Machine Learning at the Mobile Edge: The Case of Dynamic Adaptive Streaming Over HTTP (DASH)

Rasoul Behravesh, Akhila Rao, Daniel F. Perez-Ramirez, Davit Harutyunyan, Roberto Riggio, *Senior Member, IEEE*, and Magnus Boman

***Abstract*—Dynamic Adaptive Streaming over HTTP (DASH) is a standard for delivering video in segments and adapting each segment's bitrate (quality), to adjust to changing and limited network bandwidth. We study segment prefetching, informed by machine learning predictions of bitrates of client segment requests, implemented at the network edge. We formulate this client segment request prediction problem as a supervised learning problem of predicting the bitrate of a client's next segment request, in order to prefetch it at the mobile edge, with the objective of jointly improving the video streaming experience for the users and network bandwidth utilization for the service provider. The results of extensive evaluations showed a segment request prediction accuracy of close to 90% and reduced video segment access delay with a cache hit ratio of 58%, and reduced transport network load by lowering the backhaul link utilization by 60.91%.**

***Index Terms*—Video streaming, DASH, caching, prefetching, machine learning, MEC, 5G.**

I. Introduction

THE POPULARITY of high-quality video streaming has made video content providers the predominant bandwidth consumers on the Internet. There is also an ever-increasing interest in live video streaming scenarios such as sports events, live performances, news, and e-gaming. More specifically, the widespread deployment of high-capacity mobile networks like 4G and 5G has boosted the adoption of video streaming scenarios over dynamic wireless networks.

Currently, Dynamic Adaptive Streaming over HTTP (DASH) is the dominant video delivery standard, adopted by a sizeable body of video content providers [1]. The main idea behind DASH is to adapt to the varying availability of network bandwidth to provide an acceptable level of video quality while minimizing stalls. This is achieved by dynamically adjusting the bitrate [2] to the availability of bandwidth. DASH [3] dictates that videos in the DASH server be split into equal duration segments, each of them available at multiple video bitrates (indicating video qualities). Once the DASH client initiates video streaming, the video segments are requested and downloaded sequentially from the DASH server. An Adaptive Bitrate (ABR) algorithm at the DASH client selects the bitrate of the segment requested. This algorithm decides on the bitrate of a segment given multiple parameters, such as network status, application state, and buffer state at the client (cf. [4]). The idea is to request the highest quality that the network can deliver without the risk of stalls or reduced viewing experience due to frequent jumps in video quality.

A. Problem

With higher bandwidth availability on 4G and 5G, an increasing number of users streaming video are now on mobile wireless networks. This increases the challenge for DASH to adapt to the rapidly changing network state in mobile networks. Furthermore, the streaming of popular live videos by many users results in redundant usage of the backhaul (BH) bandwidth, which is a valuable resource for Mobile Network Operators (MNO), which can become overloaded quickly [2], [5]. However, with the standardization and adoption of Multi-access Edge Computing (MEC) technology and deployment of MEC nodes close to mobile base stations, there is an opportunity to provide better streaming services to users in live streaming as well as Video On Demand (VOD). The current implementation of DASH-based streaming cannot rely on its own to tackle the challenges of highly dynamic networks that cause stalls and rapid bitrate changes, as well as the challenge of inefficient usage of the backhaul for MNOs due to the transmission of redundant live video segments. When it comes to video content in general, bringing the content closer to the users based on the popularity of videos has been achieved using Content Delivery Networks (CDNs) to reduce latency [6], [7]. While CDNs do relieve the larger network by

Manuscript received 14 May 2022; revised 8 July 2022 and 18 July 2022; accepted 20 July 2022. Date of publication 26 July 2022; date of current version 31 January 2023. This work has been funded by the EU's Horizon 2020 project 5G-CARMEN (grant no. 825012), by the H2020 AI@Edge project (grant. no. 101015922), and by the Swedish Foundation for Strategic Research (SSF) Time Critical Clouds project (grant. no. RIT15-0075). We thank Dr Rebecca Steinert for important guidance and feedback in the writing process. The associate editor coordinating the review of this article and approving it for publication was T. Wauters. *(Rasoul Behravesh and Akhila Rao contributed equally to this work) (Corresponding author: Rasoul Behravesh.)*

Rasoul Behravesh is with the Digital Socity Center, SNESE Unit, Fondazione Bruno Kessler, 38121 Trento, Italy, and also with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40126 Bologna, Italy (e-mail: rbehravesh@fbk.eu).

Akhila Rao and Daniel F. Perez-Ramirez are with the Connected Intelligence, Research Institutes of Sweden AB, 16440 Stockholm, Sweden, and also with the Department of Computer Science, KTH Royal Institute of Technology, 11428 Stockholm, Sweden (e-mail: daniel.perez@ri.se; akhila.rao@ri.se).

Davit Harutyunyan is with Corporate Research, Robert Bosch GmbH, 70465 Gerlingen, Germany (e-mail: davit.harutyunyan@de.bosch.com).

Roberto Riggio is with the Information Engineering Department, Università Politecnica delle Marche, 60121 Ancona, Italy (e-mail: r.riggio@univpm.it).

Magnus Boman is with the Department of Computer Science, KTH Royal Institute of Technology, 11428 Stockholm, Sweden (e-mail: mab@kth.se).

Digital Object Identifier 10.1109/TNSM.2022.3193856

bringing popular content closer to the users, in the context of MNOs, they do not reduce the backhaul bandwidth utilization of MNOs [8], [9] and do not leverage the resources available at the mobile edge MEC nodes.

B. Solution

MEC provides processing, storage, and virtualization resources at the mobile edge, in the proximity of the mobile end users, and hence most exposed to dynamic network state variations. It also offers a set of services, such as Radio Network Information Service (RNIS), location service, and traffic offloading, to name a few [10], at the edge that can be utilized by over-the-top applications (OTTs) such as HBO Max, Hulu, Netflix, and others to improve their service delivery measurably.

A promising scenario for MEC is to employ it for prefetching and caching video content at the network edge. Prefetching is a technique to move the content closer to the end-user (to the Radio Access Network (RAN) in our scenario) before being requested by the user. Prefetching video segments during playback of an ongoing video reduces access delay and improves performance for the user. Moreover, if the video streaming is from a use case scenario mentioned earlier, e.g., sports events, live performances, or e-gaming, then it also reduces backhaul bandwidth utilization. By implementing prefetching at the MEC, we allow the RAN to potentially stream a segment to the user while the next segment is being fetched from the video server. And by also implementing a video segment cache, the MNO backhaul bandwidth utilization from live streaming videos can be significantly reduced [8], [11]. However, considering that the caching storage at the edge is limited and potentially shared among a large number of applications and users, we need to make intelligent prefetching and caching decisions with existing constraints.

We propose a solution based on prefetching and caching at the mobile edge to bring video segments closer to the users, as well as cache them for repeated requests in the case of live streaming scenarios. The live streaming scenarios we consider are in practice near-live scenarios due to inherent delays in digital video streaming with streams often delayed by a few segments, because of which we assume that at least one segment ahead is created at the video server at the time of download of the current segment. We emphasize here that the prefetching and caching are not based on the popularity of the entire video stream. It is instead done after a user initiates a video stream on a per-segment basis, with segments being prefetched and cached as the streaming progresses. While the resources available at the edge can be used to bring video content even closer to mobile users, the scale of resources available at the mobile edge is much lower in comparison to CDNs. So instead of moving content closer to the edge based on long-term video popularity, we apply prefetching, with benefits to live scenarios as well as VOD.

Prefetching involves proactively fetching a segment before the user requests it. However, since in DASH, each video segment is requested at a particular bitrate, we need to be able to predict this bitrate of the expected segment request from a client. We propose methods to make these predictions at the MEC, deployed in the Radio Access Network (RAN), by leveraging the metrics available there. As dictated by a recent MEC standard [12], each MEC node co-located with one or more mobile base stations has access to RAN metrics through the RNIS API. We use cross-layer monitoring, combining metrics from the application layer (metrics specific to the DASH video application) with metrics from the RAN (accessed through RNIS) and evaluate their ability to add insight to the prediction of segment bitrate, which is also an application-specific metric.

While the focus of our approach is on live streaming scenarios with video segment overlap, it also provides benefits to VOD scenarios. In live streaming we prefetch currently relevant segments at the predicted bitrate to provide savings in backhaul bandwidth utilization as well as move the content closer to the users. In a VOD scenario, however, we still reap the benefits of reduced delay in segment download because we move segments (at the predicted bitrates) closer to the clients.

In summary, our proposed solution consists of two algorithms:

- To predict the bitrate of expected video segment.
- Intelligently decide on which requests to prefetch given resource constraints at the edge.

C. Methodology

We explore both classical Machine Learning (ML) and neural networks approaches for predicting segment request bitrates, using metrics from cross-layer monitoring (RAN and application metrics). We begin with an exploratory analysis of the dataset and then provide an in-depth analysis of the associated ML task of predicting bitrate of segment requests. Moreover, we propose novel prefetching methods to maximize cache hit and byte hit ratios at the MEC under varying constraints. In particular, we use Integer Linear Programming (ILP) techniques for studying the trade-offs between different prefetching approaches as well as online video transcoding at the edge. By employing transcoding, we can encode a high bitrate segment into a lower bitrate one. Thus, we are able to use processing resources at the MEC in order to generate a desired bitrate. We consider the cost to transcode as well as the cost to prefetch in our prefetching optimization model. Finally, we propose a heuristic to combat the scalability issue of the proposed ILP model, reaching a near-optimal solution.

We use datasets generated from ns-3 [13] simulations to train and validate our ML models as well as to evaluate the bitrate predicting and prefetching approach.

D. System Architecture

Fig. 1 illustrates our considered system architecture, which constitutes four main components: a remote video server deployed in the Data Network (DN), the 5G Core (5GC), MEC servers, and DASH clients, which are running on the users. The video server hosts all the videos, with each video available in segments at multiple different bitrates. In live streaming, it is hosted at the service provider's server, which then plays the role of the video server. The prefetcher algorithm situated

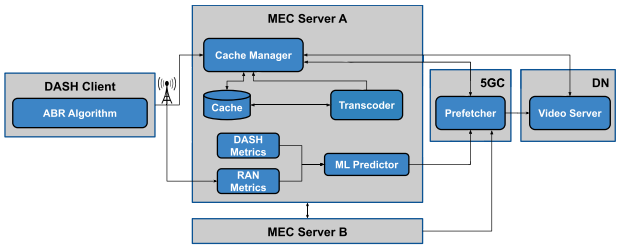

Fig. 1. Our system architecture for a MEC-enabled mobile network, with a video server providing streaming service to a DASH client. The components detail the various functional units necessary for monitoring metrics, predicting segment requests, prefetching, as well as all supporting components required for these tasks.

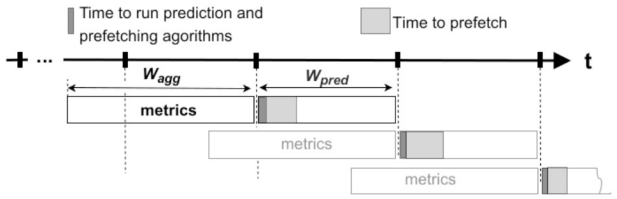

Fig. 2. Metrics aggregated over an aggregation window, used to predict bitrate of requests in upcoming prediction window.

in the 5GC gathers prediction inferences of all the connected MEC nodes at the edge and makes decisions about prefetching video segments to the MEC nodes or directly serving the client from the video service. This setup requires using a DASH application proxy at the MEC, which hosts this proxy for OTT services. This proxy is needed at the MEC for it to have an observation point (for metrics) and a control point (to prefetch) that sits between the clients and the server. In our proposed system architecture, the proxy also maintains the state of each client it serves and computes application metrics at the MEC that would otherwise only be available at the DASH client, such as buffer occupancy. This is done by maintaining the state of the duration of segments served and the actual time passed since streaming began. An alternate implementation approach is for the DASH clients to insert this information into their segment request messages. The hosting of a proxy at the MEC is a potential MEC service provided by the MNOs to OTT services to help them meet their service requirements to their clients [14]. It is worth mentioning that the state to be maintained for each client is around 20 bytes, computed based on the number of features logged per client. Therefore, with a relatively small amount of state memory (e.g., 10 MB) we could store the state of 500 clients, which would be reasonable for the users potentially served through a MEC.

Given MEC constraints, the prefetcher decides which of the predicted segment requests shall be fetched. It can also be that the prediction for the bitrate of the segment was incorrect, and the bitrate prefetched is not the one that gets requested. This means that the cache at the MEC may not have prefetched every requested segment at the requested bitrate. In such a case, a cache miss occurs, and the client is directly served from the main video server.

If the predicted request can be satisfied by segments already prefetched and placed in the cache, then it can be served from the cache. If the cache cannot fulfill the request, the prefetcher makes one of the following decisions: (i) prefetch the predicted segment requests with the predicted bitrate, (ii) prefetch at a higher bitrate, and then transcode the segment to serve user requests of lower bitrates, (iii) redirect the request to one of the neighbor MEC servers that can have the same or even a higher bitrate that can be transcoded, or (iv) redirect the request to the video server. It is important to mention that this process is performed after handling the request for a segment and before the next segment is requested.

In what follows, the segment bitrate prediction problem is presented in Section II, following by the prefetching algorithm in Section III. The process of dataset generation for our experiments is described in Section IV, following by the evaluation of the ML models used for bitrate prediction presented in Section V. The evaluation of the ILP and heuristic for prefetching decisions are given in Section VI. Relevant research findings are presented in Section VII, before conclusions in Section VIII.

II. Approach: Video Segment Prediction

A. Problem Formulation

In DASH video streaming, segments are requested sequentially by the clients at the bitrate chosen by the client's ABR. Once a segment is downloaded or fetched, the ABR algorithm at each client decides, not only the bitrate of the next segment, but also when it should be requested. Considering a limited buffer at the client and the risk of stalls, the ABR algorithm decides when to request the next segment based on the status of its buffer. In our approach, the prefetching of a segment is also done in sync with the client's expected requests, so segments are prefetched to the MEC only when they are expected to be needed on the client side. The time between requests is not constant and decided by the ABR algorithm in the client. The requests from clients are also not synchronized and spread over time. The goal of the prefetching algorithm is to intelligently regulate the prefetching of expected segments from the different clients to the MEC nodes at the edge, taking into account the scarcity of resources at the edge. It takes a finite amount of time (discussed in Section VI) to process these requests and make a decision which adds a cost to the process. We thus aggregate requests from all active clients over a short time window for decision making. The choice of window size is evaluated and motivated further below.

Fig. 2 shows the timeline of how bitrate predictions and prefetching are done in time windows. Periodically every W_{pred} seconds (2 seconds in our implementation), the MEC node uses the previous W_{agg} seconds (16 seconds in our implementation) of cross-layer metrics and predicts the bitrate of the next request from each client. These requests, predicted for all active clients served by a MEC node, are combined and passed on to the prefetching agent, which makes the decision of which of these requests to prefetch and then prefetches the segments. The prediction and prefetching decisions are made as indicated in the legend of Fig. 2, and the same goes for the time to prefetch. This time to prefetch at the MEC is typically much lower than the time to stream it over the radio

TABLE I
INPUT FEATURES TO THE LEARNING MODEL AND THE OUTPUT FEATURE TO BE PREDICTED

Input features	**Description**
Last #seg.	Number of requests sent out in the previous window.
Last bitrate	Bitrate of the previous segment request.
Seg. thput	Throughput over the last downloaded segment.
Window thput	Throughput over the last 10 s.
Buffered Bytes	Bytes in the video playback buffer.
Buffered #seg.	Number of segments in the video playback buffer.
DL RSRP	Downlink reference signal received power.
DL SINR	Downlink signal to interference and noise ratio.
DL MCS	Downlink modulation and coding scheme.
DL thput	Downlink MAC throughput.
Output value	**Description**
Next bitrate	Predicted bitrate of the segment to be requested next.

network since the bandwidth of the backhaul is much larger than the bandwidth of the mobile wireless channel [8]. This means that a segment is likely to be prefetched at the MEC before a client sends out the next request. However, if a client requests a segment before the prefetcher is able to prefetch it then the request is forwarded to the video server instead of being served form the MEC resulting in a MEC cache miss.

The cross-layer metrics aggregated in W_{agg} are the input features to the ML algorithm, and the bitrate of the one step ahead expected segment request is the output feature. The features used have been listed and described in Table I. There are two categories of input features. RAN metrics from the PHY and MAC layers and application metrics from the DASH application. The input metrics also contains one step prior bitrate of the previous segment request.

As mentioned before, the requests from clients are not regular and hence creating a periodic W_{pred} window means that we might have samples where we see that more than one segment got requested, or even that no segments were requested. If we create larger W_{pred} windows then most windows would be populated, but it would result in several windows having more than one request. If W_{pred} is too large then the ML algorithm would need to predict the bitrate and number of segments expected to be requested at each inference and decision instant, and in the right order, which is challenging and reduces the prediction accuracy as compared to only predicting one step ahead at a time. If W_{pred} is too small then we would not aggregate enough requests from multiple clients, and increase the cost of computing the prefetching decision in terms of compute and time resources. These considerations have been taken into account in the selection of W_{pred}.

Since the MEC aggregates expected requests from all clients it serves to make a decision at the prefetcher, most decision instants have many expected segment requests. However, the segment bitrate predictor makes predictions per user and does not need to predict every W_{pred}. If an already prefetched segment is yet to be requested by the client, then new predictions are not made until the segment in the cache is requested. This time between segment requests can vary depending on the video buffer occupancy, and on the differences between estimated channel data rate and achieved data rate, making the download of a segment faster or slower than expected. The dataset created for the validation of our prediction approach excluded these empty W_{pred} windows wherein nothing is predicted.

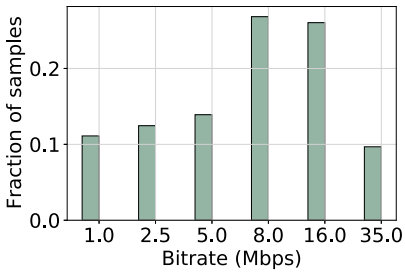

Fig. 3. The requested bitrate class distribution.

To summarize, the prediction problem formulation is to predict *next bitrate*, for each client, every W_{pred}, using as input the features in W_{agg}. The overall complexity of the prediction task scales linearly with the number of clients in the network. However, since the prediction task for each client can be run independently, it can be parallelized to run over distributed compute resources.

B. Proposed Solution Using Machine Learning

We use a supervised learning approach and model the learning problem as a multi-class classification problem, with each class being one of the 6 available segment bitrates. The range of values available for bitrate varies between implementations. The set of available bitrates for each segment and hence the categorical outcomes are {1, 2.5, 5, 8, 16, 35} Mbps, with the higher rates corresponding to HD, 2K, and 4K video qualities, respectively. Fig. 3 shows the histogram of occurrences of bitrate in the training dataset for each class. We note that while there is some class imbalance between the six classes, it is not high and hence better for us to use the dataset with its existing class distribution without the need for imbalance handling methods. Note, that the cost of misclassification from any class to any other class has the same impact on our approach and hence it is not important to study the impact of class imbalance.

We explore both ensemble tree-based classical ML models as well as neural network based models. We use Random Forest (RF), Gradient-Boosted trees (XGB), multi-layer perceptron (MLP) and Long-short term memory (LSTM). RF is a method used in prior research in this domain with good results [15], [16]. We explore gradient-boosted trees [17] due to their consistently good performance over structured tabular data [18]. We observed in our scoping review (Section VII) that tree-based approaches constitute the family of most used methods, in part due to fast training and with few hyperparameters to tune. They also have the advantage of being explainable, via feature importance. While expensive without much benefit given small datasets, deep neural network models can be pre-trained using, e.g., simulation data and then through transfer learning tuned to the varied scenarios in which they are deployed. These advantages motivated us to explore their efficacy and compare them against RF and XGB. The data

TABLE II
ARCHITECTURE AND HYPERPARAMETERS USED FOR THE M MODELS TRAINED TO PREDICT NEXT SEGMENT BITRATE

RF and XGB	
Estimators	350
Max depth	25
Min samples per leaf	2
MLP and LSTM	
Hidden layers	2
Num. neurons in hidden layers	MLP (30, 15) LSTM (30, 15)
Learning rate	0.01
Batch size	32
Epochs	1000
Activation function	ReLU
Loss function	Categorical cross-entropy

we have is time series and LSTM is an approach designed to natively handle such data through its recurrent network architecture. LSTM models, however, have a large number of parameters, require even larger datasets to learn from, and are relatively opaque.

Table II lists the architecture and hyperparameters used in our models. We observed that the results of the model were not very sensitive to hyperparameter tuning and hence a detailed analysis has been left out of the paper. We have made the dataset used and all the code open-sourced [19] to enable reproducibility.

Baselines: To test the value of our ML model we use a persistent prediction baseline (Baseline 1), where the next bitrate is predicted to be the same as the previous one. Such an indiscriminate predictor baseline for time series data is commonly used to assess the additional prediction power over a simple low-complexity one. Such a naive prediction could be useful for slow-changing networks. However, in real networks under dynamic conditions, the bitrate is frequently changing and hence more challenging to predict.

Additionally, to have a fair comparison of our work with existing work we need to evaluate our approach with work that has been conducted in a setup with a similar architecture where application level predictions of one step ahead segment bitrate are made at the edge. We use the recent work of Kheibari and Sayýt [20] who have used an LSTM model to obtain segment bitrate predictions with the same number of bitrate classes as in our evaluations. They use the same DASH application metrics as we do with a one segment history. In our work, in addition to this we also incorporate radio metrics from the base station to provide additional cross-layer monitoring information that could increase the information in the data about client's perceived channel bandwidth. This result from [20] provides a baseline with which we compare our work. They report results with a prediction accuracy of 70% when using an LSTM model, which we use as a second baseline for our evaluations (Baseline 2).

III. APPROACH: SEGMENT REQUEST PREFETCHING

A. Problem Statement

As shown in Fig. 4, our envisioned mobile network is composed of a 5G Core and multiple gNBs, with gNBs collocated with MEC servers. The MEC servers are characterized

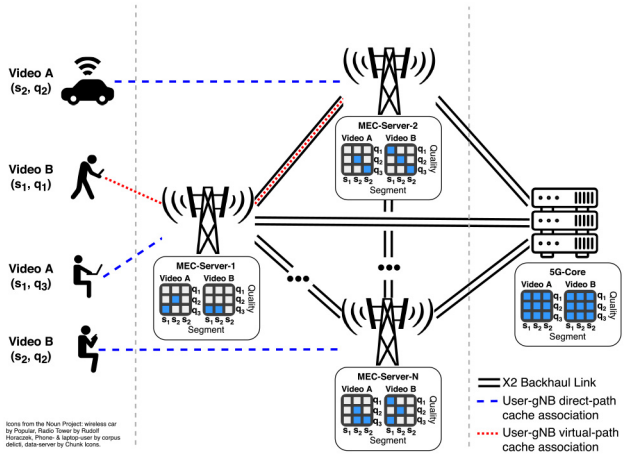

Fig. 4. Sample mobile network model, user request model, and video segment prefetching.

by processing, memory, and storage resources. While these resources are scarce at the edge they become abundant as we draw near the core. Consequently, resource provisioning at the core becomes much cheaper than that of the network edge, though the former imposes higher transport network usage.

We assume that at any given time, a set of requests will be issued from the users asking for a set of segments, possibly in different bitrates. The ML model proposed in Section III is responsible for predicting bitrate of the video segments. After obtaining prediction outcome from the ML model, the prefetching method (i.e., ILP or heuristic) decides whether to prefetch the predicted video segment(s) in specific bitrates to the network edge or, if available, to transcode a higher bitrate segment available at the network edge to make sure that the expected bitrate of the user is satisfied while network resources are used properly.

Depending on the segment duration, segment bitrate, and availability of the substrate network resources, there might be multiple prefetching alternatives, each in favor of optimizing certain aspects of the network. The problem of joint video segment prefetching, transcoding, and resource allocation is formally articulated as follows.

Given: a 5G network composed of MEC servers collocated with gNBs and the 5G Core, interconnected through backhaul transport network links. And a set of users that are associated with gNBs, making video segment requests each in a specific bitrates.

Find: joint video segment prefetching, transcoding, and resource allocation.

Objective: maximize (i) the cache hit ratio and (ii) the byte hit ratio. We define the cache hit ratio as the number of requests served from the edge (whether directly from the same gNB, from neighbor gNBs, or using transcoding) divided by the number of requests issued to the network. Similarly, the byte hit ratio is defined as the number of bytes served from the edge divided by the total number of bytes requested by the users.

B. Mobile Network Model

Let $G = \langle N, E \rangle$ be an undirected graph modeling the mobile network, where N represents the set of computing

TABLE III
MOBILE NETWORK PARAMETERS

Parameters	Description
$G = \langle N, E \rangle$	Graph representing the mobile network.
N	Set of nodes that can host videos $N = M \cup C$
E	Set of links connecting the nodes in G.
M	Set of MEC nodes in G.
C	Set of core nodes/servers in G.
V	Set of videos.
S^v	Set of segments of video $v \in V$.
$Q^{v,s}$	Set of available bitrates for each segment $s \in S^v$ of video $v \in V$. Bitrates are in order from the lowest to the highest $q_1 < ... < q_n$.
$cpu(h, q)$	Number of CPU cores required for transacting a segment from bitrate h to the desired bitrate q of the user.
$cpu(n)$	Number of CPU cores available on node $n \in N$.
$stg(n)$	Caching storage of node $n \in N$ in Megabytes.
$bwt(e)$	The bandwidth capacity of the substrate link $e \in E$.
τ	Segment time duration. All the segments are considered to be in the same duration.
α_m^r	A parameter to assign profit/utility to embedding of request r on MEC node m.

TABLE IV
UE REQUEST PARAMETERS

Parameters	Description
$\bar{G}\langle R, L \rangle$	Video request graph.
R	Set of user requests in $\bar{G}$.
Q^r	Set of possible bitrates for a video segment in $r \in R$ that can fulfill the request. It includes the exact requested bitrate q and bitrates higher than q.
$vid(r)$	The video inside request $r \in R$.
$seg(r)$	The segment inside request $r \in R$.
$br(r)$	The bitrate value of request $r \in R$.
L	Set of links in $\bar{G}$ representing the mapping of users to their requested bitrate.

TABLE V
BINARY DECISION VARIABLES

Variables	Description
$\chi_n^{v,s,h}$	Indicates if segment $s \in S^v$ of video $v \in V$ in bitrate $h \in Q^{v,s}$ has been mapped on edge node $m \in M$.
$\chi_n^{r,h}$	Indicates if request $r \in R$ with bitrate h has been mapped on node $n \in N$.
$\chi_e^{\bar{e}}$	Indicates if virtual link $\bar{e} \in L$ is mapped on substrate link $e \in E$.

nodes, which is the union of MEC nodes *M* and the 5GC *C*, $N = M \cup C$. *E* represents the set of backhaul and Xn links, interconnecting the gNBs/MEC with the 5GC and MEC nodes with each other, respectively. As already mentioned, MEC nodes are collocated with gNBs, and MEC nodes are characterized by storage *stg* (*n*) and processing capacity *cpu* (*n*). While storage resources are used to cache video segments, processing resources, if needed, is used to transcode video segments from a high bitrate *h* to a lower bitrate *q* (i.e., the one predicted to be requested by the user). There is a link $e^{m,n} \in E$ between the nodes $m, n \in N$ if they are directly connected, which has a certain amount of bandwidth denoted by *btw*(*e*). *V* represents the set of videos available to the users. Each video $v \in V$ is divided into multiple segments S^v, each of which $s \in S^v$ is available in multiple bitrates $Q^{v,s}$. Table III summarizes the parameters of the mobile network.

C. User Request Model

Fig. 4 depicts a sample video caching mapping in which a group of users are associated with gNBs and making requests for their desired video segment bitrate. User requests are modeled as a directed graph $\bar{G} = \langle R, L \rangle$, where *R* is the union of users and their requested bitrate for a specific video segment, and *L* represents the virtual links between users and their requested bitrate. It is possible to have multiple requests from the same user at any given time and *br*(*r*) represent the bitrate of a request from the user. Table IV summarizes the notations used for the user requests.

D. Problem Formulation

The joint video segment prefetching, transcoding, and resource allocation problem is similar to the Virtual Network Embedding problem, which is proven to be NP-hard [21]. Our proposed method solves the problem in two steps: (i) content embedding on the nodes and (ii) link embedding for accessing remote contents. In the content embedding step, each video content (e.g., UEs and video segments) in the request is mapped to a substrate node (e.g., gNBs and MEC nodes). In the link embedding instead, each virtual link can span multiple phsyical links to establish a path from the UE to the video segment. What comes below formally presents our proposed methods for tackling the joint problem of video content prefetching, transcoding, and resource allocation. First we detail our proposed ILP model for tackling this problem and later we present a heuristic algorithm to approximate the optimal solution obtained by the ILP model.

1) Proposed ILP Model: ILP technique is used to formulate the described content placement problem that has two objective functions. These two objective functions are different, and each follows a different logic, but they both have the same number of constraints that should be respected for the model to reach a valid solution. While the first objective (1) tends to maximize the cache hit ratio, the second (2) maximizes the byte hit ratio. Table V summarizes the variables used in the ILP model.

In both objective functions, we have the parameter α_m^r, which is responsible for assigning a profit/utility to a placement solution. The value of α is fixed at prior in the range of 1 to 5 and does not depend on the network condition. The value for each solution is selected based on the order described below. There are multiple ways to increase the cache hit ratio, and each of them has a certain profit/utility, and α_m^r is responsible for associating a profit/utility to each of the mapping options. The more distant a segment is from the user, the more unlikely it is to be used by that user. Therefore, α_m^r gets a higher value when the content is closer to the user.[1] When a user requests a video segment(s) with a specific bitrate, if the content is already cached/available at the edge, then

[1]The value of α_m^r can be tuned arbitrarily, for instance, can be derived from other metrics such as latency, cost of resources, etc.

it can be served either from the MEC node collocated with the associated gNB or from a neighbor MEC node leveraging the Xn interface. In such a case, α_m^r gets higher values of 5 and 4, respectively. Suppose the desired bitrate of the requested segment does not exist in any of the MEC nodes while higher bitrates of the same segment are available. In this case, these segments can be transcoded to the ultimate desired bitrate/quality. Similarly, here α_m^r gets a value, and it is our next preferred solution since we are still serving the user from the edge (for local transcoding $\alpha = 3$, and for transcoding on neighboring MEC nodes $\alpha = 2$). And finally, α_m^r gets the lowest value ($\alpha = 1$) when the content is served from the main video server since it is more distant to the user compared to other solutions and indirectly incurs more latency and link utilization.

$$\textbf{\textit{CacheHit}}: \quad max \left(\sum_{r \in R} \sum_{m \in M} \sum_{\substack{h \in Q^r \\ h \geq q}} \alpha_m^r \chi_m^{r,h} \right) \tag{1}$$

The second objective function (2) aims to maximize byte hit ratio. This objective is particularly beneficial for storing videos with high storage demand at the edge, and depending on the prediction performance it can result in backhaul utilization improvements. *br*(*r*) returns the bitrate value of a segment with a specific quality that the user requests. With this parameter, we aim to force the objective to prefer bulky-sized segments over small-sized ones when deciding on prefetching.

$$\textbf{\textit{ByteHit}}: \quad max \left(\sum_{r \in R} \sum_{m \in M} \sum_{\substack{h \in Q^r \\ h \geq q}} \alpha_m^r \chi_m^{r,h} br(r) \right) \tag{2}$$

In the following, we present the constraints that, regardless of the objective function, have to be satisfied in order for the model to obtain a valid solution. The first constraint ensures that the storage used for storing segments on a node is less than or equal to the maximum storage capacity available on that node.

$$\forall m \in M : \sum_{v \in V} \sum_{s \in S^v} \sum_{q \in Q^{v,s}} \chi_m^{v,s,q} br(q) \tau \leq stg(m) \tag{3}$$

At any given time, each users that makes a request, should be provided with the video segment in the requested bitrate.

$$\forall r \in R : \sum_{n \in N} \sum_{\substack{h \in Q^r \\ h \geq q}} \chi_n^{r,h} = 1 \tag{4}$$

Constraint (5) ensures that the virtual links (links between users and their requested video segment) are mapped on a substrate link as long as the link has sufficient capacity.

$$\forall e \in E : \sum_{\bar{e} \in L} \chi_e^{\bar{e}} br(\bar{e}) \leq bwt(e) \tag{5}$$

Constraint (6) enforces a continuous path to be established between each user and its requested bitrate of the video segment in the virtual request $r \in R$. This constraint makes sure that when a user cannot be served from its local MEC server but is served from a neighboring MEC server or the main video server, then there should be transport bandwidth assigned to it, and a route from the user to the selected remote serving node should be established.

$$\forall k \in N, \forall e^{m,n} \in L : \sum_{e \in E^{k\rightarrow}} \chi_e^{e^{m,n}} - \sum_{e \in E^{\rightarrow k}} \chi_e^{e^{m,n}} = \begin{cases} -1 & \text{if } k = m \\ 1 & \text{if } k = n \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $E^{k\rightarrow}$ represents the links originating from node $k \in N$, while $E^{\rightarrow k}$ represents all the links entering node $k \in N$.

Finally, constraint (7) makes sure that the number of CPU cores utilized for transcoding video segments in bitrate $h \in Q^r$ to the lower desired bitrate q are not higher than number of CPU cores available on that edge node. It is worth mentioning that the model decides to transcode a segment when caching space at the MEC server is not enough to store a new segment request. In such a case, it is possible to serve the request by converting a higher-quality existing segment to a lower-quality one on the same MEC server without needing to store it. For instance, consider two users (UE1 and UE2) requesting the same segment (same segment ID) but in different bitrates. UE1 asks for the segment in a higher bitrate, and UE2 asks for the same segment but in a lower bitrate. Also, in this scenario assume we do not have enough cache to store both of these segments, but we can afford to store one of them. In such a case, we can fetch the higher-bitrate segment to the edge and use the cache storage to store it and serve UE1 and then transcode the same segment to convert it to a lower bitrate and serve UE2 as well. With this strategy, we could meet both requests directly from the edge, one through the cache and the other via transcoding.

$$\forall m \in M : \sum_{r \in R} \sum_{\substack{h \in Q^r \\ h > q}} \chi_m^{r,h} * cpu(h,q) \leq cpu(m) \tag{7}$$

2) Proposed Heuristic Algorithm: Although our proposed ILP model achieves an optimal solution under different network configurations, the problem becomes computationally intractable as the network size grows, meaning we have more nodes, users, videos, and segments in the network. We address this problem by proposing a heuristic algorithm (see Algorithm 1) that is capable of attaining a near-optimal solution for the joint problem of video segment prefetching, transcoding, and resource allocation in a considerably shorter time scale compared to its ILP-based counterpart model, even for very complex network configurations.

Our proposed heuristic, called *Heu cache_hit*, pursues the same objective of maximizing cache hit ratio for the video segments. Although our presented algorithm only follows the objective of maximizing cache hit, it can easily be tuned to pursue the objective of maximizing byte hit by sorting requests based on bitrate in ascending order and then giving them as input to the algorithm. Similar to the ILP cache hit algorithm, *Heu cache_hit* uses the α_m^r parameter to assign profit/utility to embedding options before running the algorithm. After obtaining prediction results, we have a pre-processing procedure that calculates profit/utility value (α_m^r) for requests from

Algorithm 1: ***Heu cache_hit***

Input: $(G, \bar{G})$
Output: (video, segment, bitrate) prefetching, transcoding, and resource allocation ;

```
1  for r ∈ R do
2      q ← br(r) ;
3      m ← gnbAssociation(r) ;
4      • Computing profit/utility value α_m^r for request r;
5      • S ← Sorting requests based on the value of α_m^r in descending order and
         return tuple of the request and preferred node to list ;
6      for h, n ∈ S do
7          if alloc[r, h, n] == 1 then
8              if n ≠ m and btw(n, m) > br(r) then
9                  • Allocate path P_{n,m} and update resources;
10             • Allocate (h, n, r) and update resources;
11             break;
12         else
13             if stg(n) > br(r) ∗ τ then
14                 if h > q and cpu(h, q) < cpu(n) then
15                     if n ≠ m and btw(n, m) > br(r) then
16                         • Allocate path P_{n,m} and update resources;
17                     • Allocate (h, n, r) and update resources;
18                     cpu(n) ← cpu(n) − cpu(h, q) ;
19                     break;
20                 else
21                     if n ≠ m and btw(n, m) > br(r) then
22                         • Allocate path P_{n,m} and update resources;
23                     • Allocate (h, n, r) and update resources;
24                     break;
```

users. Based on the gNB association of a user, we calculate a value for α_m^r in the following way. The highest α_m^r value for a user is the MEC node directly connected to its associated gNB. The second-highest value is assigned to a solution that decides to transcode a high bitrate segment to a lower segment on that local MEC node. The idea behind prioritizing serving users from their local MEC node is that we can avoid consuming transport links and avoid unwanted transport latencies by serving a user from a local MEC node. The third-highest value will be assigned to the neighboring MEC nodes that host the required video segment. Finally, the smallest value will be assigned to the case a video segment is transcoded on the neighbor MEC nodes. It is worth mentioning that the policy for setting values for α_m^r is completely dependent on the service provider and can easily be tuned and adapted based on the desire.

What follows explains the details of the proposed heuristic. The algorithm considers each predicted request to be issued during the next time window, extracting its predicted bitrate *q* and the MEC node collocated with its associated gNB *g*. We remind the reader that video, segment, and bitrate are inherently included in the request *r*. The heuristic then calculates the profit/utility parameter α_m^r for each request based on the UE-gNB association and sorts them in descending order. This is followed by traversing all the bitrates and nodes as possible solutions in the sorted list of the profit/utility parameter α_m^r. The *alloc* is an array to store embedding decisions. If a candidate solution has already been embedded on the substrate network, indicated by *alloc*[*r*, *h*, *n*] == 1, then the UE will be served from the same video segment on the same node without prefetching a new segment. If the desired segment is on a node different from the hosting MEC, then a path will be established between the hosting MEC node and the node serving the desired segment if the physical links on the path have sufficient capacity.

In case a segment at the requested bitrate is missing on the hosting MEC node, the algorithm checks for computing capacities available on the node in order to perform transcoding operations. If transcoding is chosen as the solution to serve the user (line 14), the allocation variable *alloc* will be updated, and computing resources will be updated accordingly. Finally, if none of the already existing solutions are chosen for the request, the new segment with the new bitrate will be prefetched, and a path from the requesting user will be allocated in order to reach the segment (lines 22 and 23).

As we demonstrate later in Section VI, the execution time of the algorithm stays linear as the problem size grows. Overall, the time complexity of the algorithm is $T(c_1RNQ+c_2RNQ)$, where c_1 and c_2 are constants and negligible, R is the number of requests, N is the number of nodes in the network, and Q is the number of qualities/bitrates available for the video segment requested by the user. Therefore, the time complexity of the algorithm is of order *O*(*RNQ*). Note that, for each request, the video ID and segment ID are known beforehand, and we do not need to loop over different videos and segments.

IV. Data Generation and Processing

A. Simulation Setup for Data Generation

An ns-3 [13] simulated network is used to generate data to train and validate the prediction models, and to evaluate the prefetching algorithms. We generated two datasets (available at [19]) from a simulated urban mobile network deployment scenario similar to the scenario described in the 3GPP report [22]. We use the ns-3 DASH module implemented by Vergados *et al.* [23] to simulate the DASH client-server interaction for video segment requests and response. The ABR algorithm implemented on the DASH clients (FDASH [24]) uses both network bandwidth and buffer occupancy information to select the bitrate of the segments requested. The set of available bitrates are {1, 2.5, 5, 8, 16, 35} Mbps. Each DASH client implements a video buffer of 30 MB. It is important to emphasize here that our approach can handle any ABR at the client, as long as it can be trained on that data.

Our simulation setup consists of LTE base stations and UEs since at the time of generating this dataset there were limitations in ns3-5G modules. However, our work is fully applicable to 5G and beyond networks. The simulation setup consists of 12 base stations (four cell sites with three gNBs each), and uses carrier aggregation with three component carriers of 20 Mhz (100 physical resource blocks) to provide a maximum downlink bandwidth of 60 Mhz (300 physical resource blocks), which can reach a maximum downlink data bandwidth of up to 225 Mbps. The actual data rate that a user receives from the gNB is dynamic and varies as the signal quality varies as it moves within and between cells. It also varies depending on the number of users the channel is shared with and their request patterns. A remote server hosts the DASH video server. Mobile users, between 27 to 68, depending on the simulation instance, move between these gNBs.

The user movement is dictated by the random waypoint mobility model with velocities samples from a uniform distribution between 1.4 - 5.0 m/s (walking/cycling speeds).

Each user in the network is watching one of the 10 live videos being streamed. These videos are generated by the ns-3 DASH component by inserting random bits to create segments of the defined duration and at defined bitrates. The start time of requests for each user are staggered to be within a short time range of 10 seconds. This is done to create the random time differences between the beginning of segment requests by various clients.

The frame rate is 50 frames per second and the segment duration is 8 seconds. This choice of segment size was informed by the range used in current implementations which range from 2 seconds (for e.g., in Microsoft Smooth Streaming) to 10 seconds (in Apple HTTP Streaming). While shorter segments give lower latency, longer segments give better throughput.

We generated over 26,000 seconds (7.2 hours) of data from the simulation. On separating video playback data per client (between 27 and 68) we have 1.2 million seconds (14 days) of video streaming data. On pre-processing the data as described in Section IV-B into windows, we get 117,000 samples. We use 80,000 samples to train the ML models, and 20,000 samples to validate the trained models. On obtaining a validated ML model for expected segment bitrate prediction, we evaluate the entire approach of predicting, prefetching and caching. We evaluate this by using 17,000 test samples, unseen during training or validation. These test or evaluation samples are used to make bitrate predictions and evaluate the approach by comparing the decision of the prefetching and caching, with the ground truth of what segment gets requested after the prediction and prefetching is made. Our evaluation was conducted in an offline manner to circumvent the lack of implementation of the needed architectural components in ns-3.

While the size of the topology used in our simulation is small, it is scaled down to create a dataset that is still representative of the problem for which we propose a solution. Through the randomized deployment of users in the topology, varying number of clients per basestation, and user mobility, we create network states where the DASH clients are exposed to varying degrees of saturated and unsaturated network states as well as signal qualities as indicated by the histogram of segment bitrates achieved by the clients as shown in Fig. 3. By creating a representative scenario with a smaller number of users, we are able to simulate a much larger dataset that captures data from a longer duration of simulation time.

B. Data Preprocessing

All the cross-layer monitored metrics we consider as input to the ML model are time series quantities observed with different periodicities. Some are sample metrics (like RSRP), wherein we have a value for every observation, while others (like MAC throughput) can only be measured over windows. To reconcile these differences in how metrics are observed, we aggregate them over time windows (W_{aggr}, the metrics aggregation window) to generate a structured tabular dataset. Since the data is a time series, there is time correlation between samples of input metrics. Since we are predicting the bitrate of a segment of 8 seconds of video, small time correlations do not impact the output much and can be averaged over, as we have done using W_{aggr}. The choice of W_{aggr} size should be informed by the rate of change of state in the radio network. The trade-off here is between too little information in small windows and too stale information in large windows. The motivation for our choice of W_{pred} has been discussed in Section II-A.

TABLE VI
PREDICTION ACCURACY USING RF OVER VARYING AGGREGATION AND PREDICTION WINDOW SIZES

W_{pred} (s) \ W_{aggr} (s)	4	8	12	**16**	20	24
2	0.864	0.886	0.898	**0.901**	0.895	0.887
4	0.843	0.875	0.890	0.891	0.887	0.879
6	0.799	0.862	0.876	0.882	0.886	0.876

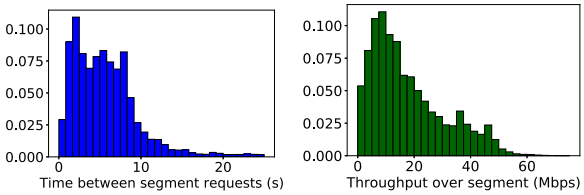

Fig. 5. Histogram of the time between segment requests and the throughput achieved during the download of a segment at the clients.

The results of our empirical evaluation of varying window sizes is presented in Table VI. We compare the sizes using the prediction accuracy obtained using a random forest model and chose W_{aggr} to be 16 seconds and W_{pred} to be 2 seconds. We obtain the similar order of results for all models we explore and have hence presented the results from one model with the results available in an appendix on our GitHub [19].

The scripts for data generation, the raw and pre-processed data, the ML and prefetcher scripts are all made open source for repeatability [19].

V. EVALUATION: VIDEO SEGMENT PREDICTION

A. Exploratory Data Analysis

Understanding the variability in our dataset: We begin by presenting some statistics that represent the dataset we have generated. The average time between segment bitrate changes in this dataset is around 23 seconds, and the fraction of segments that represent a change in bitrate is 34%. These numbers indicate that there is a significant rate of change of bitrate in the dynamic environment created to justify the evaluation of our prediction approach. Fig. 5 contains the histogram of the time between segment requests at the DASH clients. By setting the prediction window size at 2 seconds we capture most of the probability mass from this distribution which means that there will be very few instances where a client requests for more than one segment that has been prefetched withing the

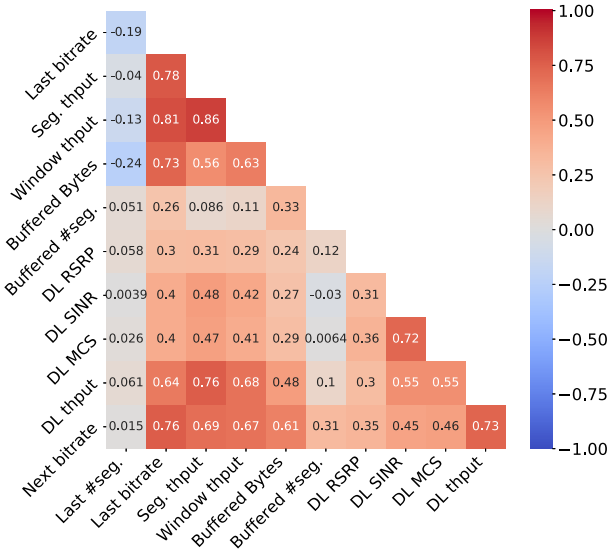

Fig. 6. Correlation coefficient heatmap between input and output features.

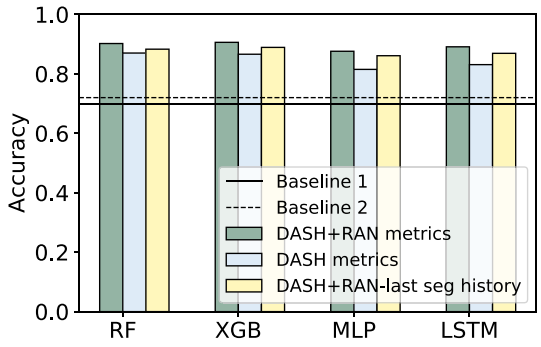

Fig. 7. Comparison of model performance for the three bitrate prediction tasks. Prediction accuracy increase is around 20% from both baselines shown as black solid and dashed lines.

prediction window time. Fig. 5 also contains the histogram of the throughput achieved during the download of a segment. This histogram shows the variation in the channel quality as experienced by the clients due to mobility and contention between users. We see that the probability mass is aggregated around the 10 Mbps value, and is distributed between 0 to over 60 Mbps showing that the available download datarate varies in a significant range and can support the use of the maximum segment bitrate in our scenario of 35 Mbps. The statistics and histograms discussed here, show the extent of network variability in our scenario and justify the choice of W_{pred} window size.

Understanding the correlations in our dataset: To understand the relationship between input features, as well as between the input and output features, we plot a heatmap of the Kendall correlation coefficients between them as shown in Fig. 6. The bottom row, next bitrate, represents the output feature, and the rest are input features. This Kendall score captures the strength of the monotonic relationship between variables. It is a rank-based metric and preferred over Spearman because it is better at handling rank clashes. We see several strong correlations between the input and output features, indicating a signal for predictive power. We also see strong correlations between input features themselves, indicating some co-variation among variables and hence redundancy in the features.

B. Model Evaluation

The input features available in our dataset can be categorized as RAN metrics and DASH application metrics. We study the impact of each feature set by evaluating the behaviour of the ML model on three different sets containing 1) DASH and RAN metrics, 2) only DASH metrics, 3) both DASH and RAN metrics but no historical information of last bitrate. Feature set 1) includes all features observed, feature set 2) removes RAN features to see how much the addition of RAN observability adds to the prediction signal, and the feature set 3) is used to assess the reliance of the model on information from the last sample of bitrate.

Fig. 7 shows the results of evaluation over RF, XGB, MLP, and LSTM models for each of the three different feature sets. We use prediction accuracy as the metric for comparison between models and the baselines. Accuracy metric is defined as (true positives + true negatives) / total predictions. This was chosen to compare models since, from the perspective of the prediction model, the cost of misclassification between classes is considered the same in the considered scenario. We see that our models improve over both the simpler Baseline 1 as well as Baseline 2 from recent work, with a prediction accuracy increase of around 20%. The network scenario and dataset considered in the work of Baseline 2 has low network dynamics with a rate of change of segment bitrate of less than 10%. In comparison our network scenario is more dynamic with segment bitrate changes at 34% of the total stream. This results in our model being a stronger predictor in more dynamic networks. We also see that all the models considered have similar performance with a difference of at most 3% in prediction accuracy.

The minor difference in accuracy when using DASH+RAN features and only DASH features, indicates that the contribution of RAN features over DASH features is small. Comparison between DASH+RAN and DASH+RAN-last segment history, also indicates that there is redundancy in the features that can contribute to prediction, even when the last segment value for bitrate is not available. This is important because it shows that the model can learn to predict bitrate in a non-trivial way using the network and application state, even when the new bitrate is not correlated with the previous bitrate. This improvement over the baselines is expected to be even higher in more dynamic networks, where the rate of change of network state is even higher, resulting in a lower correlation between requested bitrates.

Fig. 8 shows a plot of the normalized confusion matrix for RF and MLP as a heatmap. The maximum difference between the true positives between bitrate classes is at most 21% for RF and 14% for MLP. This shows that we are not much better at predicting one class compared to others, and that the prediction power is fairly distributed over classes. It also indicates that the

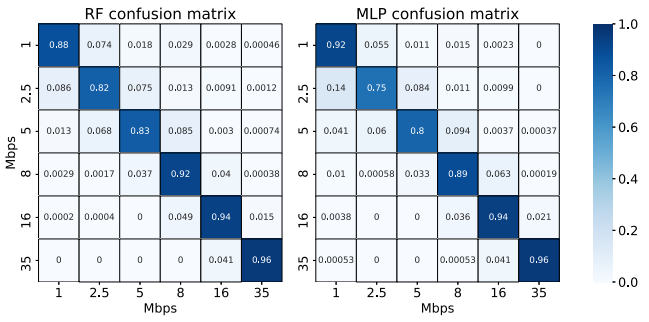

Fig. 8. Ground truth normalized confusion matrix for RF and MLP predictions.

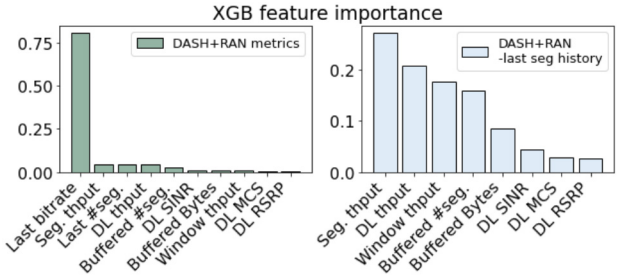

Fig. 9. Feature importance for different sets of metrics.

minor imbalance in the classes has not affected the prediction power of the classes. The type-1 and type-2 errors are both small and symmetric between classes. This would be a positive trait of the predictions if type-1 and type-2 errors have different costs, i.e., our approach handles the misclassifying as a lower bitrate as different from the misclassification as a higher bitrate. This is a positive aspect to note if in future work these misclassifications are handled differently.

One of the advantages of RF or XGB is their inherent model interpretability through observable feature importance. Fig. 9 shows the feature importance graphs for XGB for DASH+RAN, and the DASH+RAN-last segment history feature sets. We omit the graph for RF since the order of features remains the same and XGB feature importance shows the additional contribution of each feature over the previous, as opposed to RF, where multiple redundant features are shown to have the same importance. When *Last bitrate* is available, it seems to be the most important feature. This makes sense since the persistent prediction baseline (Baseline 1) by itself gives us close to 70% accuracy. However, with that history removed, we still get similar accuracy but the contribution of the features as indicated by their importance is distributed over all other existing features. This shows that even when the rate of change of bitrate is high, the model can predict without the assistance of *Last bitrate*.

C. Discussion

While all four models have similar performance, with a prediction accuracy increase of around 20% over both the baselines, employing different feature sets taught us that the models are able to learn to predict changes in bitrate, without the history of previous bitrate from the last window. The decision tree-based RF and XGB have the advantage of interpretability, while the neural network models have the advantage of transfer learning and generalizability. We see that using LSTM did not offer any advantages in this scenario over a much simpler MLP. This could be because of the much larger number of trainable parameters in an LSTM model, which requires a larger dataset to train on.

It is important to note that while the implementation of the number of bitrates over which the DASH server can encode a given segment is fixed in our evaluations, this could be varied. The machine learning model could be readily adapted in such a scenario by retraining on this different number of bitrate classes while keeping the rest of the approach, leaving the prefetching algorithm unchanged.

VI. Evaluation: Segment Request Prefetching

This section elaborates on the simulations carried out in Python using the Gurobi mathematical optimization solver [25]. The evaluations regarding the ILP-based model and the heuristic algorithm consider the following additional evaluation parameters. The 5GC node, where the DASH video server runs, is equipped with a 16-core 2.4 GHz processor. The edge MEC servers are equipped with a 4-core 1.6 GHz processor with varying cache storage of {25, 50, 75, 100, 125, 150} MB. The caching storage is assumed to be very limited since resources at the edge are extremely scarce and shared among a diverse set of applications that are running at the edge. Moreover, computing resources at the edge are usually reserved for applications with stringent demands in terms of latency, leading us to assume that we are pretty limited in terms of caching space at the edge dedicated for live video streaming. The edge MEC nodes are interconnected over 5 Gbps Xn links, which are also used to transfer video segment data along with exchanging control information. The MEC nodes are connected to the 5GC over a 20 Gbps backhaul link.

The prefetching algorithm (ILP or heuristic) runs periodically every 4 seconds, in what we call a prefetching time slot. In each prefetching time slot, the Decision Maker submodule (described in the system architecture) at the 5GC obtains the segment/bitrate predictions for all the UEs from the Predictor component at the MEC, which then assembles all prefetching requests. A request encompasses four items: video ID, segment ID, bitrate, and gNB association of the UE. The job of the prefetching algorithm is to find solutions that can serve predicted requests. The ILP and the faster heuristic still take a finite amount of time to run. However, the prefetching time slot is always longer than the time to run the algorithm to ensure that the requests from one slot are fetched before the beginning of the next slot.

The predicted requests can be served by either prefetching the requested segments to the edge, transcoding an existing segment at the edge, or skipping the cache and waiting for the video server in the 5GC to serve the actual request. There are two cases in which a user will be served from the central video server in the 5GC: (i) the *prefetching algorithm decides* that the user is to be served from the main video server due to the optimization decisions (i.e., lack of storage resources at

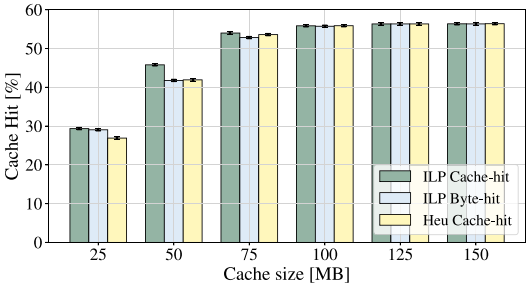

Fig. 10. Cache hit ratio for different cache sizes.

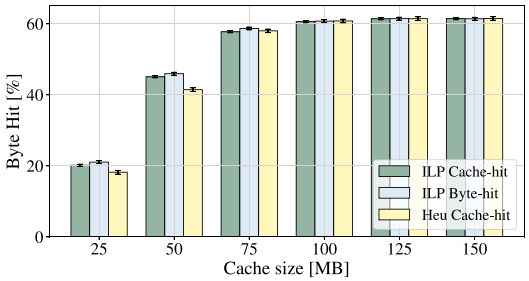

Fig. 11. Byte hit ratio for different cache sizes.

the edge), (ii) due to *prediction error*, a wrong segment bitrate is prefetched and the requested bitrate is not at the edge, then the user again should be redirected to the main video server in the 5GC.

The caching system at the MEC node uses a simple Least Recently Used (LRU) replacement strategy for content management. The Cache Manager informs the Request Handler about the cache status and the requests that can be served from the edge, and it frees up cache space if needed by discarding any other segments based on the LRU algorithm.

The reported results are the average of four simulation experiments with 95% confidence intervals, where each simulation runs for 1000 seconds. In other words, if we consider the duration of 4 seconds for each segment, then with each experiment, we repeat the tests 250 times (1000 times overall). A varying number of requests are issued during each simulation run, and the prefetching algorithm is responsible for making an intelligent decision to embed the requests on the network.

Cache hit ratio: As stated earlier, a request can be served from the edge in multiple ways. The first solution is to serve the request directly from the MEC node collocated with the gNB that the UE is associated with, and it can be served either by the exact bitrate stored at the MEC node or by transcoding a higher bitrate segment to the target bitrate. The second solution is to serve the request through the neighbor MEC nodes with the same methods. We define the cache hit ratio as the ratio between the number of users served from the edge to the overall number of users. We argue that the higher cache hit results in higher satisfaction for both the end-users and the MNO. While users experience less delay, less jitter, and higher bitrate, MNOs can offload backhaul to a large extent.

Here we study the cache hit ratio under different cache sizes. It can be observed that increasing cache size results in a higher cache hit ratio. Therefore, we aim to study the impact of the cache hit ratio in different network configurations. It is worth mentioning that we decrease the cache size to a large extent to show the system's different behaviors for testing purposes.

As shown in Fig. 10, the average cache hit ratio with different cache sizes is higher for the *ILP Cache-hit* algorithm compared to the other two algorithms. As expected, the *ILP Cache-hit* achieves the highest cache hit ratio due to the significance of the number of hits in the objective function. The proposed heuristic algorithm also demonstrates a satisfactory level of performance and attains a cache hit ratio close to the ILP counterpart. The superior performance of the *ILP Cache-hit* becomes obvious when the cache storage is very limited and all the predicted segments cannot be accommodated at the MEC nodes. Therefore, this is a scenario where *ILP Cache-hit* can make more intelligent decisions than *ILP Byte hit* and *Heu Cache-hit* methods in selecting a set of segments and bitrates to be prefetched, leading to the maximum cache hit. Although increasing the cache size may lead to a scenario in which we can accommodate all the predicted segments at the edge, making prefetching decisions straightforward for all the methods to prefetch whatever is predicted and achieve a high cache hit ratio. Even in such a scenario, we experience some cache misses due to the wrong predictions from the ML-based prediction model.

Byte hit ratio: Similar to the cache hit ratio, we also study the byte hit ratio for the proposed methods. Obviously, with the more bytes served from the edge, the more savings over the backhaul link can be achieved. With this performance metric we intended to depict the ratio between the number of bytes served from the edge and the overall number of bytes requested. Therefore, we expect the *ILP Byte-hit* to perform better than the other *ILP Cache-hit* and *Heu Cache-hit* in prefetching higher bitrate segments that can be shared among multiple UEs at the edge to save the bandwidth. Another advantage of prefetching high bitrate segments is that a higher bitrate segment can be transcoded to the lower bitrate segments and avoid redirecting requests for low bitrate segments to the core and serve more users from the edge.

As illustrated in Fig. 11, the number of bytes served from the edge for the *ILP Byte-hit algorithm* is more than both *ILP Byte-hit* and *Heu Cache-hit* algorithm. We can observe that the *ILP Byte-hit* method shows a better performance than the other algorithms when the cache size is smaller since it tends to prefetch high bitrate segments compared with other methods that do not consider the size of segments as a factor in the objective function. As the figure depicts, the *ILP Cache-hit* shows a relative performance to *ILP Byte-hit* because it can prefetch more segments, increasing the number of served bytes but still being less than the ones achieved by *ILP Byte-hit*. As expected, *heu Cache-hit* also achieves a comparable performance to what is achieved by *ILP Cache-hit*. Even though *ILP Byte-hit* objective achieved a higher byte hit, in case the prediction results are not correct, its performance significantly degrades as it soon saturates storage at the edge with bulky-size wrong segments and does not leave space for other segments to be prefetched.

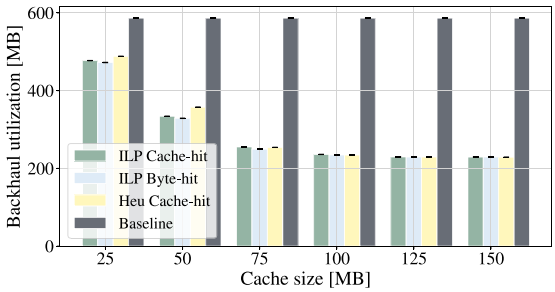

Fig. 12. Backhaul link utilization for different cache sizes.

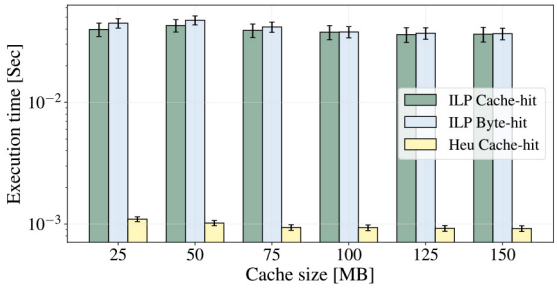

Fig. 13. Execution time for different cache sizes.

Link utilization: Fig. 12 illustrates the backhaul link utilization as a function of cache size, averaged over four simulation experiments, where each of them runs for 1000 seconds. We have evaluated our proposed algorithms and compared them with the baseline in which all the segments are served from the main video server in the 5GC. Therefore, no prefetching is performed with the baseline, meaning all video segments are served directly from the 5GC and transferred through the backhaul. As can be observed from the plot, *Heu Cache-hit* and *ILP Byte-hit* achieve, respectively, the highest and the lowest backhaul link utilization compared with the baseline. This is justified by the fact that *ILP Byte-hit* tends to prefetch high bitrate segments that might be shared among multiple requests at the edge. Therefore, a smaller portion of high bitrate requests must be directed to the main video server; consequently, a significant load on the backhaul is mitigated. It is worth noting that even though for large cache sizes (i.e., 150 MB) on each edge node, we have enough aggregated capacity to host all the requested video segments at the edge, we still have around 20% of backhaul utilization. Such behavior stems from the fact that we have a few wrongly predicted requests in each run; therefore, our proposed system at the edge redirects such cases to the main video server in 5GC, leading to backhaul consumption. Fig. 12 reveals that our proposed ILP-based methods save up to 60.91% of the backhaul compared to the baseline. Although these results are obtained under large cache sizes, they still perform satisfactorily even with very limited cache sizes (see the cases of 25 to 100 MB). Regarding the heuristic algorithm *Heu Cache-hit*, which follows the same objective as the *ILP Cache-hit*, it shows a slightly lower performance but quite close and comparable to its ILP-based counterpart.

Execution time: The main intention behind proposing our heuristic algorithm (*Heu Cache-hit*) is to combat the scalability issue of the *ILP Cache-hit* model, which becomes computationally intractable as the network size increases, meaning the network includes more videos, more segments, and more unique requests from the users. Fig. 13 demonstrates the execution time of the proposed algorithms averaged over four experiments. As expected, our execution time remains constant both for the ILP models and the heuristic. The reason is that our model aggregates individual users by their access gNB, so the number of decision variables does not grow. In other words, our model considers users that come from the same gNB and ask for the same segment on the same bitrate as one request. We claim that with the increase in the number of nodes, videos, and bitrates, the problem becomes intractable for the ILP models, and they cannot scale properly; thus, the heuristic algorithm *Heu Cache-hit* is proposed to address this challenge. We observed in the previous results that the *Heu Cache-hit* exhibits lower performance compared to its ILP counterpart in terms of the cache hit ratio, byte hit ratio, and link utilization, but it proves to be competitive and also much quicker in terms of execution time, which makes it applicable to large-size networks in real-world scenarios.

VII. Related Work and Contribution

A. ML Prediction

Prediction for RANs is especially challenging due to the continually changing conditions of the physical channel, and the availability of different radio access technologies [26]. Employing ML to predict specific metrics (e.g., channel throughput) for RAN has gained importance [16], [26], [27], [28]. Within the context of DASH, previous work employing ML has most of its focus on bandwidth estimation at the client, which constitutes an input to most ABR algorithms. Raca *et al.* [29] demonstrate that integrating throughput prediction in the client can increase QoE, regardless of the employed ABR algorithm. This idea is then further explored [15], as the authors used an RF algorithm at the client to predict the expected average throughput over a time horizon. Moreover, Mao *et al.* [30] developed a reinforcement learning method for directly obtaining the bitrate for the next video chunk. The model employs an Actor-Critic neural network model at the client, whose input includes historical throughput information, buffer state, and next chunk sizes. An even more general reinforcement learning model that was not improved by Actor-Critic modeling has been developed and tested on real networks for RAN self-management [31]. Liang *et al.* [32] demonstrate and motivate the benefits of predictive prefetching. They consider streaming over a wired network wherein the rate of change of segment bitrate is very low, justifying their assumption that the next bitrate requested can be the same as the previous bitrate requested, i.e., our baseline.

Our approach performs the predictions at the MEC server, and its main aim is to assist the overall caching process at the core. To this end, it employs an end-to-end ML solution that uses the RAN metrics and past history of segments served to a client. These are in turn used to predict the number of

segments requested over a time horizon and the mode of these segments' qualities per client.

B. Prefetching and Caching

Li *et al.* consider a mobile network, enabled with RNIS information from the MEC servers for service improvement [33]. An adaption algorithm runs on the MEC nodes, responsible for alleviating network congestion and improving the user's QoE. Another study considers a heterogeneous network in which each client can switch between different wireless networks. A MEC application is introduced, capable of estimating the bandwidth and updating the client about the network condition [34]. Tan *et al.* propose a MEC-enabled solution capable of utilizing RNIS to perform caching and updating the cache for DASH video services [35]. They propose two kinds of popularity for caching videos—request popularity and expected popularity—to improve video quality and reduce buffer time. Another MEC-based DASH video caching strategy has been presented in which the proposed algorithm stores the highest bitrate of each segment on the edge nodes and employs the processing power accessible at the MEC nodes to transcode the video segment on demand [36]. A cache prefetching scheme able to prefetch video segments using an adaptation algorithm that takes into account the throughput measurements from the client and the predicted throughput at the cache has also been suggested [37]. Distributed and cooperative caching methods have also been studied, also for cloud RANs [38], and have been shown to have attractive worst-case time complexity when approximated [39]. Finally, a learning-based caching and prefetching method is devised by Shi *et al.* [2] to improve users' QoE for adaptive video streaming. The algorithm caches the most popular video segments at the edge in order to mitigate the problem of network jitter.

Our work stands out in that we study the trade-off between prefetching and transcoding higher bitrate video segments. Moreover, we consider the cooperation among the MEC nodes, making it possible to deliver the requested video segment hosted on adjacent MECs. Furthermore, we study a realistic scenario in which users can request various segment bitrates at any given time, independent from previous requests. Finally, unlike the studies cited above, we jointly study the prefetching, transcoding, and resource allocation problems in the scenario of mobile networks. This makes the problem more complex and realistic, due to the dynamicity of the network and the mobility of the users.

VIII. Conclusion

We proposed a novel approach for predictive prefetching of segments in a DASH video streaming application using neural networks and classical ML methods using services in MEC-enabled mobile networks. We demonstrated that with an accuracy of 90% for the predictive task, we achieved a MEC cache hit ratio of 58%, which means that, through predictive prefetching, we were able to reduce the access delay for 58% of the requests. The prediction algorithm predicts the segment request bitrate over 2 seconds prediction time windows using metrics from a 16 seconds metrics aggregation window. An ILP model with two objectives is proposed to reach an optimal solution for video content prefetching and transcoding at the edge, followed by a heuristic algorithm that achieves a near-optimal solution in a considerably shorter time scale. We demonstrated that the backhaul link utilization could be reduced by 60.91% through caching at the edge using *ILP Byte-hit* objective in a live streaming scenario with segment request overlaps. Our proposed heuristic resulted in reduction of the execution time for prefetching in the 25 MB cache size scenario by 90% with a reduction in the cache hit ratio by 7.5% and an increase of 2% in the backhaul link utilization compared to the optimal solution.

References

[1] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A survey on bitrate adaptation schemes for streaming media over HTTP," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 562–585, 1st Quart., 2019.

[2] W. Shi *et al.*, "LEAP: Learning-based smart edge with caching and prefetching for adaptive video streaming," in *Proc. ACM IWQOS*, Phoenix, AZ, USA, 2019, pp. 1–10.

[3] *Information Technology—Dynamic Adaptive Streaming Over HTTP (DASH)—Part 1: Media Presentation Description and Segment Formats*, ISO/IEC 23009-1:2019, 2012.

[4] T. X. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 1965–1978, Sep. 2019.

[5] L. Tang, Q. Huang, A. Puntambekar, Y. Vigfusson, W. Lloyd, and K. Li, "Popularity prediction of Facebook videos for higher quality streaming," in *Proc. USENIX ATC*, 2017 pp. 111–123.

[6] D. Huang, X. Tao, C. Jiang, S. Cui, and J. Lu, "Trace-driven QoE-aware proactive caching for mobile video streaming in metropolis," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 62–76, Jan. 2020.

[7] Y. Guan, X. Zhang, and Z. Guo, "PrefCache: Edge cache admission with user preference learning for video content distribution," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 4, pp. 1618–1631, Apr. 2021.

[8] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.

[9] "Mobile edge computing (MEC); Technical requirements," ETSI, Sophia Antipolis, France, ETSI document GS MEC 002 V1.1.1, 2016.

[10] "Mobile edge computing (MEC); mobile edge management," ETSI, Sophia Antipolis, France, ETSI document GS MEC 010-1 V1.1.1, 2017.

[11] Q. Li, C. Lu, B. Cao, and Q. Zhang, "Caching resource management of mobile edge network based on Stackelberg game," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 18–23, 2019.

[12] "Mobile edge computing (MEC); radio network information API," ETSI, Sophia Antipolis, France, ETSI document GR MEC 012 V1.1.1, 2017.

[13] "Network Simulator ns-3." 2022. [Online]. Available: https://www.nsnam.org/

[14] R. Viola, A. Martin, M. Zorrilla, and J. Montalbán, "MEC proxy for efficient cache and reliable multi-CDN video distribution," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, 2018, pp. 1–7.

[15] D. Raca *et al.*, "Empowering video players in cellular: Throughput prediction from radio network measurements," in *Proc. ACM MM*, New York, NY, USA, 2019, pp. 201–212.

[16] C. Yue, R. Jin, K. Suh, Y. Qin, B. Wang, and W. Wei, "LinkForecast: Cellular link bandwidth prediction in LTE networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 7, pp. 1582–1594, Jul. 2018.

[17] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD*, New York, NY, USA, 2016, pp. 785–794.

[18] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proc. ACM ICMLC*, New York, NY, USA, 2006, pp. 161–168.

[19] "Predictions-for-Edge-Enabled-Dash." [Online]. Available: https://github.com/akhila-s-rao/predictions-for-edge-enabled-dash (Accessed: May 15, 2022).

[20] B. Kheibari and M. Sayıt, "Quality estimation for DASH clients by using deep recurrent neural networks," in *Proc. 16th Int. Conf. Netw. Serv. Manag. (CNSM)*, 2020, pp. 1–8.
[21] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart., 2013.
[22] "LTE; evolved universal terrestrial radio access (E-UTRA); FDD home eNode B (HeNB) radio frequency (RF) requirements analysis," ETSI, Sophia Antipolis, France, ETSI Rep. TR 136 921 V9.0.0, 2010. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/136900_136999/136921/09.00.00_60/tr
[23] "An MPEG/DASH Client-Server ns3 Module." [Online]. Available: https://github.com/djvergad/dash (Accessed: Jul. 23, 2020).
[24] D. J. Vergados, A. Michalas, A. Sgora, D. D. Vergados, and P. Chatzimisios, "FDASH: A fuzzy-based MPEG/DASH adaptation algorithm," *IEEE Syst. J.*, vol. 10, no. 2, pp. 859–868, Jun. 2016.
[25] "Gurobi Mathematical Optimization Solver." [Online]. Available: https://www.gurobi.com/ (Accessed: Jul. 5, 2020).
[26] A. Samba, Y. Busnel, A. Blanc, P. Dooze, and G. Simon, "Instantaneous throughput prediction in cellular networks: Which information is needed?" in *Proc. IFIP/IEEE IM*, Lisbon, Portugal, 2017, pp. 624–627.
[27] M. Karimzadeh *et al.*, "Mobility and bandwidth prediction as a service in virtualized LTE systems," in *Proc. IEEE CloudNet*, Niagara Falls, ON, Canada, 2015, pp. 132–138.
[28] D. Raca *et al.*, "Back to the future: Throughput prediction for cellular networks using radio KPIs," in *Proc. ACM MOBICOM*, New York, NY, USA, 2017, pp. 37–41.
[29] D. Raca *et al.*, "Incorporating prediction into adaptive streaming algorithms: A QoE perspective," in *Proc. ACM SIGCOM NOSSDAV*, New York, NY, USA, 2018, pp. 49–54.
[30] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with Pensieve," in *Proc. ACM SIGCOM COMM*, New York, NY, USA, 2017, pp. 197–210.
[31] D. Corcoran, P. Kreuger, and M. Boman, "Reinforcement learning for automated energy efficient mobile network performance tuning," in *Proc. IEEE CNSM*, 2021, pp. 216–224.
[32] K. Liang, J. Hao, R. Zimmermann, and D. K. Yau, "Integrated prefetching and caching for adaptive video streaming over HTTP: An online approach," in *Proc. ACM MM*, Portland, OR, USA, 2015, pp. 142–152.
[33] Y. Li, P. A. Frangoudis, Y. Hadjadj-Aoul, and P. Bertin, "A mobile edge computing-based architecture for improved adaptive HTTP video delivery," in *Proc. IEEE CSCN*, Berlin, Germany, 2016, pp. 1–6.
[34] Y. Li, P. A. Frangoudis, Y. Hadjadj-Aoul, and P. Bertin, "A mobile edge computing-assisted video delivery architecture for wireless heterogeneous networks," in *Proc. IEEE ISCC*, Heraklion, Greece, 2017, pp. 534–539.
[35] Y. Tan, C. Han, M. Luo, X. Zhou, and X. Zhang, "Radio network-aware edge caching for video delivery in MEC-enabled cellular networks," in *Proc. IEEE WCNCW*, Barcelona, Spain, 2018, pp. 179–184.
[36] S. Kumar and D. S. Vineeth, "Edge assisted DASH video caching mechanism for multi-access edge computing," in *Proc. IEEE ANTS*, Indore, India, 2018, pp. 1–6.
[37] S. K. Mehr, P. Juluri, M. Maddumala, and D. Medhi, "An adaptation aware hybrid client-cache approach for video delivery with dynamic adaptive streaming over HTTP," in *Proc. IEEE/IFIP NOMS*, Taipei, Taiwan, 2018, pp. 1–5.
[38] T. X. Tran, D. V. Le, G. Yue, and D. Pompili, "Cooperative hierarchical caching and request scheduling in a cloud radio access network," *IEEE Trans. Mobile Comput.*, vol. 17, no. 12, pp. 2729–2743, Dec. 2018.
[39] K. Poularakis, G. Iosifidis, A. Argyriou, I. Koutsopoulos, and L. Tassiulas, "Caching and operator cooperation policies for layered video content delivery," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

Rasoul Behravesh received the B.Sc. degree in information technology from the Payam-e-Noor University of Mahabad, the M.Sc. degree in computer networks from QIAU, Iran, and the Ph.D. degree in telecommunications from the University of Bologna, Italy. He is a Researcher with the Smart Networks and Services Unit, Fondazione Bruno Kessler, Trento, Italy. He is currently working on service management and orchestration in 5G networks. His main research interests include 5G networks, multi-access edge computing, network function virtualization, and network slicing.

Akhila Rao received the bachelor's degree in electronics and communications from VTU, India, and the master's degree in wireless systems from KTH, Sweden. She is a Researcher with the Research Institutes of Sweden, Sweden. Her current research interest is in applying artificial intelligence methods for network automation. More specifically, using machine learning for performance prediction in wireless networks and network resource management over heterogeneous infrastructures. She has prior research experience working with information centric networking and IoT networks.

Daniel F. Perez-Ramirez received the B.Sc. degree in mechanical engineering and the M.Sc. degree in robotics, Cognition, Intelligence with emphasis on cognitive systems and machine learning from the Technical University of Munich, Germany. He joined the Research Institutes of Sweden AB (RISE), Stockholm, in 2019, where he currently researches machine learning to solving combinatorial optimization problems. Previous to RISE, he worked on applied machine learning for the automotive industry, knowledge representation, and sensor integration for robotics applications and agile methods for mechatronic product development. His research interests spans cognitive and intelligent autonomous systems.

Davit Harutyunyan received the bachelor's and master's degrees (Hons.) in telecommunication engineering from the National Polytechnic University of Armenia in 2011 and 2015, respectively, and the Ph.D. degree (Hons.) in information and communication technology from the University of Trento in 2019. He was an Expert Researcher in the SENSE research unit with FBK. He is Currently a Research Engineer in industrial 5G with the Corporate Research Sector, Robert Bosch. He has published more than ten papers in internationally recognized journals/conferences. His main research interests include non-public mobile networks, next-generation radio access networks, multi-access edge computing, and network slicing. He was the recipient of the Best Student Paper Award of IEEE CNSM 2017 and IEEE NetSoft 2019.

Roberto Riggio (Senior Member, IEEE) received the Ph.D. degree from the University of Trento, Italy. He was a Postdoctoral Fellow with the University of Florida, a Researcher/Chief Scientist with CREATE-NET, Trento, Italy, the Head of Unit with FBK, Trento, a Senior 5G Researcher with the i2CAT Foundation, Barcelona, Spain, and a Senior Researcher with RISE AB, Stockholm, Sweden. He is an Assistant Professor with the Polytechnic University of Marche in Ancona, Italy. He has published more than 130 papers in internationally refereed journals and conferences. His research interests revolve around optimization and algorithmic problems in networked and distributed systems. His current fields of applications are edge automation platforms, intelligent networks, and demand-attentive networking.

Magnus Boman has been a professor of Intelligent Software Services with KTH, since 2003. He defended his Ph.D. thesis on federated architectures in 1993 and is currently working on scalable federations for health data sharing in precision medicine, for which AI is employed for multimodal fusion and automated analysis of outputs from complex sensor systems.