# Leveraging QoE Heterogeneity for Large-Scale Livecast Scheduling

Rui-Xiao Zhang*, Ming Ma‡, Tianchi Huang*, Hanyu Li*, Jiangchuan Liu**, Lifeng Sun§
* BNRist, Department of Computer Science and Technology, Tsinghua University
§ Key Laboratory of Pervasive Computing, Ministry of Education
‡ Beijing Kuaishou Technology Co., Ltd., China
** Simon Fraser University
*zhangrx17@mails.tsinghua.edu.cn,§ sunlf@tsinghua.edu.cn

## ABSTRACT

Livecast streaming has received great success in recent years. Although many prior efforts have suggested that dynamic viewer scheduling according to the quality of service (QoS) can improve user engagement, they may suffer inefficiency due to their ignorance of viewer heterogeneity in how the QoS impact quality of experience (QoE).

In this paper, we conduct measurement studies over large-scale data provided by a top livecast platform in China. We observe that QoE is influenced by a lot of QoS and non-QoS factors, and most importantly, the QoE sensitivity to QoS metrics can vary significantly among viewers. Inspired by the above insights, we propose HeteroCast, a novel livecast scheduling framework for intelligent viewer scheduling based on viewer heterogeneity. In detail, HeteroCast addresses this concern by solving two sub-problems. For the first sub-problem (i.e., the QoE modeling problem), we use the deep factorization machine (DeepFM) based method to precisely map complicated factors (QoS and non-QoS factors) to QoE and build the QoE model. For the second sub-problem (i.e., the QoE-aware scheduling problem), we use a graph-matching method to generate the best viewer allocation policy for each CDN provider. Specifically, by using some pruning techniques, HeteroCast only introduces slight overhead and can well adapt to the large-scale livecast scenario. Through extensive evaluation on real-world traces, HeteroCast is demonstrated to increase the average QoE by 8.87%-10.09%.

## CCS CONCEPTS

• **Information systems** → *Multimedia streaming*; • **Computing methodologies** → *Neural networks.*

## KEYWORDS

content delivery networks, livecast, QoE-aware scheduling

**ACM Reference Format:**
Rui-Xiao Zhang*, Ming Ma‡, Tianchi Huang*, Hanyu Li*, Jiangchuan Liu**, Lifeng Sun§. 2020. Leveraging QoE Heterogeneity for Large-Scale Livecast Scheduling. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20), October 12–16, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394171.3413918


*MM '20, October 12–16, 2020, Seattle, WA, USA*

ACM ISBN 978-1-4503-7988-5/20/10…$15.00
https://doi.org/10.1145/3394171.3413918

## 1 INTRODUCTION

Livecast services have become increasingly popular and received great development in the past few years, and many livecast platforms have shown unprecedented growth across the world (e.g., Twitch[1] and Kuaishou[2]). As reported by Cisco, there have been more than 36% of Internet users watching live videos in 2016, and the livecast market will grow up to over 70 billion by 2021.

To enhance the high quality of experience (QoE) for the livecast viewers, multi-CDN (content delivery networks) have become the most common structures when delivering streaming content [1, 30, 31]. However, due to the large-scale dynamic viewers and the temporal variation in CDN performance, how to effectively scheduling viewers among CDN providers is still challenging. Compared with other kinds of videos (e.g., video on demand), livecast is equipped with some unique properties. First, the livecast platforms will face an explosive number of viewers due to the numerous broadcasters. For example, it is reported that in 2018, Twitch had more than 2.2 million broadcasters and 15 million daily unique users (DAU) [11], while the DAU of Kuaishou was more than 100 million in 2019 [12]. Such large-scale viewers may bring significant burdens to the CDN performances( §3). Second, given the various service requirements, different viewers may have their individual preference [21, 26]. For example, the viewers who enjoy interacting with broadcasters are much more sensitive to stalling event [3], while for those who only want to take a glance and frequently switch among different channels, startup latency is much more important.

Unfortunately, existing work cannot handle the above features well. Most of them have no considerations of viewer heterogeneity[15, 16, 19, 30, 31], which separates them from efficiently scheduling livecast viewers. Meanwhile, some work indeed pays attention to the viewer heterogeneity [21, 26, 33], but due to their ignorance of the computation overhead, it is hard for them to be practically used for the livecast scenario with large-scale viewers.

To tackle the above concerns, we propose HeteroCast, a novel framework that explores the opportunity of scheduling viewers based on QoE heterogeneity. The critical insight behind our approach is that: simply optimizing the QoS equally across all the

[1] twitch.com
[2] kuaishou.com
[3] A stalling event happens when there is no data in the video player.

viewers can be sub-optimal, as the viewers have heterogeneous sensitivities. Specifically, from the measurement studies on real-world traces, we verify that the relationship between QoE and QoS metrics is complicated and remains a non-linear curve (§3). This indicates that Although this insight is promising, however, directly applying it into practice is still challenging.

▹ The first challenge is how to build up a model that can well characterize user engagement. In this paper, we use watching duration as the measure of engagement. This metric has been widely used in both academia and industry [2, 7]. Actually, user engagement is affected by multiple factors: e.g., different QoS metrics (startup latency, stalling duration, etc.), channel categories (entertainments, outdoors, etc.), and even device types (iPhone, Android, etc.). At the same time, these factors are not independent with each other, and empirical assumptions are unlikely to work.

▹ The second challenge is how to deal with the correlation between the scheduling decision and CDN performance. Different from some static application scenarios (e.g., Spark job services [20]), in which the service performance is the intrinsic quality, and will not be influenced by the decision, in our problem, the QoS of CDN providers will suffer degradation if too many viewers are scheduled to it (i.e., overloaded).

▹ The third challenge is how to run the HeteroCast over large-scale viewers in a real-time way. Even if we have overcome the above two challenges well, it is still impossible to decide for each individual viewer, since the massive viewer number can result in unacceptable computation overhead.

HeteroCast addresses the above challenges by decoupling the original problem into two sub-problems (i.e., the QoE modeling problem and the QoE-aware scheduling problem.). For the first sub-problem, we try to build up the QoE model by leveraging the deep factorization machine (DeepFM) method, which has been widely used in recommendation systems. DeepFM is quite suitable since it can naturally characterize the interactions of multiple factors. However, directly using it will cause too much computation overhead. As a result, we propose to use some pruning methods, which is driven by an insight that viewers sharing some context factors are more likely to have similar QoE patterns. For the second sub-problem, we try to schedule viewers based on the graph-mapping technique. In detail, considering the workload influence on CDN performance, we first allocate viewer requests regardless of their heterogeneity. After that, we then leverage the graph-matching method and QoE model to make a more fine-grained assignment for viewers (i.e., considering the heterogeneity.)

In summary, our contributions are as follows. 1) We carry out a large-scale measurement on real-world data, and we get some insights into a better QoE model and a more effective livecast scheduling method. 2) Based on the the QoE model, we propose HeteroCast, a novel QoE-aware scheduling approach that makes full use of the viewer heterogeneity to schedule viewers. 3) After the extensive trace-driven experiment, HeteroCast is demonstrated to significantly outperform baseline algorithms.

## 2 RELATED WORK

In the past few years, Livecast has received great attention in both academia and industry. There is a large literature focusing on how to improve the viewer QoE. Some of them pay attention to the systematic optimization by introducing additional technique supports (e.g., the cloud services or edge devices). For instance, [5, 9, 25, 27, 34] propose to use cloud-assisted architectures to optimize the content delivery or live video transcoding, while authors in [21] suggest to use edge devices as relays to optimize the first-mile video transmission. Since the above methods need extra equipment supports (e.g., the cloud and edge devices), they cannot be directly applied in contemporary CDN structures.

Multi-CDN has been used as a fundamental delivery architecture in contemporary Internet video services. Especially, due to the dynamics of the CDN provider performance, plenty of work has focused on the multi-CDN selection problem and propose to schedule viewers adaptively. For example, [3, 4, 15] use some feature engineering methods to predict the QoS of different CDN providers, and schedule as many as viewers to the best CDN (i.e., with the highest QoS), while authors in [16] use an exploration-exploitation (E2) based method to estimate the CDN performance, and make decisions in a real-time way. There are also some work use end-to-end learning based methods to schedule viewer [30, 32] However, since these methods all use QoS as the optimization objective, they do not take the heterogeneity of viewer requests into account. In contrast, the viewer scheduling in HeteroCast addresses this concern by building up a precise QoE model. Meanwhile, some work [8, 21, 26] shares with us the idea of providing services based on the QoE of users. Nevertheless, these methods either result in too much computation overhead as they need to be run over each individual viewer, or generate their policies based still on some simple QoE models (e.g., the weighted sum of different QoS metrics). In contrast, by utilizing some pruning techniques and a more precise QoE model, HeteroCast only introduces slight overhead, and can well optimize the QoE for large-scale viewers. There is also some work focusing on modeling QoE. E.g., [13, 28] propose FM-liked QoE models; [29] uses a DNN-based models to characterize viewer engag ement. Our work differs with them as we further integrate the QoE method into the scheduling problem.

## 3 MOTIVATION

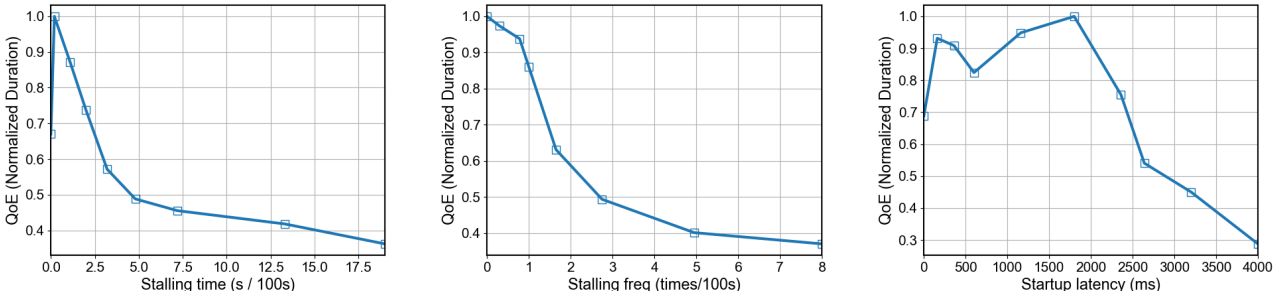
### 3.1 Dataset

The dataset in this paper is collected with the help of Kuaishou, a top livecast platform in China. The dataset consists of two parts: the first part is the CDN data, which contains the performance of different CDN providers, including the following industry-standard QoS metrics [2, 15, 31]:

▹ *Startup latency (StartUp)*: it represents the duration between a viewer requests the video and the video starts to play.

▹ *Stalling frequency (StallFreq)*: it represents the frequency of the stalling events that happened during the session, which can be calculated as the ratio between the overall stalling times and the session duration ($\frac{\sharp stallingevent}{sessiontime}$). For better illustration, we multiply $StallFreq$ with the constant 100, which then represents the stalling frequency per 100 seconds.

▹ *Stalling time (StallTime)*: Unlike $StallFreq$, $StallTime$ focuses on the time spent on the stalling event, and it can be calculated as the rate between the time of stalling and session duration. We also use stalling time per 100 seconds.




(a) The relationship between stalling time and watching duration (b) The relationship between stalling frequency and watching duration (c) The relationship between stalling time and watching duration

**Figure 1: The relationship between QoS factors and the QoE (i.e., normalized watching duration).**

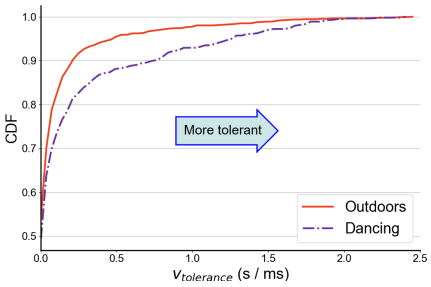



**Figure 2: The influence of channel category.**

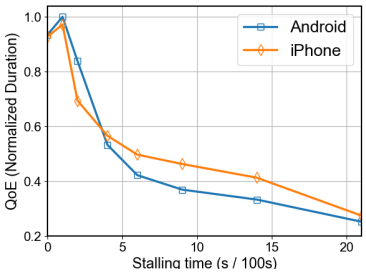



**Figure 3: The influence of device type.**

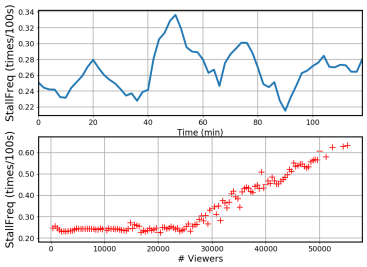



**Figure 4: The dynamics of CDN providers.**

The second part is the user data, which contains about 1000M user request traces spanning one week in 2019. The key features include: access time, leaving time, user ID, channel ID, the device type, the CDN provider, and channel category (e.g., Dancing, Gaming, Outdoors, etc.). All user IDs and channel IDs are anonymized.

## 3.2 QoE Heterogeneity

We first investigate how different QoS factors influence livecast user engagement. Like previous work [2, 7], we use the watching duration as the estimate of QoE[4]. Given the data pair $(x, y)$ in which $x$ is the value of the QoS metric, and $y$ is the QoE value, we group them into the bins. For each bin, we calculate the average QoE value of the sessions that fall in the bin. Figure 1 shows how the three QoS metrics interact with QoE, from which we can derive the following two observations. First, the QoS factors can significantly impact user engagement. For example, in Figure 1(a), with the stalling time increasing from 0.25 to 2.75 second per 100s, the QoE drops sharply from 1.0 to 0.58. This actually indicates that providing better QoS is crucial to improve user QoE. Second, user engagement with different service levels may have different sensitivity to QoS. For example, in Figure 1(b), when the stalling frequency is short (lower than 1 times per 100s), the watching duration is almost unchanged with QoS. At the same time, for the QoE with the stalling frequency between 1 to 1.5, it has the sharpest slope, which means that the QoE is much more sensitive to stalling frequency in this interval. The similar observation can also be derived in Figure 1(c) (i.e., the QoE is insensitive when $StartUp$ is in the interval $(0, 1900)ms$, while much more sensitive in $(1900, 4000)$).

The above observations actually demonstrate that although providing better QoS can generally improve the QoE, the improvement still varies due to the heterogeneity of QoE sensitivity. Meanwhile, these observations also inspire us to schedule viewers in a QoE-aware manner, which may facilitate a better livecast scheduling approach.

## 3.3 Challenges

Despite the fact that integrating heterogeneity of QoE sensitivity into livecast scheduling is promising, directly applying it into livecast scheduling is challenging in both algorithmic and systematic perspectives.

▹***The algorithmic perspective:*** to facilitate a QoE-aware scheduling approach, we first need to build up a QoE model to precisely characterize the relationship between QoS and QoE, which, however, is complicated. As shown in Figure 1, we can see that user engagement decreases non-linearly with QoS factors. This suggests that the QoE model should be able to capture this nonlinear relationship. Second, the QoE is also influenced by multiple non-QoS factors, such as the channel category, and also the device type. These non-QoS factors will interact with the QoS factors, which makes the QoE model even more complicated. As a practical example, we present the relationship between startup latency and QoE of two channel categories in Figure 2. Specifically, for each session, we calculate the ratio between the watching duration and the startup latency (i.e., $v_{tolerance} = \frac{Duration}{StartUp}$) to denote the viewer tolerance. Obviously, a larger $v_{tolerance}$ indicates that these viewers are more

[4]As required by the data provider, we normalize the duration.

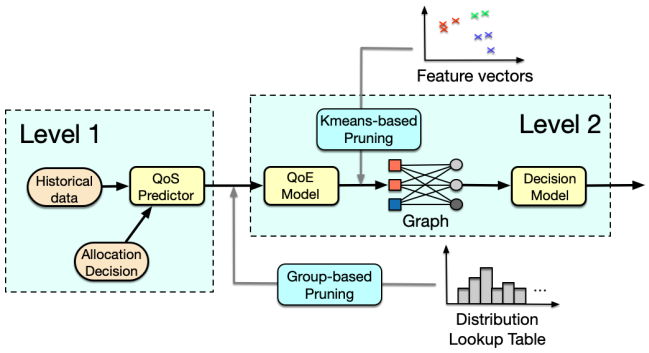



**Figure 5: The system overview of HeteroCast.**

tolerant, as they have longer watching duration for a unit startup latency. We present the $v_{tolerance}$ of two channel categories in the form of CDF, and show the results in Figure 2. We can find that the viewers of "Dancing" channels are more tolerant than the viewers of "Outdoors" channels. This can be explained as the "Outdoors" viewers are likely to interact with broadcasters and other audiences, which makes them more intolerant to delays. Additionally, we also give an example of how the device type influences user engagement. We select the traces from the same channel, and categorize them based on the device type. We present the results in Figure 3, and we can also find that the tolerance of iPhone and Android users also shows some heterogeneity.

▹***The systematic perspective:*** in addition to solving the difficulties encountered from the algorithmic perspective, a practical scheduling approach also needs to meet the following two systematic requirements. First, the scheduling approach should be light-weighted and run in a real-time way. As denoted by previous work [21], livecast viewers are much more sensitive to latency (since they are more likely to interact with others). Therefore, too much overhead will inevitably increase the overall decision time and violate user engagement. In fact, this real-time requirement makes it difficult to apply many algorithms that aggregate requests first and then process them collectively [33]. Second, the scheduling approach should also consider the dynamics of CDN providers. In detail, the dynamics consist of two parts: the temporal dynamics and the workload-related dynamics. For the temporal dynamics, we take startup latency and stalling time as examples and present them in Figure 4. We can find that the performance of the CDN provider varies significantly with time. This can be explained that CDN providers have their own offloading policies (e.g., how many servers should be used) which are blind to livecast platforms, and the policies may change over time [31]. For the workload-related dynamics, we also group the data into the equal-sized bins by the workload, and calculate the average QoS for each bin. The results are shown in Figure 1. As illustrated, the CDN provider will suffer significant QoS degradation when the viewer number exceeds a certain threshold. In other words, compared with a lot of previous work, in which the service performance is assumed as the intrinsic property and will not be influenced by the decision (e.g.,[19]), the CDN performance in our problem is correlated with the scheduling decision. This is reasonable as too many viewers will overload the CDN and thus violate its performance.

**Algorithm 1** The decision logic of HeteroCast at time $t$

**Require:** The QoS model: $QoS$; The QoE model: $QoE$; The historical data (e.g., workload, QoS, and etc.): $HisInfo$; The allocation candidates for all CDN providers: $AlloCandi$

**procedure** HeteroCast
  Initialize the best scheduling policy $\pi$ and its QoE $Q$.
  /* Greedy search in the first level, which determines the QoS of CDN providers*/
  **for all** $allocandi \in AlloCandi$ **do**
    $QoS \leftarrow QoS(allocandi, HisInfo)$;
    /* Group-based and K-means-based pruning */
    $Clusters \leftarrow PruningMethod()$
    /* Generate the bipartite graph $G$ */
    $G \leftarrow QoE(QoS, Clusters, ...)$
    $\pi', Q' \leftarrow GraphMatching(G)$
    /* Update the best policy $\pi$ */
    **if** $Q' > Q$ **then**
      $Q = Q', \pi = \pi'$

From the above analysis, we can conclude that: 1) there exists heterogeneity for QoE sensitivity, which can be used to generate a better viewer scheduling policy. 2) To leverage the heterogeneity, the scheduling approach needs to build up a precise QoE model, which should not only capture the non-linear relationship between QoS and QoE, but also well characterize the complex interactions among non-QoS and QoS factors. 3) The scheduling approach should satisfy the real-time requirements of the livecast scenario. At the same time, it should also be aware of the dynamics from CDN providers, which are both time-variant and workload-related.

## 4 SYSTEM OVERVIEW

The system architecture of HeteroCast is shown in Figure 5. As presented, HeteroCast is composed of the following parts: the QoS predictor, the QoE model, and the Decision model. The QoS predictor is used to characterize the dynamics of CDN performance, which will predict the QoS based on the decision and historical data; the QoE model provides the QoE estimation after receiving the QoS prediction; the decision module will take the QoE estimation as the inputs and output the scheduling decision for each request.

The logic of HeteroCast consists of two levels: the first level will traverse the allocation candidate (denoted as *AlloCandi*), which determines the QoS of each CDN provider. Each candidate in *AlloCandi* represents how many viewer requests will be scheduled to each possible CDN provider. For instance, suppose that there are 3 CDN providers (denoted as CDN A, B, and C), and one candidate (denoted as *allocandi*) is $(x\%, y\%, z\%)$, it means that x%, y% and z% of total viewers will be scheduled to CDN provider A, B, and C, respectively[5]. For the second level, HeteroCast will find the best scheduling decision for each request based on the first level's determination. These two levels will repeat until we obtain the best QoE. The workflow is shown in Algorithm 1. The intuition behind this design is: the CDN performance is influenced by the overall request number (i.e., the workload), but not which specific requests

[5]To reduce the computation overhead, the search space is discretized into a certain interval (e.g., 10%).

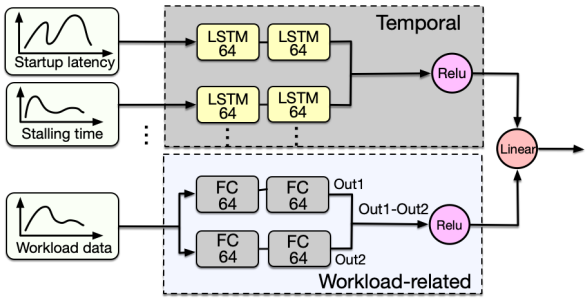



**Figure 6: The structure of the QoS predictor.**

| *Algo.* | LR | ARMA | FC | LSTM | Ours |
|---|---|---|---|---|---|
| **RMSE** | 0.0638 | 0.0202 | 0.0153 | 0.0124 | 0.0056 |

**Table 1: Performance of the QoS predictor**

are allocated. Therefore, the first-level will determine the QoS of CDN providers. At the same time, for each possible allocation, the best scheduling decision for each request can be optimally solved by treating it as a graph matching problem (i.e., the second level). It is also notable that we need to use some pruning methods (i.e., the Group-based and K-means-based pruning) to make HeteroCast satisfy the real-time requirement of the livecast scenario (details of the pruning methods are shown next).

## 4.1 Level 1: QoS Prediction

**QoS predictor.** With the inspiration provided by recent work [13, 21, 31], we propose to use a Deep Neural Network-based (DNN) model to predict the QoS values of different CDN providers. We use DNN mainly because it has both the generalization ability and high prediction precision. However, instead of directly using the exiting neural structures without any modification, we attempt to integrate our observed insights when designing the model. In detail, we build up two DNN blocks to separately characterize the temporal dynamics and workload-related dynamics. Figure 6 shows the model structure. For the temporal dynamics, since the Long-short-term-memory (LSTM) has shown great success in the time-series prediction problem, we use it to process the input data. For the workload dynamics, as we have observed that the CDN performance will drop dramatically when the workload exceeds a certain threshold (see Figure 4), we especially design the workload-related block to capture this "piece-wise linear" relationship. As shown in Figure 6, the historical workload data are input into two DNN structures independently, each of which uses two fully connected (FC) layers, and we then minus one of their outputs from another (i.e., $out1 - out2$), and the result is input into one FC layer which uses the "Relu" as the activation function. Since the "Relu" function will be zero when the input is less than zero (i.e., $out1 - out2 \leq 0$), it is quite suitable to represent the "piece-wise linear" relationship between QoS and workload. By using the above DNN blocks, our proposed QoS model can well characterize the dynamics of different CDN providers. We compare our QoS model with other baselines, e.g., Linear Regression (LR), Autoregressive Moving Average (ARMA), and use root mean square error (RMSE) as the estimation function. The results are in Table 1 (we only take the *StallFreq* as the comparison metric, and others lead to similar results), and we can see that our model has the highest precision. Notably, the model with the workload-related block significantly outperforms the model that only has the temporal block (i.e., the LSTM in Table 1).

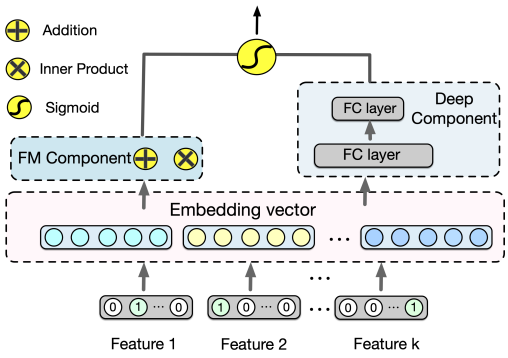



**Figure 7: The structure of the DeepFM-based QoE model.**

## 4.2 Level 2: QoE-aware Scheduling

The second level of HeteroCast can be divided into two stages, i.e., 1) the QoE modeling stage and 2) the scheduling stage. In the QoE modeling stage, HeteroCast will predict the QoE value of each viewer based on the QoS prediction from the first level. Then in the scheduling stage, HeteroCast will utilize the bipartite matching method to find the best CDN assignment for each request.

**The DeepFM-based QoE model:** First, we try to facilitate a more precise QoE model, which should be featured with the following properties: 1) it can well capture the non-linear relationship between QoS and QoE, 2) and also well characterize the complicated interactions between QoS and non-QoS factors.

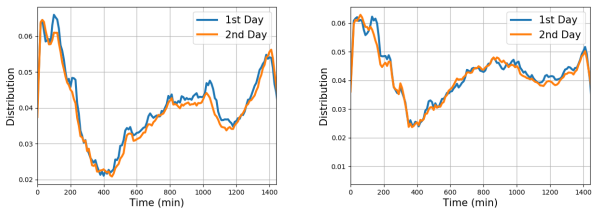
To address the above concerns, we propose to use the DeepFM to model the QoE of livecast viewers, and the DeepFM structure is shown in Figure 7. We can see that it consists of two parts: 1) the FM component, which extracts the feature interactions, 2) and the deep component, which extracts non-linear feature relationships. FM has been widely used in recommendation systems [10, 23]. The intuition of FM is to estimate the target value by modeling all interactions between each pair of features. Mathematically, given the target value $y_{fm}$ and a real-valued feature vector $x \in \mathbb{R}^n$ ($n$ is the feature number), the FM model can be formulated as:

$$y_{fm}(\mathbf{x}) = w_0 + \sum_{i=0}^{n-1} w_i x_i + \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \boldsymbol{v_i^T v_j} \cdot x_i x_j \tag{1}$$

where $w_0$ is the global bias, and $w_i$ is the linear weight of feature $i$. The $\boldsymbol{v_i^T v_j}$ term denotes the factorized interaction part, in which $\boldsymbol{v_i} \in \mathbb{R}^m$ is the embedding vector for feature $i$, and $m$ is the dimension of the embedding vector. Therefore, the learnable parameters of FM model are $w_0, w_i \in \mathbb{R}$ and $\boldsymbol{v_i} \in \mathbb{R}^m$. In addition to the FM component, DeepFM also uses the deep component to increase modeling ability for non-linearity relationships. Especially, since the FM and deep component share the same input embedding vector, DeepFM can learn the low- and high feature interactions simultaneously.

DeepFM only has linear complexity with feature number $n$: its FM component, which is responsible for the major complexity, can be efficiently computed as it satisfies the following equation:

$$\sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \boldsymbol{v_i^T v_j} \cdot x_i x_j = \frac{1}{2} \sum_{f=1}^{m} ((\sum_{i=0}^{n-1} v_i^f x_i)^2 - \sum_{i=0}^{n-1} v_{i,f}^2 x_i^2) \tag{2}$$




(a) The distribution of **group A** in two different days (b) The distribution of **group B** in two different days

**Figure 8: The distributions of the same group are consistent in different days.**

| ***Context*** | None | Device | Channel | (Device, Channel) |
|---|---|---|---|---|
| **std. of** $v_{tolerance}$ | 0.112 | 0.094 | 0.097 | 0.056 |

**Table 2: The std. of the $h_{tolerance}$ for different contexts.**

in which the $v_i^f$ is the $f$-th element of vector $\boldsymbol{v_i}$. This equation only has linear complexity in both vector dimension $m$ and feature number $n$ (i.e., its computation complexity is in $\mathbb{O}(mn)$). For more proof details of Eq.(2), we recommend readers to [22].

DeepFM is quite suitable for modeling the QoE of livecast viewers, as it has the following advantages. First, DeepFM naturally characterizes the complex relationship between multiple factors (e.g., the QoS factors and non-QoS factors). By introducing the embedding vector $\boldsymbol{v}$, the interactions between feature $i$ and feature $j$ can be easily modeled by calculating their inner-product (i.e., $\boldsymbol{v_i^T v_j}$). Second, DeepFM supports non-linearity modeling of different factors. By utilizing the deep component, DeepFM can perform the non-linear transformation on the latent space of the embedding vectors, and thus it can well capture the non-linear relationship between the features and target values (e.g., the relationship between QoS and QoE).

However, directly applying it in livecast scenario is still problematic: due to the massive user number, it is over-expensive to calculate and store this customized vector for each viewer. As a result, we need to use some pruning methods to reduce the overhead.

**Group-based pruning:** To tackle this problem, we first use a group-based pruning method. The critical insight that motivates us is: although the QoE of different viewers is heterogeneous, the viewers with the same "context" are more likely to have similar QoE patterns. In our problem, we consider two context features, i.e., the device type and channel category. We take the startup latency to illustrate this insight. We calculate average $v_{tolerance}$ of the viewers sharing different numbers of the context features, and present the standard deviation value (std.) in Table 2. We can see that viewers matching on both two context features (i.e., (device, channel)) have much smaller std., which demonstrates that these viewers have similar QoE patterns. Actually, similar conclusion has been obtained by previous work. For example, video sessions sharing the same access point and IP-prefix tend to have similar QoE [15, 16, 18, 24]; As a result, instead of training an embedding vector for each viewer, we will train the vector for each combination (device, channel category). In other words, the viewers in the same group are treated equivalently, and we will run DeepFM over the group rather than each individual request. We accept that this group-based pruning method may harm the precision of the QoE model, but it is essential for reducing the computation overhead.

| ***Algo.*** | LR | ARMA | SVM | FC | LSTM | DeepFM |
|---|---|---|---|---|---|---|
| **RMSE** | 0.3031 | 0.3715 | 0.2755 | 0.2641 | 0.2531 | 0.1418 |

**Table 3: Performance of our QoE model**

Additionally, since livecast viewers are sensitive to latency, we also need to run this DeepFM model in a real-time way. Different from some previous work [33], which aggregates all the requests first before making decisions, we need to predict the viewer number of each group and generate the scheduling policy in advance. Fortunately, predicting the size of each group is not that hard. Figure 8 presents how the distribution of viewers within the same group changes over time. We can see that the dynamics of their distribution are quite stable on different days for both groups A and B. Taking Figure 8(a) as an example, we can see that both in the 1st and 2nd day, the viewer number of group A at 400 min all takes about 2.2% of total viewers. This phenomenon indicates that we can cache the distribution of different groups in a **distribution lookup table**, through which we can directly query the viewer number of each group without aggregating the requests first. In our experiment, storing this table only takes about 100 KB.

To demonstrate the effectiveness of our proposed method, we compare the group-based DeepFM with other baseline algorithms, including both traditional methods and some DNN-based methods. The results are shown in Table 3. We can see that DeepFM can well predict the QoE value and significantly outperform other baselines.

So far, we have obtained a precise DeepFM-based QoE model, and we can also save the computation overhead by using a group-based pruning method and a distribution lookup table. Then we would like to compute the optimal assignment.

**Graph-matching-based scheduling:** The key idea of calculating the optimal assignment is to convert the problem into a bipartite graph matching problem, which can be efficiently solved in polynomial time [6, 17]. In detail, there will be four steps, and we present them in Figure 9.

▹ We first divide all the viewers into $n_b$ "buckets", and all the buckets have the same size. In Figure 9(a), all the viewers are divided into four buckets: three buckets belong to CDN A, one bucket belongs to CDN B (the "cluster" in Figure 9(a) will be explained later).

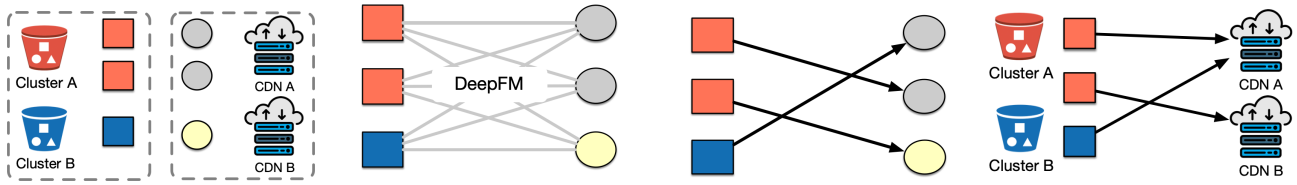
▹ We then construct an n-to-n weighted graph, in which the left nodes are the buckets belonging to different groups, while the right nodes are the buckets belonging to different CDNs. Especially, the edge $e_{ij}$ between two nodes is the QoE value when assigning the bucket (which belongs to group $i$) to the CDN $j$.

▹ After getting the weighted bipartite graph, we then calculate the optimal assignment by finding the best matching sub-graph[6]. Figure 9(c) shows an example.

▹ Finally, we convert the best matching sub-graph into the assignment decision. As shown in Figure 9(d), the final decision is that all viewers in Cluster B are assigned to CDN A, and the viewers in Cluster A are equally assigned to CDN A and CDN B.

As a result, the key problem now is how to set the bucket size. The criterion is that the viewers in the same bucket should be "similar" (e.g., the viewers in the same group). A straight-forward method is to set the bucket size as the number of viewers in the smallest group, and other groups and CDN providers are divided

[6]The best matching sub-graph is a graph where each node has only one edge, and the sum of total edge weights is maximized.




(a) Divide the users and CDNs into multiple "buckets"

(b) Generate the bipartite graph by using the DeepFM-based QoE model

(c) Calculate the best matching sub-graph

(d) Convert the best matching sub-graph into the final scheduling decision

**Figure 9: Scheduling viewers based on graph matching. Notably, to reduce the overhead of the graph-matching process, we further aggregate "similar" groups into clusters by using their embedding vectors. This figure shows that there are two clusters of viewers scheduled to two CDN providers. The clusters and CDNs are first divided into three "buckets", and the buckets with different colors mean that they have different properties (e.g., the buckets in grey color have different QoS from the bucket in yellow.).**

according to this size (in our experiment, we use the ceiling value, i.e., the smallest integer not less than the divided result). However, since the complexity of the fastest bipartite-matching algorithm is cubic in the bucket number $n_b$ (i.e., $\mathbb{O}(n_b^3)$), this division method may introduce too much computation overhead. For example, if the smallest group size is 0.1% (i.e., 0.1% of total viewers are in the smallest group), then there will be $n_b = \lceil \frac{100\%}{0.1\%} \rceil = 1000$ buckets ($\lceil * \rceil$ is the ceiling function), which is too time-consuming to be solved in a real-time way.

**K-means-based pruning:** Our idea for tackling this concern is to cluster the "similar" groups, and the similarity between the groups is estimated by the Euclidean distance of their embedding vectors. Mathematically, the distance between group $i$ and $j$ is defined as:

$$d = ||\boldsymbol{v_i} - \boldsymbol{v_j}||_2 \tag{3}$$

The groups will be aggregated into $K$ clusters, and $K$ is a hyper-parameter. The rationale behind this clustering is that the value of embedding vectors will affect the QoE prediction (i.e., the edge weight of the bipartite graph), so the groups with similar embedding vectors tend to have the similar QoE prediction value. There are plenty of algorithms that can be used for the clustering, and in this paper, we use K-means. Since the cluster size is much larger than the group size, the bucket number can be significantly reduced, and thus the best-matching sub-graph can be calculated efficiently.

# 5 EVALUATION

## 5.1 Methodology

**Implementation:** In our trace-driven experiments, we consider 5 CDN providers. For the first level, the searching space is descretized into 10% interval (i.e., a possible *allocandi* is (20%, 10%, 20%, 20%, 30%)). Therefore, there will be $C_9^4 = 126$ candidates in the out-loop.

For the QoS predictor, we use two LSTM layers (each with size 64) and one FC layer (with size 64) to construct the temporal block, and we use two FC layers (each with size 64) cascaded by one FC layer (with size 64) to build the workload block. The outputs of both blocks will be merged into one FC layer, which uses the "linear" activation function. Without additional explanation, all layers use "Relu" as the activation function. The structure is shown in Figure 6.

In the second level, the dimension of the embedding vector of DeepFM is set as $m = 128$, and its deep component consists with two FC layers (each with size 128 and 64); the outputs from FM- and the deep component will be concatenated together, and passed into the output layer (with size 1). All the feature layers use "Relu" except the output layer, which uses "sigmoid" function. The Details of DeepFM are shown in Figure 7. For the scheduling stage, we set cluster number as $K = 20$

**Baseline algorithms:** We compare HeteroCast with the following baseline algorithms:

▹*RoundRoubin:* One of the most widely-used scheduling method by nowadays livecast platforms. This algorithm will assign viewers to CDN providers randomly.

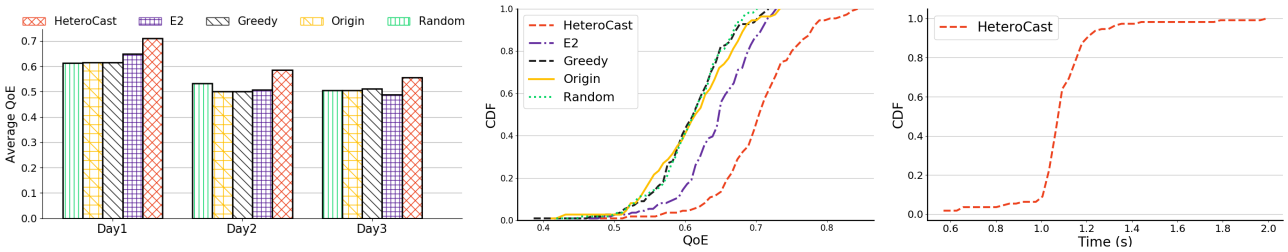
▹*Exploration and exploitation (E2):* One of the state-of-the-art algorithms proposed in [14, 16]. This kind of algorithm will schedule as many as viewers to the CDN that has the largest upper confidence bound (UCB). As recommended in [16], we choose the Discounted-UCB algorithm.

▹*Greedy:* This algorithm will predict the QoS of different CDN providers, and schedule as many viewers as possible to the CDN with the highest QoS values.

▹*The original algorithm:* The algorithm used by the data provider, and we can get it through the collected data.

## 5.2 Results and Discussion

We first use three days of data to evaluate the performance of HeteroCast. The results are shown in Figure 10(a), and we can derive the following observations. First, HeteroCast achieves the best performance in all three days. We can see that from day1 to day3, HeteroCast outperforms the second best algorithm (i.e., E2, Random, and Greedy) by 9.61%, 10.09%, and 8.87% respectively. This is expected, since by using a more precise QoE model, HeteroCast can utilize the heterogeneity of different users and well address the shortcomings of previous algorithms. Second, we can find that among all baselines (i.e., E2, Greedy, Origin, and Random), no algorithm can consistently outperform others. For example, on day1, the best baseline is the E2 algorithm, while on day2 and day3, the best algorithm becomes the Random and Greedy. The reason behind this phenomenon is: since all baselines try to optimize QoE through optimizing some proxies (such as different QoS factors), they are all separated from getting the best average QoE. In contrast, HeteroCast directly uses the QoE as the optimization objective function, which ensures that it can maintain good performance even in different scenarios. Moreover, instead of only comparing the average QoE of HeteroCast, we further evaluate HeteroCast by comparing the QoE obtained at each step of day1 with other algorithms, and present the results in the form of CDF. The results are shown in Figure 10(b), and the righter the curve locates, the better the algorithm is. We can see that HeteroCast outperforms other baselines




(a) The average QoE achieved by HeteroCast and other baselines.

(b) The QoE distribution of HeteroCast and other baselines at Day1.

(c) The decision time of HeteroCast.

**Figure 10: Comparing HeteroCast with other baseline algorithms. As denoted, we can find that HeteroCast improves the average QoE by 8.87% to 10.09%. Especially, the QoE distribution of HeteroCast demonstrates that it outperforms the baselines at almost all the steps. Meanwhile, the decision time of HeteroCast also illustrates that it can schedule viewers in a real-time way.**

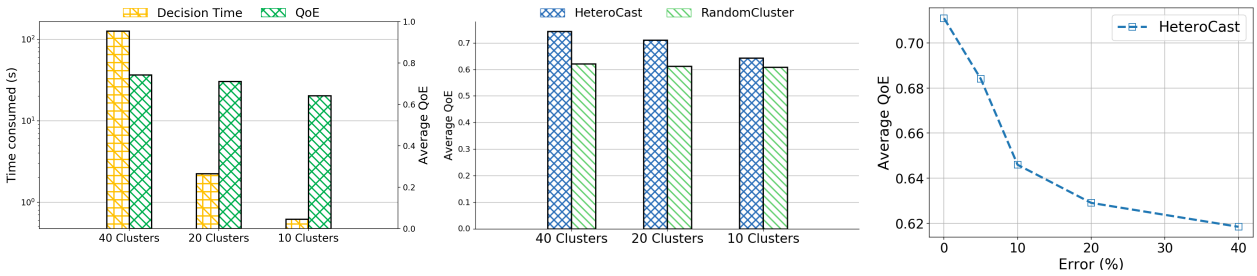



**Figure 11: The HeteroCast with different cluster number $K$.**

**Figure 12: Comparing K-means pruning method with the random cluster method.**

**Figure 13: The influence of the prediction error.**

at almost all the steps. In addition, we also analyze the decision time of HeteroCast, and the results are provided in Figure 10(c). As denoted, we can see that most of the decisions consume less than 1.4 seconds, while all the decisions are finished within 2 seconds. This actually indicates that HeteroCast can work in a real-time way, which satisfies the systematic requirement of the livecast scenario.

## 5.3 HeteroCast Deep Dive

In this section, we further study how different factors influence HeteroCast performance. We first analyze the influence of the cluster number $K$. In detail, we investigate both the performance and decision time. Figure 11 shows the results, and we can derive that as the cluster number of K-means decreases from 40 clusters to 10 clusters, the average decision time of HeteroCast decreases significantly (from 120 seconds per step to 0.5 seconds per step). This is because by aggregating more items together, the bucket number in the bipartite matching process will be reduced significantly. However, although we can speed up HeteroCast by using fewer clusters, its performance may suffer degradation at the same time. As shown in Figure 11, we can see that comparing to the HeteroCast with 40 clusters, the average QoE of HeteroCast with 20 clusters and 10 clusters will degrade about 4.4% and 13.5% respectively. This is rationale: with the cluster number decreasing, the viewers with large heterogeneity are more likely to be "mistakenly" aggregated into the same cluster, which therefore results in a sub-optimal scheduling decision. Considering the performance and the computation overhead, we finally choose $K = 20$. Moreover, we further study the effectiveness of our K-means-based pruning method by comparing it with the random cluster method (i.e., instead of using K-means, this method aggregates groups randomly). The results are shown in Figure 12, and we can find that for different cluster numbers, our K-means method significantly outperforms the random method. Especially, we observe that the performance of the random cluster is almost equivalent even for different cluster numbers.

To better illustrate the importance of our DeepFM-based QoE model, we also investigate the impact of its prediction error by manually introducing noises, and the results are presented in Figure 13. We can obtain that with the prediction error increasing from 5% to 40%, the performance of HeteroCast will decrease by 3.7% to 12.9%. It is also notable that the performance declines quickly at the beginning, and then changes much more slowly. A possible reason is: when the QoE model's prediction error grows to a certain level, HeteroCast will no longer characterize the heterogeneity of different viewers, and it will make decisions like a random strategy.

## 6 CONCLUSION

We propose HeteroCast, a novel algorithm for the livecast viewer scheduling problem. Different from previous scheduling methods, which either have no consideration of the viewer heterogeneity or use inaccurate QoE model as the optimization objective, HeteroCast addresses the above problems by using a DeepFM-based QoE model and a graph-matching technique. Through extensive experiments on real-world traces, HeteroCast is demonstrated to significantly outperform the existing viewer scheduling methods.

**Acknowledgement:** This work is supported by NSFC under Grant 61936011, 61521002 the National Key R&D Program of China (No. 2018YFB1003703), and Beijing Key Lab of Networked Multimedia.

## REFERENCES


[1] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhili Zhang. 2012. Unreeling netflix: Understanding and improving multi-CDN movie delivery. (2012), 1620–1628.

[2] 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. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 339–350.

[3] Yonghwan Bang, June-Koo Kevin Rhee, KyungSoo Park, Kyongchun Lim, Giyoung Nam, John D Shinn, Jongmin Lee, Sungmin Jo, Ja-Ryeong Koo, Jonggyu Sung, et al. 2016. CDN interconnection service trial: implementation and analysis. *IEEE Communications Magazine* 54, 6 (2016), 94–100.

[4] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. 2018. Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix cdn. *ACM SIGCOMM Computer Communication Review* 48, 1 (2018), 28–34.

[5] Fei Chen, Cong Zhang, Feng Wang, and Jiangchuan Liu. 2015. Crowdsourced live streaming over the cloud. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2524–2532.

[6] Michael L Fredman and Robert Endre Tarjan. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)* 34, 3 (1987), 596–615.

[7] Thiago Guarnieri, Idilio Drago, Alex B Vieira, Italo Cunha, and Jussara Almeida. 2017. Characterizing QoE in large-scale live streaming. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 1–7.

[8] Fatima Haouari, Emna Baccour, Aiman Erbad, Amr Mohamed, and Mohsen Guizani. 2019. QoE-Aware Resource Allocation for Crowdsourced Live Streaming: A Machine Learning Approach. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[9] Jian He, Di Wu, Yupeng Zeng, Xiaojun Hei, and Yonggang Wen. 2013. Toward optimal deployment of cloud-assisted video distribution services. *IEEE transactions on circuits and systems for video technology* 23, 10 (2013), 1717–1728.

[10] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 355–364.

[11] https://www.businessofapps.com/data/twitch statistics/. 2020. Twitch Revenue and Usage Statistics.

[12] https://www.chinainternetwatch.com/30115/kwai-dec 2019/. 2020. Kuaishou live broadcast DAU exceeded 100 million in 2019.

[13] Alexis Huet and Dario Rossi. 2019. Demo Abstract: Explaining Web users' QoE with Factorization Machines. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 945–946.

[14] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, et al. 2016. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 286–299.

[15] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. 2016. {CFA}: A practical prediction system for video qoe optimization. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 137–150.

[16] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 393–406.

[17] Roy Jonker and Anton Volgenant. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38, 4 (1987), 325–340.

[18] Hongqiang Harry Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitendra Padhye, and Ming Zhang. 2016. Efficiently delivering online services over integrated infrastructure. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 77–90.

[19] Hongqiang Harry Liu, Ye Wang, Yang Richard Yang, Hao Wang, and Chen Tian. 2012. Optimizing cost and performance for content multihoming. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 371–382.

[20] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*. 270–288.

[21] Haitian Pang, Cong Zhang, Fangxin Wang, Han Hu, Zhi Wang, Jiangchuan Liu, and Lifeng Sun. 2018. Optimizing Personalized Interaction Experience in Crowd-Interactive Livecast: A Cloud-Edge Approach. In *Proceedings of the 26th ACM international conference on Multimedia*. 1217–1225.

[22] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[23] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 635–644.

[24] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. (2016), 272–285.

[25] Feng Wang, Jiangchuan Liu, Minghua Chen, and Haiyang Wang. 2014. Migration towards cloud-assisted live media streaming. *IEEE/ACM Transactions on networking* 24, 1 (2014), 272–282.

[26] Fangxin Wang, Cong Zhang, Jiangchuan Liu, Yifei Zhu, Haitian Pang, Lifeng Sun, et al. 2019. Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized QoE. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 910–918.

[27] Zhi Wang, Lifeng Sun, Chuan Wu, Wenwu Zhu, and Shiqiang Yang. 2014. Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 91–99.

[28] Ting Yue, An-Ming Wei, Hong-Bo Wang, Xiang-Dong Deng, and Shi-Duan Cheng. 2016. A comprehensive data-driven approach to evaluating quality of experience on large-scale Internet video service. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 1479–1486.

[29] Huaizheng Zhang, Han Hu, Guanyu Gao, Yonggang Wen, and Kyle Guan. 2018. DeepQoE: A unified framework for learning to predict video QoE. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 1–6.

[30] Rui-Xiao Zhang, Tianchi Huang, Ming Ma, Haitian Pang, Xin Yao, Chenglei Wu, and Lifeng Sun. 2019. Enhancing the crowdsourced live streaming: a deep reinforcement learning approach. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 55–60.

[31] Rui-Xiao Zhang, Ming Ma, Tianchi Huang, Haitian Pang, Xin Yao, Chenglei Wu, Jiangchuan Liu, and Lifeng Sun. 2019. Livesmart: A QoS-Guaranteed Cost-Minimum Framework of Viewer Scheduling for Crowdsourced Live Streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*. 420–428.

[32] Rui-Xiao Zhang, Ming Ma, Tianchi Huang, Haitian Pang, Xin Yao, Chenglei Wu, and Lifeng Sun. 2020. A Practical Learning-based Approach for Viewer Scheduling in the Crowdsourced Live Streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 16, 2s (2020), 1–22.

[33] Xu Zhang, Siddhartha Sen, Daniar Kurniawan, Haryadi Gunawi, and Junchen Jiang. 2019. E2E: embracing user heterogeneity to improve quality of experience on the web. In *Proceedings of the ACM Special Interest Group on Data Communication*. 289–302.

[34] Yifei Zhu, Jiangchuan Liu, Zhi Wang, and Cong Zhang. 2017. When cloud meets uncertain crowd: An auction approach for crowdsourced livecast transcoding. In *Proceedings of the 25th ACM international conference on Multimedia*. 1372–1380.