backlog of the cellular uplink buffer stays unchanged but the measured RTT at RTP keeps increasing over time, we consider that a bottleneck happens elsewhere than cellular uplink. To detect such cases, we monitor current RTT and compare it with the sum of the minimum RTT (observed over last 10s to handle route changes that may alter the minimum RTT of the path [6]) and the moving average of LTE queueing delay. If the current RTT exceeds the summation, we assume that the queueing happens on the forward path, implying that the cellular uplink has no bottleneck. At the time, the adaptation module would allow that GCC takes control to reflect the e2e network condition, i.e., $r_{encoder} = r_{GCC}$, governed by GCC. This way, we reuse the legacy WebRTC rate control logic. Later, when the current RTT approaches the sum of the minimum RTT and averaged queueing delay, our adaptation module returns to the PERCEIVE rate control.

6 EVALUATION

6.1 Methodology

For fair comparison under the same network conditions, we use fine-grained network traces collected from three major U.S. operators' LTE networks as the time-varying input in our simulation.

Prediction algorithms. We implement four different prediction algorithms: (i) **PERCEIVE** (ours), (ii) **LinkForecast** [70], (iii) **PROTEUS** [69], and (iv) **Rebera** [39]. For (ii), we train the prediction model based on Random Forest using the past 1s input of TBS, RSRP, and RSRQ. For (iii), we train the prediction model based on regression trees using the past 20s input of TBS, while for (iv), we train the model based on EWMA (exponential weighted moving average) with the past 1s TBS values. All prediction algorithms are faithfully implemented following their original papers.

Measurement metric. In addition to achieved throughput, we observe the following two metrics for measuring user experience.
(i) Video quality: We select Peak Signal to Noise Ratio (PSNR) [63] for video quality assessment. For calculation of PSNR regardless of frame drops, we insert the index barcode into each frame to precisely match the sender-side frame with the receiver-side frame.
(ii) e2e latency: We synchronize the time of sender and receiver using NTP. Then, the sender records per-frame index and time before encoding. Meanwhile, the receiver stores receiving frames as a video file, and generates frame-level index and rendering time. Finally, the time difference between the frames of the same index from two-endpoints becomes e2e latency.

6.2 Implementation

Online LTE monitoring tool. We developed a real-time LTE monitoring tool based on MobileInsight [44], providing uplink scheduling information and serving cell details. To minimize the delay of decoding the log, our tool is built as a MobileInsight plugin [52], and a memory-mapped file technique [62] is applied for supporting rapid inter-process communications.

Uplink throughput predictor. This consists of two phases: (i) training a model offline, (ii) real-time inference on Android. For (i), we build and train our LSTM-based model using Tensorflow-Lite [27] and Keras [15] in a desktop. Then, we compress the trained model to reduce computation at the inference phase by using Tensorflow-Lite converter and Tensorflow Model Optimization Toolkit [28] as mentioned in §4.4. For (ii), we implement our compressed learning model using Java Tensorflow Lite library. We build a Java Android application that can load the compressed model, which infers uplink throughput with LTE-layer data received from the online monitoring tool. The prediction from the 2-stage inference model is made every 100ms.

Rate adaptation for WebRTC. We implement a prediction-based rate adaptation module by modifying source codes of WebRTC. We set the video encoding rate with the predicted value from the uplink throughput predictor. Also, the adaptation module directly receives firmware buffer length and TBS from the online LTE monitor to calculate queueing delay so that it can timely handle mismatch and fallback operations.

Trace-driven simulator. This consists of three modules: (i) a frame handler module extracts frames from a raw video file and uses the ffmpeg encoder [3] to compress them according to a predicted bitrate. (ii) a rate adaptation module runs the inference logic of four different algorithms based on necessary information from the network trace. (iii) a transport module performs packetization for each frame and transmits packets through a simulated network path whose bandwidth changes dynamically following the collected traces (§4.1). Also, it maintains a firmware buffer by comparing the encoded frame size and actual transmission size.

6.3 Comparison with state-of-the-art

6.3.1 Comparison of prediction algorithms. We compare the prediction performance of PERCEIVE with existing prediction algorithms over 8.7-hour network traces. Figure 11 shows the comparison of

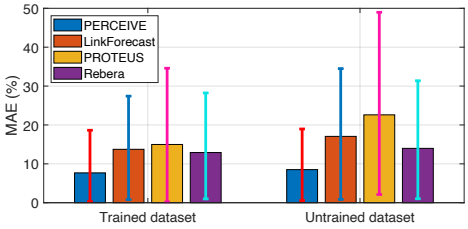


Figure 11: Prediction performance comparison between PERCEIVE and other algorithms.

	PERCEIVE	LinkForecast	PROTEUS	Rebera	FBCC	GCC
Latency (avg)	**138.5ms**	318.2ms	1279.2ms	224.2ms	206.9ms	227.5ms
Latency (p95)	**159.1ms**	668.2ms	4505.2ms	364.3ms	353.9ms	471.4ms
Stall (>500ms)	**0.08%**	8.78%	39.58%	1.32%	3.0%	4.7%

Table 2: e2e latency and time portion of video stall comparison between PERCEIVE and other algorithms: p95 stands for 95-th percentile.

MAE performance between them by separating into trained and untrained datasets. We observe that PERCEIVE outperforms other prediction algorithms consistently and achieves the smallest error (i.e., <20% in 95th-percentile) across the whole datasets, while others do not work well due to much larger errors especially in presence of abrupt throughput fluctuations.

Another interesting observation is that existing algorithms show very poor e2e transport-layer latency[2], all of which are much higher than that of PERCEIVE (Table 2). Their prediction inaccuracy incurs frequent overshooting from actual throughput and then builds up queueing delay at the sender. Such lagged behaviors can lead to severe video stalling events. Table 2 shows the time portion of video stalls that is defined as the event of excessively long latency (>500ms), from which we expect that PERCEIVE can effectively prevent stalling in live video streaming. Note that all prediction mechanisms show comparable throughput (and PSNR) performance despite huge differences in prediction accuracy.

Generality check of algorithms. We test data gathered at untrained places to verify the generality of other prediction algorithms. Figure 11 depicts the prediction results with untrained datasets including mobility scenarios. As shown, in contrast to PERCEIVE's universality (§4.3), other learning-based algorithms show far worse performance compared to the trained case. The main reason is that they cannot exploit the hidden property shared by the underlying scheduling for the unseen network traces.

A microscopic analysis. We find that PERCEIVE's gain comes from its ability to accurately estimate instantaneous uplink throughput. Figure 12 showcases a representative example of a 30-second session. As shown, PERCEIVE precisely follows the ground-truth values (via TBS calculation) in spite of high throughput dynamics. Existing learning-based algorithms do not capture severe bandwidth variations well, even if they can adapt to the gradual change. This results in severely delayed adaptation and frequent overuse of the available bandwidth, causing higher streaming latency.

6.3.2 Comparison of rate adaptation algorithms. We compare the performance of PERCEIVE-based adaptation with two rate control

[2]This excludes application-layer delay such as encoding time at the sender, jitter buffer delay [25] and decoding time at the receiver.

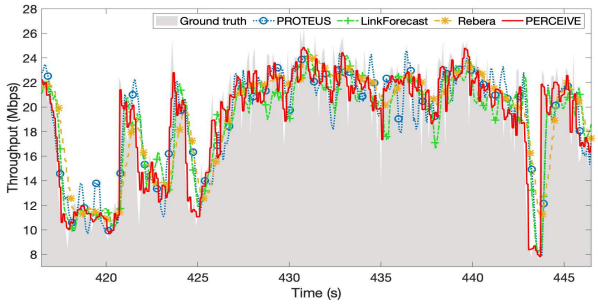


Figure 12: Predicted throughput traces of PERCEIVE and other ML algorithms.

algorithms developed for WebRTC, GCC (default in WebRTC) [33] and an LTE firmware buffer-aware congestion control (FBCC) [65], over the same network traces used in the previous evaluation. Table 2 contains their results. We observe that PERCEIVE outperforms both GCC and FBCC, while the latter two algorithms show moderate performance. The benefit of FBCC stems from its interaction with the LTE chipset, where it cuts off the encoding rate from GCC when it detects the overuse of cellular uplink by observing the increasing trend of LTE firmware buffer occupancy. However, such responsive detection cannot handle the overestimation quickly. Also, it does not have an efficient ramping up mechanism for bandwidth probing over GCC. In contrast, PERCEIVE shows the shortest e2e latency while it effectively prevents video stalling thanks to the high prediction accuracy.

6.4 Micro-benchmarks

We use trace-driven simulations to quantify the impact of different system components of PERCEIVE for deep understanding under identical network conditions.

Impact of queueing feedback. For complementing prediction mismatch, we devised the adaptation mechanism based on queueing feedback from LTE firmware (§5.2). We evaluate its impact by testing PERCEIVE with and without that mechanism. Figure 13 shows the comparison of normalized predicted throughput ($= \frac{\text{predicted tput}}{\text{actual tput}}$) and LTE firmware queueing delay between two cases. As expected, PERCEIVE without adaptation shows a little high throughput performance compared to that with adaptation. However, PERCEIVE with adaptation can achieve much smaller queueing delay of 7.6ms (from 46.1ms) at the small throughput loss of 2.5%. For $\Delta \in [0.8, 0.99]$, we can reduce the number of the overuse from 42.9% to 35.2% with the averaged Δ of 0.97. Thus, we believe that PERCEIVE with the adaptation can provide the best trade-off between throughput and e2e latency in realistic conditions.

Impact of prediction window. Our framework targets short-term prediction to cope with abrupt throughput fluctuations. But, for a low-end device with low capabilities to use our prediction framework, computational requirements may need to be relaxed. One solution is to predict throughput using a larger prediction window (e.g., 1000ms). For this, we re-train PERCEIVE and evaluate its performance: MAE of 7.73% (vs. 7.67% in 100ms prediction). The result confirms that PERCEIVE performs persistently in both prediction windows. Thus, depending on the computational resources

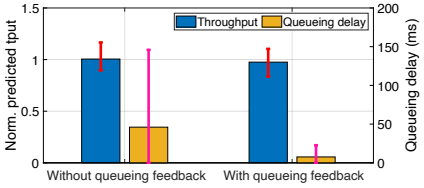


Figure 13: Throughput and queueing delay comparison of PERCEIVE w.r.t. queueing feedback.

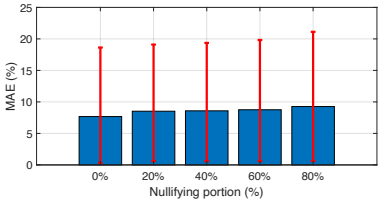


Figure 14: Prediction performance of PERCEIVE with different traffic patterns: $X\%$ is off period in 100ms interval.

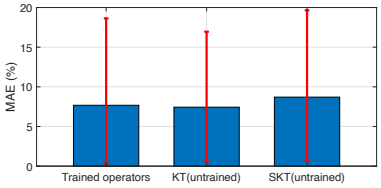


Figure 15: Prediction performance comparison between trained operators in U.S. and untrained ones in Korea.

of a target mobile device, we can choose the model with a proper prediction window.

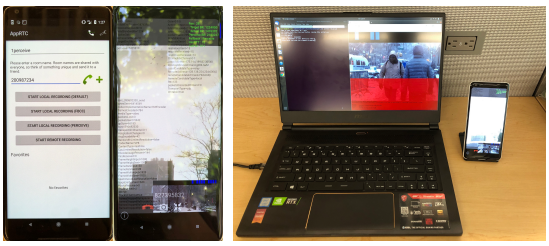
Impact of model compression technique. As described in §4.4, we apply pruning and quantization for mobile deep learning compression. We evaluate the impact of those techniques: the runtime for a single inference calculation is reduced by about 10× (≈22.3ms in Pixel 3 XL) while acheving comparable prediction performance (MAE of 8.73%). This is because our simplified neural network can still maintain the key property from the originally trained model (i.e., time-varying dependency between RB allocation and uplink load dynamics).

Impact of traffic pattern. To verify the performance of PERCEIVE in various traffic patterns, we change the burst portion of input data by nullifying the original data over every 100ms interval. That is, if we use the nullifying portion of 20%, it means 20ms removal from the input trace every 100ms. Figure 14 shows the prediction performance of PERCEIVE with the nullified input trace from 0 to 80%. We can see only a small increase in MAE with the increasing portion of nullification. Since the BS schedules uplink users at 1ms granularity, we can instantly infer the cellular link throughput by observing the packet transmission patterns even in a very short time window (e.g., 10ms), as the previous work observed in the cellular downlink [55]. Thus, we confirm that PERCEIVE is robust against a realistic traffic pattern with burstiness.

Impact of untrained operator. To see the generality of PERCEIVE in different cellular networks, we perform the test with LTE network traces gathered from two (untrained) major operators (KT and SKT) in South Korea. Note that our inference model is trained with the dataset gathered in the U.S. (§4.1). As shown in Figure 15, PERCEIVE reveals high prediction accuracy even in the networks deployed by different operators. The MAE of the untrained operators (KT and SKT) is comparable with that of the trained operators (within 1% point difference on average). We conjecture that such a small gap is attributed to similar resource allocation patterns realized by the widely used PF scheduling policy among different operators, even if they have proprietary configurations in detail.

6.5 Performance in the wild

Experimental setup. As a mobile sender, we run the whole components of our proposed system on Pixel 2/3 XL. For a receiver, we use the high-end wired server (Figure 16) to evaluate the primary scenario where cellular uplink is bottleneck. To obtain fair results between different algorithms, we repeat each experimental scenario five times with the session of two minutes, and instead of using a live video from camera, we modify the WebRTC code



(a) WebRTC app **(b) WebRTC receiver/sender**

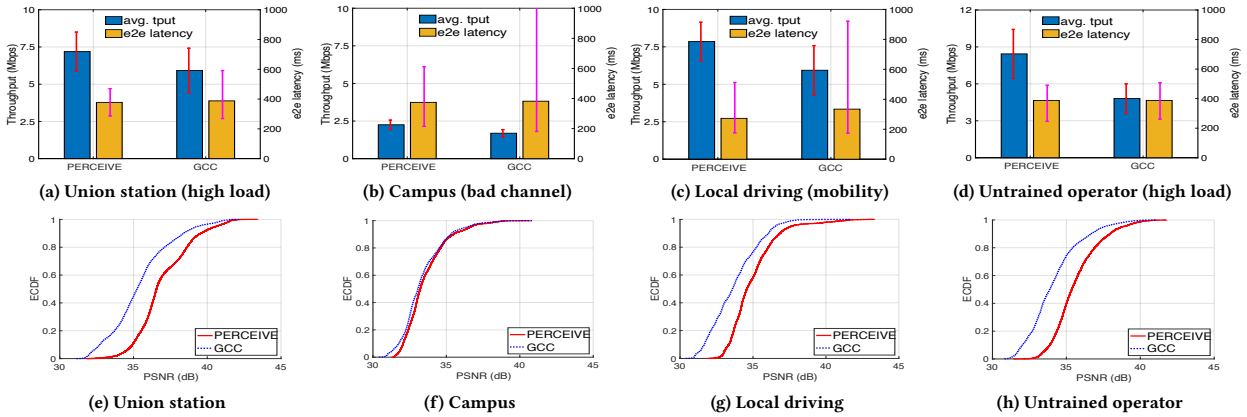
Figure 16: Equipments for our experiment.

to feed a looping raw video file into the video encoder. Unless mentioned otherwise, WebRTC is configured with the resolution of 1920x1080 (FHD) at 30fps and the max video encoding rate of 10Mbps following the guideline in [26].

High cell load scenario. We verify the resiliency of PERCEIVE by conducting the experiments at the union station of a metropolis. Figure 17a shows average throughput and e2e latency comparison between PERCEIVE and GCC. We find that PERCEIVE can achieve higher throughput while keeping e2e latency smaller than GCC. The high utilization of cellular uplink bandwidth can lead to the PSNR gain, as shown in Figure 17e. The reason of improvement is that PERCEIVE can figure out available bandwidth very quickly from the deep learning-based predictor, while GCC conservatively explores throughput based on delay-induced feedback from the receiver (due to negligible loss in-between). Note that the higher cell load dynamics, the bigger performance gap appears.

Weak channel quality. We test the robustness of PERCEIVE by comparing the performance under poor RSRP conditions (around -110dBm). The experiments are conducted during idle time to minimize the impact of background traffic. Figure 17b shows average throughput and e2e latency comparison between PERCEIVE and GCC. We find that PERCEIVE has slightly better throughput performance and much smaller 95th percentile latency than GCC. As the achievable uplink throughput is in small regime, the PSNR gain is small (Figure 17f). However, it is impressive that PERCEIVE can adapt to a poor channel condition even more efficiently than GCC, the most conservative algorithm, does. We also find that with increasing RSRP, the gain becomes large because GCC takes more time to reach achievable throughput due to its slow bandwidth adaptation, while PERCEIVE can reach quickly thanks to the deep learning-based prediction mechanism.

Mobility scenario. User mobility can affect channel conditions in LTE networks. To evaluate this impact, we conduct tests by moving a phone inside a vehicle at 35mph. During our experiments,



(a) Union station (high load) (b) Campus (bad channel) (c) Local driving (mobility) (d) Untrained operator (high load)

(e) Union station (f) Campus (g) Local driving (h) Untrained operator

Figure 17: Performance comparison between PERCEIVE and GCC in the wild: (a)-(d) show average throughput and e2e latency for different scenarios, while (e)-(h) show ECDF of PSNR.

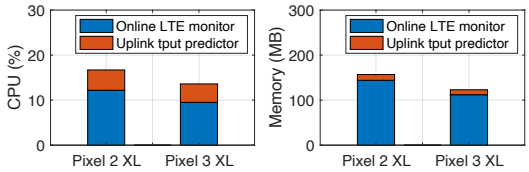


Figure 18: CPU/memory usages of PERCEIVE.

the phone was under seamless LTE network coverage. Figure 17c shows the result for the vehicle mobility scenario. We observe that PERCEIVE performs better than GCC regarding both throughput and latency. Also, we see the substantial gain in video quality as shown in Figure 17g. Interestingly, we can see that the tail latency of PERCEIVE is much shorter than that of GCC. To keep e2e latency low, PERCEIVE handles queueing delay in a timely manner, which is enough to catch up with the changing speed of throughput even by vehicular mobility.

Untrained operator's network. We evaluate the performance of PERCEIVE in a completely different LTE environment deployed by an untrained operator. Here, we test 4K resolution with max encoding rate of 15Mbps. We perform the experiments to see the impact of high load dynamics in the downtown of a metropolitan city. Figure 17d reveals the superiority of PERCEIVE even at the untrained operator's network. Further, PERCEIVE can provide better video quality than GCC regarding PSNR (Figure 17h) and the gap would increase as the variation of cell load increases.

6.6 System overhead

CPU and memory usages. We measured CPU and memory usages on two different phones while running PERCEIVE. Figure 18 confirms that their utilization is not critical (4% in CPU and 11MB in RAM), implying that our mobile deep learning module can execute the prediction in real time and the WebRTC can use it without any significant overhead. Also, considering the time granularity (1ms), online monitoring of LTE-layer information from the phone's diagnostic mode does not severely affect normal operation, as in [44].

Energy consumption. We use the Accubattery [2] for the measurement. We run FHD video stream and record the energy consumption in each experiment, where each video lasts for 5 minutes. The extra cost of PERCEIVE and online LTE monitoring tool are about 41.1mAh and 80.1mAh on average, which is affordable compared with total battery capacity of smartphones (e.g., ≈3500mAh in Pixel 2/3 XL). Note that the energy consumption of the default WebRTC app is 93.3mAh for the same duration.

7 DISCUSSION

Different deep learning algorithms. Deep Reinforcement Learning (DRL) [51] can be another option for prediction. For example, DRL can generate the best encoding policy by learning how much bandwidth change is necessary to mitigate buffering in cellular networks. Compared to the recent work for chunk-based mobile video download [50], applying DRL to bitrate selection of live video streaming is likely to be challenging due to its tight latency budget.

Limitation of using existing codec. Despite its per-frame encoding capability [21], WebRTC's video encoder aims at achieving a specific bitrate on average for multiple frames, which has been observed by our experiments. This means that even if we set the encoding rate based on the prediction, it takes a longer time to reach the target bitrate occasionally, leading to sub-optimal performance in practice. To fully exploit the capability of PERCEIVE, we believe that a tight integration between video codec and rate control should be introduced for per-frame level optimization, e.g., [7, 21].

Limited dataset for learning. We admit that our dataset used for training may not be sufficient to cover all possible scenarios in the field. But, compared to the time scale of the input (i.e., 1s) in our deep learning model, the dataset of 48.5-hour length is likely to contain most dominant cases with respect to cell load, mobility, and signal strength, which is demonstrated by our field experiment. A recent large-scale empirical study shows that mobile traffic patterns can be classified into only five groups, even in metropolis [68].

Overhead reduction. PERCEIVE requires the rooting of a mobile device to obtain LTE-layer information from the cellular modem. Although MobileInsight fully supports many smartphones equipped with Qualcomm chipsets [44], we still have the limitation of cellular chipset availability from other vendors. To avoid the overhead of running such an LTE monitor, recent researches exploit Android API [41, 70] and transport-layer information in OS kernel [55]. Inspired by them, we believe that most of LTE-layer information can be directly collected from the kernel at a coarse granularity or

inferred by considering the timing relation between the cellular modem and kernel, which we plan to explore for future research.

Competition between multiple devices. PERCEIVE's uplink load prediction enables the device to infer the activities of competing devices in the same cell, which does not hinder fair allocation by BS scheduler. Specifically, when the existing PERCEIVE client has fully exploited the uplink resources and other PERCEIVE clients join later, the cellular BS would allocate fewer resources to the existing one based on its built-in fairness criteria (e.g., PF policy). This, in turn, makes the existing PERCEIVE client adapt to the new network condition by promptly detecting the reduced portions of RBs allocated to itself.

Other potential applications. Considering the growth of mobile live video streaming traffic [16, 56, 72], we believe that our approach can improve the performance of applications that account for the major portion of uplink traffic. Nevertheless, we can come up with other applications that exploit PERCEIVE. First, PERCEIVE can be applied to mobile Augmented Reality (AR) systems to support near real-time adaptation in cellular networks with varying channel conditions, e.g., [46], owing to its accurate short-term throughput prediction. Second, it can be used for TCP improvement in cellular uplinks, independently with the research in downlinks, as a rapid throughput probing mechanism is one of the fundamental elements in congestion control. In addition to the latency gain [40], PERCEIVE can maximize the utility of multipath TCP over multiple cellular networks especially for fast mobility scenarios, e.g., [43].

8 RELATED WORK

Cellular performance prediction. Liu et al. [47] find that wireless channel data rate shows severe variability over long time scales, but retains high memory and predictability over small time scales in 3G network. PROTEUS [69] shows the possibility of performance prediction in 3G networks by using upper-layer statistics. LinkForecast [70] identifies a variety of LTE-layer information for downlink throughput prediction. Rebera [39] studies real-time bandwidth prediction in cellular networks by using linear adaptive filtering rather than deep learning. Concerto [72] addresses the incoordination between transport and codec layers using a deep imitation learning model, but shows its gain in low bitrate ranges (<1.2Mbps). In contrast, PERCEIVE presents a novel prediction framework based on millisecond-level LTE-layer information, thus achieving very high accuracy in short-term uplink throughput prediction. More importantly, PERCEIVE can classify the degree of cell load dynamics based on RB allocation traces, which is used to enable better prediction for the next short-term window. The insight of PERCEIVE is that it is possible to predict cellular performance by observing how the BS allocates channel resources to its users. We believe that this insight would likely continue to remain valid in the future. Moreover, we expect that our prediction framework would become more valuable in mmWave-based 5G networks due to frequent cell switch and severe throughput fluctuation in the wild [54].

Cellular channel quality prediction. Apart from the throughput prediction, cellular channel quality prediction has been explored in the literature [37, 49, 74]. Zhu et al. [74] introduced an LSTM-based approach for real-time channel state information prediction of non-stationary channels. Kulkarni et al. [37] presented DeepChannel, an encoder-decoder based sequence-to-sequence deep learning model that is capable of predicting future signal strength variations based on past data, which is applicable to various wireless networks. Also, the channel state prediction in 5G networks has been studied by proposing a combined deep learning framework of a convolutional neural network (CNN) and an LSTM network [49].

Long-term cellular traffic forecast. Long-term cellular traffic load prediction is another important domain for traffic engineering. Wang et al. [60] firstly proposed a deep learning model for spatiotemporal modeling and prediction in cellular networks using a large LTE dataset. Zhang et al. [71] exploited a convolutional LSTM network to perform long-term mobile traffic volume predictions. Recently, Bega et al. [9] presented DeepCog, a novel data analytics tool able to forecast the capacity based on 3D-CNNs to facilitate network slicing in 5G networks.

Cellular downlink load estimation. LoadSense [13] experimentally showed that throughout depends not only on link quality but also on cellular load, and RSRQ can be used for estimating the load in the downlink. piStream [66] proposed to monitor PHY-layer information (i.e., per-subcarrier energy level) to estimate available bandwidth over LTE for video streaming adaptation. CLAW [67] has studied passive load inference in cellular downlink using RSRQ. After that, CASTLE [41] has improved the accuracy by considering inter-cell interference.

Cellular measurement tool. LTEye [38] is an open platform that can run on an off-the-shelf SDR and analyzes LTE radio performance by monitoring PHY-layer. OWL [12] provides a reliable measurement framework for LTE control channel decoding. MobileInsight [44] can analyze cellular networks using smartphones by exposing protocol messages from the cellular chipset. MMLab [19] is designed for parsing detailed handover related messages beyond MobileInsight. BurstTracker [8] detects if LTE downlink is bottleneck based on bursty resource allocation patterns.

9 CONCLUSION

In this paper, we proposed PERCEIVE, an LSTM-based prediction framework of uplink throughput in cellular networks. Using our extensive dataset over multiple operators' LTE networks, we confirmed that PERCEIVE can predict uplink throughput with high accuracy. As a practical application, we implemented and integrated PERCEIVE-based adaptation into WebRTC that can run on off-the-shelf smartphones. Our evaluation results show that predictive rate adaptation through PERCEIVE can significantly improve user experience over existing prediction techniques.

ACKNOWLEDGMENTS

We sincerely thank our shepherd Kyle Jamieson and anonymous reviewers for their valuable comments. This research was supported in part by the NIST under Grant No. 70NANB17H186, Cisco Systems under Grant 1368170, and IITP grants funded by the Korea government (MSIT) (No. 2017-0-00562, UDP-based Ultra Low-Latency Transport Protocol with Mobility Support and No. 2017-0-00692, Transport-aware Streaming Technique Enabling Ultra Low-Latency AR/VR Services). K. Lee and S. Ha are co-corresponding authors.

REFERENCES

[1] iPerf - The ultimate speed test tool for TCP, UDP and SCTP, June 2016. https://iperf.fr/iperf-download.php.

[2] AccuBattery, 2019. https://accubatteryapp.com.

[3] FFmpeg, October 2019. https://www.ffmpeg.org/download.html.

[4] 3GPP. LTE: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures (Release 15). http://www.3gpp.org/dynareport/36213.htm, 2019.

[5] Arora, A. Why Random Forests can't predict trends and how to overcome this problem?, December 2018. http://shorturl.at/hmyD8.

[6] Arun, V., and Balakrishnan, H. Copa: Practical Delay-Based Congestion Control for the Internet. In *Proceedings of USENIX NSDI* (2018).

[7] Baig, G., He, J., Qureshi, M. A., Qiu, L., Chen, G., Chen, P., and Hu, Y. Jigsaw: Robust Live 4K Video Streaming. In *Proceedings of ACM MobiCom* (2019).

[8] Balasingam, A., Bansal, M., Misra, R., Nagaraj, K., Tandra, R., Katti, S., and Schulman, A. Detecting if LTE is the Bottleneck with BurstTracker. In *Proceedings of ACM MobiCom* (2019).

[9] Bega, D., Gramaglia, M., Fiore, M., Banchs, A., and Costa-Perez, X. DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning. In *Proceedings of IEEE INFOCOM* (2019).

[10] Bu, T., Li, L., and Ramjee, R. Generalized Proportional Fair Scheduling in Third Generation Wireless Data Networks. In *Proceedings of IEEE INFOCOM 2006* (April 2006).

[11] Bui, N. IMDEA-OWL, August 2017. https://git.networks.imdea.org/nicola_bui/imdeaowl.

[12] Bui, N., and Widmer, J. OWL: A Reliable Online Watcher for LTE Control Channel Measurements. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges* (2016).

[13] Chakraborty, A., Navda, V., Padmanabhan, V. N., and Ramjee, R. Coordinating Cellular Background Transfers Using Loadsense. In *Proceedings of ACM MobiCom* (2013).

[14] Chen, T., Goodfellow, I., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641* (2015).

[15] Chollet, F., et al. Keras. https://github.com/fchollet/keras, 2019.

[16] Cisco. Cisco Visual Networking Index, 2016-2021, June 2017. https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1853168.

[17] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[18] Deng, H., Ling, K., Guo, J., and Peng, C. Unveiling the Missed 4.5G Performance In the Wild. In *Proceedings of ACM HotMobile* (2020).

[19] Deng, H., Peng, C., Fida, A., Meng, J., and Hu, Y. C. Mobility Support in Cellular Networks: A Measurement Study on Its Configurations and Implications. In *Proceedings of ACM IMC* (2018).

[20] Ettus Research. USRP B210. http://www.ettus.com/all-products/ub210-kit/, 2019.

[21] Fouladi, S., Emmons, J., Orbay, E., Wu, C., Wahby, R. S., and Winstein, K. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *Proceedings of USENIX NSDI* (2018).

[22] free range statistics. Extrapolation is tough for trees!, December 2016. http://freerangestats.info/blog/2016/12/10/extrapolation.

[23] Ghosh, A., Kumar, H., and Sastry, P. Robust loss functions under label noise for deep neural networks. In *Proceedings of AAAI Conference on Artificial Intelligence* (2017).

[24] Gong, Y., Liu, L., Yang, M., and Bourdev, L. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[25] Google. WebRTC playout-delay, October 2018. https://webrtc.org/experiments/rtp-hdrext/playout-delay/.

[26] Google. Recommended upload encoding settings. https://support.google.com/youtube/answer/1722171?hl=en, 2019.

[27] Google. Tensorflow Lite. https://www.tensorflow.org/lite/, 2019.

[28] Google. Tensorflow Model Optimization Toolkit. https://www.tensorflow.org/model_optimization, 2019.

[29] Google. WebRTC Native Code. https://webrtc.org/native-code/, 2019.

[30] Guo, Y., Qian, F., Chen, Q. A., Mao, Z. M., and Sen, S. Understanding On-device Bufferbloat for Cellular Upload. In *Proceedings of ACM IMC* (2016).

[31] Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[32] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1735–1780.

[33] Holmer, S., Lundin, H., Carlucci, G., Cicco, L. D., and Mascolo, S. A Google Congestion Control Algorithm for Real-Time Communication. IETF draft, January 2017. https://tools.ietf.org/pdf/draft-ietf-rmcat-gcc-02.pdf.

[34] Huang, Y., Li, S., Hou, Y. T., and Lou, W. GPF: A GPU-based Design to Achieve ~100 μs Scheduling for 5G NR. In *Proceedings of ACM MobiCom* (2018).

[35] Jansen, B., Goodwin, T., Gupta, V., Kuipers, F., and Zussman, G. Performance evaluation of webrtc-based video conferencing. *SIGMETRICS Perform. Eval. Rev. 45*, 3 (Mar. 2017), 56–68.

[36] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[37] Kulkarni, A., Seetharam, A., Ramesh, A., and Herath, J. D. DeepChannel: Wireless Channel Quality Prediction Using Deep Learning. *IEEE Transactions on Vehicular Technology 69*, 1 (2020), 443–456.

[38] Kumar, S., Hamed, E., Katabi, D., and Erran Li, L. LTE Radio Analytics Made Easy and Accessible. In *Proceedings of ACM SIGCOMM* (2014).

[39] Kurdoglu, E., Liu, Y., Wang, Y., Shi, Y., Gu, C., and Lyu, J. Real-time Bandwidth Prediction and Rate Adaptation for Video Calls over Cellular Networks. In *Proceedings of ACM MMSys* (2016).

[40] Lee, H., Flinn, J., and Tonshal, B. RAVEN: Improving Interactive Latency for the Connected Car. In *Proceedings of ACM MobiCom* (2018).

[41] Lee, J., Lee, J., Im, Y., Sathyanarayana, S. D., Rahimzadeh, P., Zhang, X., Hollingsworth, M., Joe-Wong, C., Grunwald, D., and Ha, S. CASTLE over the Air: Distributed Scheduling for Cellular Data Transmissions. In *Proceedings of ACM MobiSys* (2019).

[42] Lee, S., Choudhury, S., Khoshnevis, A., Xu, S., and Lu, S. Downlink MIMO with Frequency-Domain Packet Scheduling for 3GPP LTE. In *Proceedings of IEEE INFOCOM 2009* (April 2009).

[43] Li, L., Xu, K., Li, T., Zheng, K., Peng, C., Wang, D., Wang, X., Shen, M., and Mijumbi, R. A Measurement Study on Multi-path TCP with Multiple Cellular Carriers on High Speed Rails. In *Proceedings of ACM SIGCOMM* (2018).

[44] Li, Y., Peng, C., Yuan, Z., Li, J., Deng, H., and Wang, T. Mobileinsight: Extracting and analyzing cellular network information on smartphones. In *Proceedings of ACM MobiCom* (2016).

[45] Liaw, A., Wiener, M., et al. Classification and regression by randomforest. *R news 2*, 3 (2002), 18–22.

[46] Liu, L., Li, H., and Gruteser, M. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In *Proceedings of ACM MobiCom* (2019).

[47] Liu, X., Sridharan, A., Machiraju, S., Seshadri, M., and Zang, H. Experiences in a 3g network: Interplay between the wireless channel and applications. In *Proceedings of ACM MobiCom* (2008).

[48] LTE Quick Reference. RSRP, RSRQ, RSSI, SINR Interplay. https://www.sharetechnote.com/html/Handbook_LTE_RSRP_RSRQ_SINR_Interplay.html, 2020.

[49] Luo, C., Ji, J., Wang, Q., Chen, X., and Li, P. Channel State Information Prediction for 5G Wireless Communications: A Deep Learning Approach. *IEEE Transactions on Network Science and Engineering 7*, 1 (2020), 227–236.

[50] Mao, H., Netravali, R., and Alizadeh, M. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of ACM SIGCOMM* (2017).

[51] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[52] MobileInsight. Running Your Customized Plugin on the Phone, October 2018. http://www.mobileinsight.net/tutorial-plugin.html.

[53] Nair, V., and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of ICML* (2010).

[54] Narayanan, A., Ramadan, E., Carpenter, J., Liu, Q., Liu, Y., Qian, F., and Zhang, Z.-L. A First Look at Commercial 5G Performance on Smartphones. In *Proceedings of The Web Conference* (2020).

[55] Park, S., Lee, J., Kim, J., Lee, J., Ha, S., and Lee, K. ExLL: An Extremely Low-latency Congestion Control for Mobile Cellular Networks. In *Proceedings of ACM CoNEXT* (2018).

[56] Ray, D., Kosaian, J., Rashmi, K. V., and Seshan, S. Vantage: Optimizing video upload for time-shifted viewing of social live streams. In *Proceedings of ACM SIGCOMM* (2019).

[57] Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Proceedings of IEEE international conference on acoustics, speech and signal processing* (2013).

[58] Steinwart, I., Christmann, A., et al. Estimating conditional quantiles with the help of the pinball loss. *Bernoulli 17*, 1 (2011), 211–225.

[59] tokbox. Video Chatterbox Nation, 2018. https://tokbox.com/resources/video-chatterbox-2018.

[60] Wang, J., Tang, J., Xu, Z., Wang, Y., Xue, G., Zhang, X., and Yang, D. Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach. In *Proceedings of IEEE INFOCOM* (2017).

[61] Wikipedia. Coefficient of variation. https://en.wikipedia.org/wiki/Coefficient_of_variation, 2019.

[62] Wikipedia. Memory-mapped file. https://en.wikipedia.org/wiki/Memory-mapped_file, 2019.

[63] Wikipedia. Peak signal-to-noise ratio. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio, 2019.

[64] Winstein, K., Sivaraman, A., and Balakrishnan, H. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proc. of USENIX NSDI* (2013).

[65] Xie, X., and Zhang, X. POI360: Panoramic Mobile Video Telephony over LTE

Cellular Networks. In *Proceedings of ACM CoNEXT* (2017).
[66] Xie, X., Zhang, X., Kumar, S., and Li, L. E. piStream: Physical Layer Informed Adaptive Video Streaming over LTE. In *Proceedings of ACM MobiCom* (2015).
[67] Xie, X., Zhang, X., and Zhu, S. Accelerating Mobile Web Loading Using Cellular Link Information. In *Proceedings of ACM MobiSys* (2017).
[68] Xu, F., Li, Y., Wang, H., Zhang, P., and Jin, D. Understanding mobile traffic patterns of large scale cellular towers in urban environment. *IEEE/ACM Transactions on Networking 25*, 2 (April 2017), 1147–1161.
[69] Xu, Q., Mehrotra, S., Mao, Z., and Li, J. PROTEUS: Network Performance Forecast for Real-time, Interactive Mobile Applications. In *Proceedings of ACM MobiSys* (2013).
[70] Yue, C., Jin, R., Suh, K., Qin, Y., Wang, B., and Wei, W. LinkForecast: Cellular Link Bandwidth Prediction in LTE Networks. *IEEE Transactions on Mobile Computing 17*, 7 (July 2018), 1582–1594.
[71] Zhang, C., and Patras, P. Long-Term Mobile Traffic Forecasting Using Deep Spatio-Temporal Neural Networks. In *Proceedings of ACM MobiHoc* (2018).
[72] Zhou, A., Zhang, H., Su, G., Wu, L., Ma, R., Meng, Z., Zhang, X., Xie, X., Ma, H., and Chen, X. Learning to coordinate video codec with transport protocol for mobile video telephony. In *Proceedings of ACM MobiCom* (2019).
[73] Zhu, M., and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).
[74] Zhu, Y., Dong, X., and Lu, T. An Adaptive and Parameter-Free Recurrent Neural Structure for Wireless Channel Prediction. *IEEE Transactions on Communications 67*, 11 (2019), 8086–8096.