User-Aware and Flexible Proactive Caching using LSTM and Ensemble Learning in IoT-MEC Networks

The-Vi Nguyen, Nhu-Ngoc Dao, Van Dat Tuong, Wonjong Noh, and Sungrae Cho

***Abstract*—To meet the stringent demands of emerging IoT applications such as smart home, smart city, and virtual reality in 5G/6G Internet-of-Things (IoT) networks, edge content caching for mobile/multi-access edge computing (MEC) has been identified as a promising approach to improve the quality of services in terms of latency and energy consumption. However, the limitations of cache capacity makes it difficult to develop an effective common caching framework that satisfies diverse user preferences. In this paper, we propose a new content caching strategy that maximizes the cache hit ratio through flexible prediction in dynamically changing network and user environments. It is based on a hierarchical deep learning architecture: long short-term memory (LSTM)-based local learning and ensemble-based meta-learning. First, as a local learning model, we employ an LSTM method with seasonal-trend decomposition using loess (STL)-based preprocessing. It identifies the attributes for demand prediction on the contents in various demographic user groups. Second, as a meta-learning model, we employ a regression-based ensemble learning method, which uses an online convex optimization framework and exhibits sublinear 'regret' performance. It orchestrates the obtained multiple demographic user preferences into a unified caching strategy in real-time. Extensive experiments were conducted on the popular MovieLens datasets. It was shown that the proposed control provides up to a 30% higher cache hit ratio than conventional representative algorithms and a near-optimal cache hit ratio within approximately 9% of the optimal caching scheme with perfect prior knowledge of content popularity. The proposed learning and caching control can be implemented as a core function of the 5G/6G standard's network data analytic function (NWDAF) module.**

***Index Terms*—Mobile/multi-access edge computing (MEC), Internet-of-Things (IoT), demographic classification, hierarchical learning, ensemble learning, LSTM learning, proactive caching**

I. Introduction

With the rapid development of Internet-of-Things (IoT), the number of mobile users and IoT devices has rapidly increased, and many internet services such as high-definition video, online gaming, and virtual/augmented reality (VR/AR) have been rapidly developed. Consequently, network traffic is increasing exponentially, and the Ericsson mobility report [1] forecasted that global traffic will grow by a factor of around 4.5 from 2020 to 2026. With the rapid spread of network traffic, the limited network capacity is becoming a huge challenge for mobile network operators and users.

To address this issue, mobile/multi-access edge computing (MEC) has been introduced and demonstrated as a promising solution [2], [3]. MEC is a decentralized computing network that distributes computing resources and application services from the core network of a wireless network to the edge (access) network. MEC does not use the costly and high-latency backhaul links and ensures that applications run only on access as much as possible. By this, MEC can significantly reduce network traffic, network burden, and transmission costs [2]. However, to effectively utilize the advantages of MEC, many advanced technologies such as resource allocation, multiple access control, and mobility control should be developed.

Among them, a dynamic caching control at edge nodes can be crucial for the quality-of-service (QoS) of the network operators and users [4]. Generally, serving content directly to the users from the content cloud server is impractical for real applications [5], [6]. Therefore, by caching the popular contents in the local cache of base-station (BS), content acquisition latency and backhaul load can be dramatically reduced [7]–[10]. This edge caching can also add an extra layer of security that is useful for sensitive and private data [11]. With limited caching resources in the MEC-enabled IoT networks, for effective edge caching, the network resources such as cache storage size, computing, energy, and communication bandwidth should be carefully controlled. Recently, an efficient caching control scheme for proactive caching has been attracting enormous attention. It prefetches and caches popular content or files into the caching resources that are configured for mobile edge network nodes [12]. The technique evaluates and selects the content that should be cached, updated, or replaced, and the duration for which the content should be cached by taking account of data quality, diversity, and end-user mobility, etc. One of the criteria is content popularity, which is commonly used as a very important factor for fast content retrieval [11]. In this work, we develop a flexible and efficient proactive caching scheme based on machine-learning algorithms, which can be employed for the various MEC services in 5G/6G IoT networks.

Manuscript received .....; revised .....; and accepted July 12 2021. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT) (No. NRF-2017R1D1A1B03036526, 2020R1F1A1069119). and the National Research Foundation of Korea (NRF) under Grant NRF-2019R1A2C1090447 funded by the Korea Government (Ministry of Science and ICT). *(Corresponding author: Wonjong Noh and Sungrae Cho.)*

T.-V. Nguyen, V. D. Tuong, and S. Cho are with the School of Computer Science and Engineering, Chung-Ang University, Seoul 156-756, South Korea. *(e-mail: tvnguyen@uclab.re.kr, vdtuong@uclab.re.kr, srcho@cau.ac.kr)*

N.-N. Dao is with the Department of Computer Science and Engineering, Sejong University, Seoul 05006, South Korea. *(e-mail: nndao@sejong.ac.kr)*

W. Noh is with the School of Software, Hallym University, Chuncheon 24252, South Korea. *(e-mail: wonjong.noh@hallym.ac.kr)*

A. Related Work

Traditional caching schemes [13] updated cached content based on static rules such as first-in-first-out (FIFO), least-frequently-used (LFU), and least-recently-used (LRU). However, these schemes cannot efficiently adapt to dynamic changes in the popularity of the content. Therefore, recent studies have tried to develop dynamic caching schemes, which are classified into two categories: stochastic optimization-based and machine learning-based approaches [14].

1) Stochastic optimization-based caching approach: Some content caching schemes, which have prior knowledge of content popularity, have been developed using stochastic optimization approaches. According to [12], in random wireless networks with spatially distributed network nodes, one widely-used caching strategy is probabilistic content placement, referred to as geographic caching [15] or independent random caching [16]. In [17], and [18], caching placements in random wireless networks with Poisson distributed caching helpers were studied in different scenarios. Unlike the conventional optimization approaches that target the maximization of cache hit probability [19], in [20], the authors studied ways to optimize the probability of the average success of content delivery in stochastic wireless caching helper networks in a noise-limited regime. In [21], the optimal content placement was determined by maximizing the density of successful receptions. In [12], in contrast to the conventional cache hit optimization in cache-enabled wireless networks, an alternative optimization approach for probabilistic caching placement in stochastic wireless device-to-device (D2D) caching networks was considered, considering the reliability of D2D transmissions. Using tools from stochastic geometry, this approach provides a closed-form approximation of cache-aided throughput, which measures the density of successfully served requests by local device caches and obtains the optimal caching probabilities with numerical optimization. Golrezaei *et al.* [8] introduced an architecture that combines content caching at user devices and D2D communication to improve content delivery without deploying additional infrastructures. In [22], the authors formulated the content placement problem to minimize the average download delay experienced by users in a network. In [23], the edge caching problem was formulated as a Stackelberg game between service providers (leaders) and edge nodes (followers), where service providers pay for the edge nodes' storage and backhaul resources by designed incentive mechanisms. With the same game theory, Hamidouche *et al.* [24] introduced a many-to-many matching game between small base stations and service providers' servers for the caching problem aiming at reducing delay experienced by users and the backhaul load. Knowing the demand of users, Maddah-Ali *et al.* [25] exploited the broadcast nature of the wireless medium by coded caching to improve caching efficiency. The need to improve the downlink energy efficiency of proactive content caching was conceptualized in [26], which supposes that user requests can be predicted. In many cases, the users' content request was modeled using a Zipf-like distribution [27], [28]. Based on this distribution, it is observed that only a small percentage of the ranked content are requested from the majority of users. However, using a unified popularity distribution is unreasonable because of the time-varying and dynamic nature of content popularity. Hence, many content-popularity prediction schemes based on machine learning have been proposed to handle this problem.

2) Machine-learning-based caching approach: Some content caching schemes without prior knowledge of content popularity distribution have been developed using machine learning techniques. Bharath *et al.* [29] presented a transfer learning-based method to obtain a good estimate of content popularity within a required training period. Doan *et al.* [30] proposed a deep learning-based proactive caching with content awareness, which deals with unpublished videos, i.e., newly uploaded videos with unavailable statistical information. Yang *et al.* [31] proposed a regression-based online caching scheme that considers users' location to maximize the overall cache hit ratio in a mobile edge network. Bastug *et al.* [9] proposed a caching scheme based on collaborative filtering (CF), which uses sparse training data to estimate content popularity after the training phase. Thar *et al.* [32] employed long short-term memory (LSTM) neural networks [33] that is a promising approach in modeling time-series data in the long term. It was used to learn and predict future content popularity to support cache decisions. Sengupta *et al.* [34] proposed a coded caching scheme that uses demand history to estimate the popularity of files via a combinatorial multi-armed bandit formulation. It combines the popularity estimation and content placement schemes. In [35], the authors studied a caching problem in terms of minimizing delay. A clustering algorithm was used to cluster users based on content-based similarity, and each user group is associated with a small cell base station (SCBS). By using a reinforcement learning algorithm, the SCBS learns the popularity of its user group over time. The authors [36] modeled the caching problem as a contextual multi-armed bandit problem that considers user context information such as gender and age. By using an exploitation and exploration strategy, they proposed an online caching algorithm that quickly learns content popularity by considering service differentiation. The algorithm learns the popularity of files by observing the demands for cached content and then updating the cached content at a fixed time. It is an extension of the work in [37], which considered context information such as the file request time and user density. However, these studies consider the popularity of independent content and ignore their similarity. Even though the above studies addressed the time-varying nature of content popularity, they did not consider the impact of user preference on content popularity.

B. Motivation, Contribution, and Organization

Due to the rapid emergence of new content genres and the influence of interactive and real-time content platforms such as YouTube and Facebook, user preferences for content are dynamically changing in the short and long-term. Therefore, despite the related studies discussed above, content caching for MEC is still open problem in real-world scenarios. The main contributions of this study can be summarized as follows:

- We proposed a novel proactive content caching control based on a hierarchical online learning architecture, which enables a flexible caching control with dynamically changing content popularity and diverse user preference. The proposed learning consists of LSTM-based local learning and ensemble-based meta-learning. Recently, LSTM and ensemble learnings have been successfully applied to many areas such as popularity prediction [12] [38] and recommendation [39]. However, to the best of our knowledge, there is no machine-learning-based work on joint learning models based on LSTM learning and ensemble learning to maximize the cache hit ratio in MEC networks.
- As a local learning model, we proposed an LSTM method with seasonal-trend decomposition using loess (STL)-based preprocessing for each user group. It can decompose time-specific attributes such as seasonality and trend and can account for the time-variation in popularity. This method identifies real-time content preference variations in each user group.
- As a meta learning model, we developed a regression-based ensemble framework that works online. It combines the predictions observed from individual LSTM models to obtain an overall prediction. For optimal ensemble learning, we developed an online convex optimization approach that provides sublinear 'regret' performance, i.e., the time-averaged 'regret' approaches zero as the number of optimization iterations increases. This proposed meta-learning enables flexible caching with variations in the demographics of user groups.
- We proposed a caching policy that exploits the proposed hierarchical online learning. We also proposed an edge caching architecture for MEC, which can be implemented as a core function of the 5G/6G standard's NWDAF module.

The remainder of this paper is organized as follows. Section II presents the system model and caching optimization problem. Section III explains the time series preprocessing. Section IV presents the proposed LSTM-ensemble-based hierarchical online learning model and the corresponding caching algorithm. Section V explains the proposed edge caching architecture. Experimental results are presented in Section VI. Finally, conclusions are presented in Section VII.

II. System Model and Problem Formulation

A. System Model

In Fig. 1, we consider an edge system consisting of a base station and an MEC server (MECS), which are physically co-located near the end-users and share the same power supply at the cell site. The MECS serves the content requests submitted by N mobile users (MUs). Here, MUs are uniformly distributed within the coverage region of the BS, and the MECS caches the content that can be downloaded by the MUs in the coverage areas of the BS. By caching popular contents in the MECS, the network latency can be reduced, and the duplicated transmission from the content server can be avoided. Equipped with a powerful computing capacity,

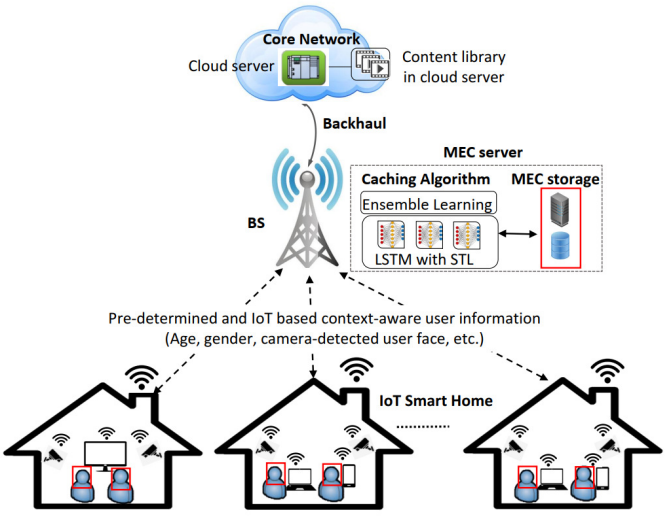


Fig. 1. System architecture of IoT-MEC learning and caching.

the MECS can perform computation-intensive tasks such as training neural network models for content popularity prediction. In our study, the MECS framework consists of artificial intelligence modules based on LSTM and ensemble learning. The proposed LSTM training module collects user experience data and analyzes the time series of content usage on various demographic user groups. Then, the outputs of the LSTM module are incorporated by an ensemble learning algorithm to predict content popularity. Utilizing this predicted popularity, we further develop an algorithm for caching content at the edge of the network. Camping on an edge, MUs can directly exploit the contents of interest if they are already stored in the edge; otherwise, the content must be retrieved from a remote cloud server through a wired or wireless backhaul link.

B. Content and Service Model

Many edge caching applications can be supported in the MEC-enabled IoT environments [40]–[42]. Also, video traffic is becoming the main part of IoT network data in [40]–[44]. Therefore, in this work, we assumed a video caching as an IoT-MEC service example. In the following, we show an example of how the proposed video caching can support the IoT system.

Example 1. We consider a practice of *smart home*, in which users enjoy movies, music videos, as shown in Fig. 1. We propose a learning-based caching scheme at the edge node exploiting users' information, which can be gathered by an IoT system in the smart home. In [45], the authors proposed content caching for self-driving cars in MEC networks. In their proposal, a CNN based-deep learning model [46] is applied to automatically extract passengers' information (age, gender, and emotion) from their images captured by cameras on the self-driving cars. Motivated by this study, we also apply this way to extract user information (age, gender, etc) from the cameras in the smart home. Also, some additional personal information can be obtained by some incentive mechanisms as in [47], [48]. Then, this information is used for clustering user groups in the edge server. Accordingly, the video requests

of such groups are recorded, preprocessed and finally fed into our proposed machine learning model for predicting video popularity and caching. Through these processes, the intelligent video caching at the MEC server provides fast and high-quality video services that meet diverse preferences of the users in the IoT environment.

In this work, to be more specific, we elaborate the proposed caching control with the movie content from [49], [50]. It enables to verify the performance of the proposed algorithm using public and popular datasets. In the MovieLens datasets [49], [50], each movie content includes information such as gender, age, and occupation. Gender feature includes two sub-features, female and male. Age feature includes seven sub-features, which represent different age ranges. Occupation feature contains twenty-one sub-features such as student, artist, etc. Accordingly, we set $\mathcal{I}$, $\mathcal{J}$, and $\mathcal{K}$ to be the sets of user groups associated with the gender, age, and occupation sub-features, respectively. Let $\mathcal{U} = \mathcal{I} \cup \mathcal{J} \cup \mathcal{K}$ denote the union of these sets, which indicates the entire user groups. On the other hand, movie genres are specified with a total of 18 movie genres. For convenience, we let $\mathcal{G} = \{1, 2, ..., 18\}$ be the set of indices representing each genre.

Remark 1. The proposed model is not only for video caching-based IoT services, but also can be applied to all types (e.g., video, audio, web pages, data) of caching-based IoT services.

C. Cache Hit Optimization Problem

We consider a movie library $\mathcal{F} = \{1, 2, ..., f, ..., F\}$, which is located on the cloud server. Movies from this library can be cached at an edge. Without loss of generality, we assume that all movies have the same size, which is normalized to 1. The cached entity at an edge has limited storage, which can cache up to C movies. Let $c_f^{(t)} \in \{0, 1\}$ denote the cache decision variable of movie $f \in \mathcal{F}$ at time slot t, where $c_f^{(t)} = 1$ if f is stored locally, and $c_f^{(t)} = 0$ otherwise. When a request for a movie f arrives at MECS, if the movie f has been stored in the cached entity of an edge, then a cache hit occurs. Otherwise, a cache miss occurs, and this movie must be fetched from the movie's cloud server.

A caching decision can be made to determine whether a movie f should be stored in the local cache and which existing movies should be removed from the cache storage. We emphasize that the caching policy in our study depends on past demand because future user demand has a close relationship with current and previous demand. To describe this dependence mathematically, we first define the *demand history matrix* as follows.

Definition 1. ***(Demand History Matrix).*** *For each user group $u \in \mathcal{U}$, the number of requests from users in this group for a specific movie genre $g \in \mathcal{G} = \{1, 2, ..., 18\}$ at time slot $t \in \{1, ..., T\}$ is recorded, denoted as $x_{g,u}^{(t)}$. We define the matrix*

$$\boldsymbol{A}_u^{(1:T)} = \begin{bmatrix} x_{1,u}^{(1)} & \cdots & x_{g,u}^{(1)} & \cdots & x_{18,u}^{(1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{1,u}^{(t)} & \cdots & x_{g,u}^{(t)} & \cdots & x_{18,u}^{(t)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{1,u}^{(T)} & \cdots & x_{g,u}^{(T)} & \cdots & x_{18,u}^{(T)} \end{bmatrix} \in \mathbb{R}^{T \times 18} \quad (1)$$

This matrix represents the demand of users in group u for all movie genres up to the current time slot T, where the t^{th} row of $\boldsymbol{A}_u^{(1:T)}$ indicates the observed demand of users in group u for 18 movie genres at the t^{th} time slot. Also, the g^{th} column of $\boldsymbol{A}_u^{(1:T)}$ indicates observed demand of users in group u on a specific movie genre g up to time slot T, which can be denoted as $X_{g,u}^{(1:T)} = \left\{x_{g,u}^{(1)}, x_{g,u}^{(2)}, \ldots, x_{g,u}^{(T)}\right\}$, i.e., a time series of demand history on movie genre g of users in group u up to time slot T. This time series will be used as the input the deep learning model.

Definition 2. ***(Caching Policy).*** *To define the caching policy formally, we denote the vector $\boldsymbol{c}^{(t)} = \left[c_1^{(t)}, \ldots, c_f^{(t)}, \ldots, c_F^{(t)}\right]^\top$ as the caching status of all movies at time slot t. In general, the caching policy is represented by a function $\zeta : \left(\boldsymbol{c}^{(t)}, \boldsymbol{A}_u^{(1:t)}, \mathcal{U}\right) \mapsto \boldsymbol{c}^{(t+1)}$ that maps from the current cache status, the demand of users for movie genres in each group, and the set of all user groups to the next cache status. Specifically, this implies that based on the observed demand history of all the user groups up to time slot t, a cache policy updates the current cache status in time slot t to the next cache status in time slot $t + 1$.*

Considering time sequence $\mathcal{T} = \{T + 1, T + 2, ..., T + M\}$, where M is a finite positive number, we use the overall cache hit ratio to measure the caching performance in $\mathcal{T}$. For a given caching policy ζ, cache hit ratio is defined as the number of cache hits over the number of requests across $\mathcal{T}$

$$\mathcal{R}(\zeta) = \frac{\sum_{t \in \mathcal{T}} \sum_{f \in \mathcal{F}} c_f^{(t)} d_f^{(t)}}{\sum_{t \in \mathcal{T}} N_t}, \quad (2)$$

where N_t is the number of movie requests at time slot t, and $d_f^{(t)}$ is the demand of the whole user population for movie f at time slot t. Table I briefly summarizes the notations used in this paper.

The objective of this study is to find a caching policy ζ at the edge that maximizes the overall cache hit ratio over a given time period $\mathcal{T}$. The cache hit optimization problem can be defined as

$$\max_{\zeta} \ \mathcal{R}(\zeta)$$

$$\text{s.t.} \ \sum_{f \in \mathcal{F}} c_f^{(t)} \leq C, \quad \forall t \in \mathcal{T}, \quad (3)$$

$$c_f^{(t)} \in \{0, 1\}, \quad \forall f \in \mathcal{F}, \text{ and } t \in \mathcal{T}, \quad (4)$$

where (3) denotes the limited caching capacity at MECS and (4) denotes the binary caching decision on movie f at time slot t. In this work, as a solution, we propose a novel caching

TABLE I
TABLE OF NOTATIONS

Notation	Description
f, $\mathcal{F}$	Movie, movie library, respectively
$c_f^{(t)} \in \{0,1\}$	Cache decision variable at time slot t for movie f
C	Cache capacity at edge
$\mathcal{I}$, $\mathcal{J}$, $\mathcal{K}$, $\mathcal{U}$, and $\mathcal{G}$	Sets of users associated with gender features, age features, occupation features, and the union of the former sets, and the set of movie genres, respectively
T, $\mathcal{T}$	Length of the time series of user group's demand, set of future time slots starting from time slot $T+1$ to $T+M$, respectively
$x_{g,u}^{(t)}$, $\bar{x}_{g,u}^{(t)}$, and $\hat{x}_{g,u}^{(t)}$	Number of requests by users in group u for movie genre g at time slot t, number of requests by users in group u for movie genre g at time slot t after preprocessing, and predicted number of requests by users in group u for movie genre g at time slot t after postprocessing, respectively
$X_{g,u}^{(1:T)}$, $\boldsymbol{A}_u^{(1:T)}$	Observed demand history of users in group u on movie genre g at the beginning to time slot T, matrix of user demands for movie genres in group u up to time slot T, respectively
$w_{g,u}^{(t)}$, $y_g^{(t)}$, $\hat{y}_g^{(t)}$	Weight assigned for the demand of user group u for movie genre g in the ensemble model, average (actual) demand of all user groups, predicted demand for movie genre g across user groups at time slot t, respectively

control that exploits the deep learning modules of LSTM and ensemble learning.

III. Time Series Preprocessing

Before training the deep learning model that learns user preference for movie genres, it is necessary to preprocess the raw data. In this section, we describe the data preprocessing for a time series $X_{g,u}^{(1:T)}$ associated with the demand of a movie genre $g \in \mathcal{G}$ in a user group $u \in \mathcal{U}$.

A. Seasonal and Trend Decomposition

To better understand a time series and make better predictions from it, it is very helpful to extract some patterns such as *seasonal* and *trend* components from the time series [51]. The seasonal components in time series are repeating patterns or cycles; they repeat regularly over time, and are affected by seasonal factors such as the hours in a day and days in a week. The trend components refer to the long-term increase or decrease in time series. According to [51], these seasonal and trend components can be combined using one of the following approaches:

$$x_{g,u}^{(t)} = S_{g,u}^{(t)} + T_{g,u}^{(t)} + R_{g,u}^{(t)}, \tag{5}$$

$$x_{g,u}^{(t)} = S_{g,u}^{(t)} \times T_{g,u}^{(t)} \times R_{g,u}^{(t)}, \tag{6}$$

where $x_{g,u}^{(t)}$ is an observation at time t of time series $X_{g,u}^{(1:T)}$, which consists of a seasonal component $S_{g,u}^{(t)}$, a trend component $T_{g,u}^{(t)}$, and a residual or remainder component $R_{g,u}^{(t)}$. The time series decompositions in (5) and (6) are called *additive* decomposition and *multiplicative* decomposition, respectively.

On the other hand, it is known that multiplicative decomposition is more appropriate when the magnitude of the seasonal variation is dependent on the level of the time series, which is the average value of the series [51]. For example, as shown in Fig. 2, our data has such a tendency, so we apply multiplicative decomposition. However, instead of directly using a multiplicative decomposition for our time series data, by the following equivalent relation, we first use data log-transform, and then utilize an additive decomposition.

$$\begin{aligned} & x_{g,u}^{(t)} = S_{g,u}^{(t)} \times T_{g,u}^{(t)} \times R_{g,u}^{(t)} \\ \Leftrightarrow\ & \log(x_{g,u}^{(t)}) = \log(S_{g,u}^{(t)}) + \log(T_{g,u}^{(t)}) + \log(R_{g,u}^{(t)}). \end{aligned} \tag{7}$$

Here, the log-transformation of $x_{g,u}^{(t)}$ in time series $X_{g,u}^{(1:T)}$, $\tilde{x}_{g,u}^{(t)}$, is defined by

$$\tilde{x}_{g,u}^{(t)} = \begin{cases} \log\left(x_{g,u}^{(t)}\right), & \min\left(X_{g,u}^{(1:T)}\right) > 0, \\ \log\left(x_{g,u}^{(t)} + 1\right), & \min\left(X_{g,u}^{(1:T)}\right) = 0, \end{cases} \tag{8}$$

The reason why we employ the log-transformation is that it can stabilize the time series when it offers many small observations, e.g., in Fig. 2, after the 300$^{\text{th}}$ day, there are many days in which a movie genre has zero views. After the log-transformation, we employ a widely used additive decomposition method, STL [52]. The extracted seasonal and trend components are exploited to normalize and deseasonalize the time series.

Remark 2. Deep-learning-based prediction models, such as the LSTM, often have difficultly capturing all the complex seasonal cycles. To mitigate this problem, time series components, e.g., seasonal and trend components, are modeled separately; hence, the complexity of the model is reduced compared with modeling the entire time series. In many previous studies [53], [54], removing seasonality improved the prediction results of the neural network model.

B. Moving-window Transformation, Time Series Normalization, and Deseasonalization

1) Moving-window transformation: In this step, the time series is transformed into multiple input and output frames, which are later fed directly into the learning model as training data. As illustrated in Fig. 3, by using the moving-window strategy, an input window (window size of p) initially covers the first p observed data points, and an output window (window size of q) covers the next q points, which are represented as dashed squares. Afterwards, two windows are moved (slid) one time-step to the right, and new input and output windows are generated, which are represented as lined squares. The input and output windows are generated in this manner until the last output window reaches the last observation point of the time series. As our problem is one day ahead prediction, we choose $q = 1$. The result is represented by

$$\left(\boldsymbol{X}_{g,u}^{\text{win}}, \boldsymbol{y}_{g,u}^{\text{win}}\right) = \left(\begin{bmatrix} \tilde{x}_{g,u}^{(1)} & \cdots & \tilde{x}_{g,u}^{(p)} \\ \tilde{x}_{g,u}^{(2)} & \cdots & \tilde{x}_{g,u}^{(p+1)} \\ \vdots & \ddots & \vdots \\ \tilde{x}_{g,u}^{(T-p)} & \cdots & \tilde{x}_{g,u}^{(T-1)} \end{bmatrix}, \begin{bmatrix} \tilde{x}_{g,u}^{(p+1)} \\ \tilde{x}_{g,u}^{(p+2)} \\ \vdots \\ \tilde{x}_{g,u}^{(T)} \end{bmatrix} \right), \tag{9}$$

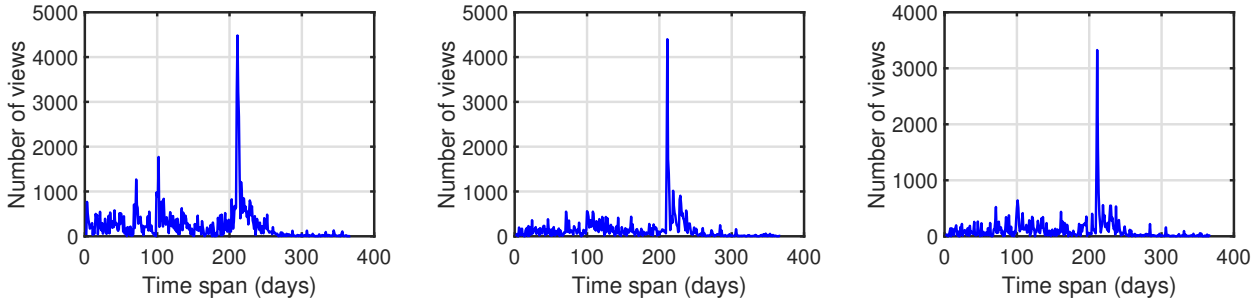


Fig. 2. Time series for the drama genre subjected to three specific groups of users in the Movielens 1M dataset [50]: (a) female group, (b) age group between 18 and 24 years, (c) student group.

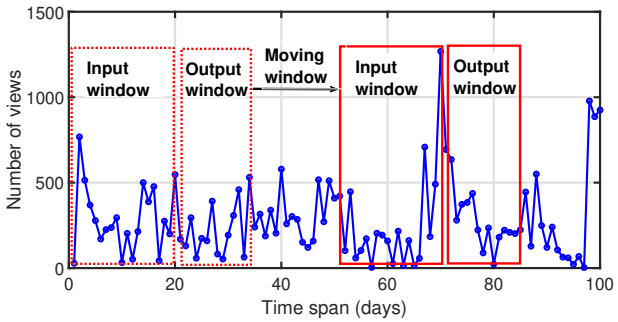


Fig. 3. Illustration of a time series and moving window transformation.

where each row of $\boldsymbol{X}_{g,u}^{\text{win}}$ and $\boldsymbol{y}_{g,u}^{\text{win}}$ shows the observations in an input window and an output window, respectively.

2) Time-series normalization and deseasonalization: In LSTM cells, when activation functions such as hyperbolic tangents and sigmoids are used, saturation occurs when the input is very positive or very negative. Therefore the output is constant, which further increases the difficulty of gradient-based learning [55]. Therefore, it is necessary to normalize the input data so that they do not fall in this saturated range. Even if LSTM is not used, normalization is important when the time series have different amplitudes [56]. Therefore, within moving-window processing, the trend value of the last data point in each input window, provided by STL decomposition, is used for local normalization. Specifically, it is subtracted from each data point in the corresponding input and output windows. This process is applied to each input and output window separately [57]. Consequently, the data in (9) becomes

$$\left(\boldsymbol{X}_{g,u}^{\text{norm}}, \boldsymbol{y}_{g,u}^{\text{norm}}\right) = \left(\begin{bmatrix} \breve{x}_{g,u}^{(1)} & \cdots & \breve{x}_{g,u}^{(p)} \\ \breve{x}_{g,u}^{(2)} & \cdots & \breve{x}_{g,u}^{(p+1)} \\ \vdots & \ddots & \vdots \\ \breve{x}_{g,u}^{(T-p)} & \cdots & \breve{x}_{g,u}^{(T-1)} \end{bmatrix}, \begin{bmatrix} \breve{x}_{g,u}^{(p+1)} \\ \breve{x}_{g,u}^{(p+2)} \\ \vdots \\ \breve{x}_{g,u}^{(T)} \end{bmatrix}\right), \quad (10)$$

where on the t^{th} row of $\left(\boldsymbol{X}_{g,u}^{\text{norm}}, \boldsymbol{y}_{g,u}^{\text{norm}}\right)$, each entry is defined by $\breve{x}_{g,u}^{(i)} := \bar{x}_{g,u}^{(i)} - T_{g,u}^{(p+t-1)}, \forall i \in \{t, ..., t+p\}$. Here, $T_{g,u}^{(p+t-1)}$ is the trend value of the last data point in the t^{th} row of $\boldsymbol{X}_{g,u}^{\text{win}}$. Next, we process the deseasonalization on the normalized data, which removes the seasonal patterns from the data. We can remove the seasonal component on the current data point by subtracting the value from the previous window. As a result, the data can be expressed as follows:

$$\left(\boldsymbol{X}_{g,u}^{\text{deseasonal}}, \boldsymbol{y}_{g,u}^{\text{deseasonal}}\right) = \left(\begin{bmatrix} \bar{x}_{g,u}^{(1)} & \cdots & \bar{x}_{g,u}^{(p)} \\ \bar{x}_{g,u}^{(2)} & \cdots & \bar{x}_{g,u}^{(p+1)} \\ \vdots & \ddots & \vdots \\ \bar{x}_{g,u}^{(T-p)} & \cdots & \bar{x}_{g,u}^{(T-1)} \end{bmatrix}, \begin{bmatrix} \bar{x}_{g,u}^{(p+1)} \\ \bar{x}_{g,u}^{(p+2)} \\ \vdots \\ \bar{x}_{g,u}^{(T)} \end{bmatrix}\right), \quad (11)$$

where on the t^{th} row of $\left(\boldsymbol{X}_{g,u}^{\text{deseasonal}}, \boldsymbol{y}_{g,u}^{\text{deseasonal}}\right)$, each entry is defined by $\bar{x}_{g,u}^{(i)} := \breve{x}_{g,u}^{(i)} - S_{g,u}^{(i-1)}, \forall i \in \{t, ..., t+p\}, t \geq 2$. Here, $S_{g,u}^{(i-1)}$ is the seasonal value of the data point in the $(i-1)^{\text{th}}$ row of $\boldsymbol{X}_{g,u}^{\text{norm}}$.

IV. Hierarchical LSTM-Ensemble Learning Based Proactive Caching

A. User Group's Behavior and LSTM-Ensemble Learning

1) User group's behavior: In addition to the specific preference of each user group for each movie genre, it can be observed that each user group has its specific preference for watching time, i.e., the period during which they often enjoy the movie, depending upon the hour or the day. For example, based on the preference for days of a week, student groups watch movies during a week, while teacher groups enjoy movies only at the weekend. Based on the time of the day, student groups watch movies often in the evening and early morning, while teacher groups often watch movies in the evening. Thus, it can be seen that different groups tend to have different preference in watching time. Therefore, leveraging user preferences for movie genre and time will improve the prediction performance of the neural networks. Based on this observation, we build an LSTM model for each group instead of using a single LSTM model for the entire user population. Then, the resulting predictions for each group are averaged to obtain the prediction across the user population.

2) Introduction to LSTM-based Ensemble Learning: LSTMs are variants of recurrent neural networks (RNNs) which are designed for learning long-term dependencies [33]. LSTMs consist of a chain of repeating modules (cells) of neural networks. Due to the self-contained memory-cell state and the gating mechanism, the LSTM network can learn the long-term dependencies in the series. Specifically, an

LSTM cell is composed of a cell state, forget gate, input gate, and output gate at time t. Cell status $c^{(t)}$ refers to the cell information to be transferred to the subsequent cells in the long-term. Forget gate with sigmoid function decides to remove irrelevant information from the previous cell state $c^{(t-1)}$, whose output is denoted as $f^{(t)}$. Input gate decides what new information will be used to update the cell state $c^{(t)}$, whose output is denoted as $i^{(t)}$. Output gate decides what will be output based on the updated cell state and the output of the sigmoid function, whose output is denoted as $o^{(t)}$. The following formulas show how a single LSTM cell works to map an input $\bar{x}_{g,u}^{(t)}$ to an output $\hat{s}_{g,u}^{(t+1)}$:

- Forget gate:

$$f^{(t)} = \text{sigmoid}\left(W_f \cdot \left[\bar{x}_{g,u}^{(t)}, h^{(t-1)}\right] + b_f\right), \tag{12}$$

- Input gate:

$$i^{(t)} = \text{sigmoid}\left(W_i \cdot \left[\bar{x}_{g,u}^{(t)}, h^{(t-1)}\right] + b_i\right), \tag{13}$$

- Cell state:

$$\begin{aligned} \tilde{c}^{(t)} &= \tanh\left(W_c \cdot \left[\bar{x}_{g,u}^{(t)}, h^{(t-1)}\right] + b_c\right), \\ c^{(t)} &= f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)}, \end{aligned} \tag{14}$$

- Output gate:

$$\begin{aligned} o^{(t)} &= \text{sigmoid}\left(W_o \cdot \left[\bar{x}_{g,u}^{(t)}, h^{(t-1)}\right] + b_o\right), \\ h^{(t)} &= o^{(t)} \odot \tanh(c^{(t)}), \\ \hat{s}_{g,u}^{(t+1)} &= h^{(t)}, \end{aligned} \tag{15}$$

where the subscripts f, i, o and c stand for forget gate, input gate, output gate, and cell state, respectively. Accordingly, (W_f, b_f), (W_i, b_i), (W_o, b_o), and (W_c, b_c) are the pairs of weight and bias, respectively. The notation 'tanh' denotes the hyperbolic tangent function, 'sigmoid' denotes the logistic sigmoid function, and $\odot$ denotes the Hadarmard product (i.e., element-wise product).

On the other hand, ensemble learning is a learning technique that trains multiple learners and combines them. Here, the term "learner" refers to a model generated from training data using a learning framework such as a neural network or a decision tree. There are different ways to combine a set of learners, such as bagging, boosting, and stacking [38]. In this study, we utilize the *stacking* ensemble method in which a learner is trained to combine the individual learners. Here, each learner is called a *first-level learner* and the combiner is called a *second-level learner* or *meta-learner*. In the stacking procedure, first-level learners are trained on the training data, and then the meta-learner is trained using the predictions of the first-level learners. In this study, because we aim to predict demand for each movie genre g (18 movie genres in total) across user groups, we utilize 18 stacking models. In Fig. 4, we illustrate an ensemble procedure for predicting the user demand for a specific movie genre g across user groups. In each user group, an LSTM based first-level learner is trained on the preprocessed data to predict user demand for genre g. Then, the predictions of first-level learners across user groups (for the same movie genre g) are aggregated as an input dataset for meta-leaner learning, to predict the demand for genre g over user groups. Here, the meta-learning uses a regression model to learn the optimal weights for combining the first-level predictions.

B. LSTM-based First-level Learner Training

In this subsection, we describe the proposed first-level learning in detail.

1) Data preparation: After the data preprocessing (see Section III), the preprocessed data (11) according to each genre g, and user group u, is partitioned into two parts. We train an LSTM-based first-level learner on the first part. After training, the learner predicts the future value on the remaining part. Specifically, they are expressed as follows

$$\left(\boldsymbol{X}_{g,u}^{LSTM}, \boldsymbol{y}_{g,u}^{LSTM}\right) = \left(\begin{bmatrix} \bar{x}_{g,u}^{(1)} & \cdots & \bar{x}_{g,u}^{(p)} \\ \bar{x}_{g,u}^{(2)} & \cdots & \bar{x}_{g,u}^{(p+1)} \\ \vdots & \ddots & \vdots \\ \bar{x}_{g,u}^{(m-p)} & \cdots & \bar{x}_{g,u}^{(m-1)} \end{bmatrix}, \begin{bmatrix} \bar{x}_{g,u}^{(p+1)} \\ \bar{x}_{g,u}^{(p+2)} \\ \vdots \\ \bar{x}_{g,u}^{(m)} \end{bmatrix}\right), \tag{16}$$

$$\left(\tilde{\boldsymbol{X}}_{g,u}^{LSTM}, \tilde{\boldsymbol{y}}_{g,u}^{LSTM}\right) = \left(\begin{bmatrix} \bar{x}_{g,u}^{(m-p+1)} & \cdots & \bar{x}_{g,u}^{(m)} \\ \bar{x}_{g,u}^{(m-p+2)} & \cdots & \bar{x}_{g,u}^{(m+1)} \\ \vdots & \ddots & \vdots \\ \bar{x}_{g,u}^{(T-p)} & \cdots & \bar{x}_{g,u}^{(T-1)} \end{bmatrix}, \begin{bmatrix} \bar{x}_{g,u}^{(m+1)} \\ \bar{x}_{g,u}^{(m+2)} \\ \vdots \\ \bar{x}_{g,u}^{(T)} \end{bmatrix}\right), \tag{17}$$

Here, the training data size $m-p$ of the first partition occupies a proportion of α of the total data size $T-p$, i.e., $m-p = \lfloor \alpha(T-p) \rfloor$, where $\lfloor x \rfloor$ denotes *floor function* that returns the greatest integer number less than or equal to x. Therefore, in the experimental evaluation,

$$m = \lfloor \alpha \cdot (T-p) \rfloor + p \tag{18}$$

In other words, m is determined by taking the integer part of the proportion α of the total data points in $\left(\boldsymbol{X}_{g,u}^{deseasonal}, \boldsymbol{y}_{g,u}^{deseasonal}\right)$ plus the input window size p.

2) LSTM training and prediction: Let $\mathfrak{L}_u$ denote the LSTM neural network associated with user group u. For the user group u, we define each first-level learner for each genre g as

$$\phi_{g,u} = \mathfrak{L}_u\left(\boldsymbol{X}_{g,u}^{LSTM}, \boldsymbol{y}_{g,u}^{LSTM}\right), \tag{19}$$

which is trained on $\boldsymbol{X}_{g,u}^{LSTM}$ using the LSTM model $\mathfrak{L}_u$. After training with LSTM models in group u, each learner $\phi_{g,u}$ predicts the future user demand for movie genre g within time slots $t = \{m+1, ..., T\}$ as

$$\hat{\boldsymbol{s}}_{g,u} = \phi_{g,u}\left(\tilde{\boldsymbol{X}}_{g,u}^{LSTM}\right), \tag{20}$$

where the vector $\hat{\boldsymbol{s}}_{g,u} = [\hat{s}_{g,u}^{(m+1)}, ..., \hat{s}_{g,u}^{(T)}]^\top$.

3) Postprocessing of predicted results: Postprocessing is conducted to reverse the effects of the preprocessing performed on the predicted vector $\hat{\boldsymbol{s}}_{g,u} = \left[\hat{s}_{g,u}^{(m+1)}, \ldots, \hat{s}_{g,u}^{(T)}\right]^\top$. The postprocessing proceeds in the reverse order of preprocessing as follows [57]. The reseasonalization step includes introducing the last seasonal component to the generated predicted

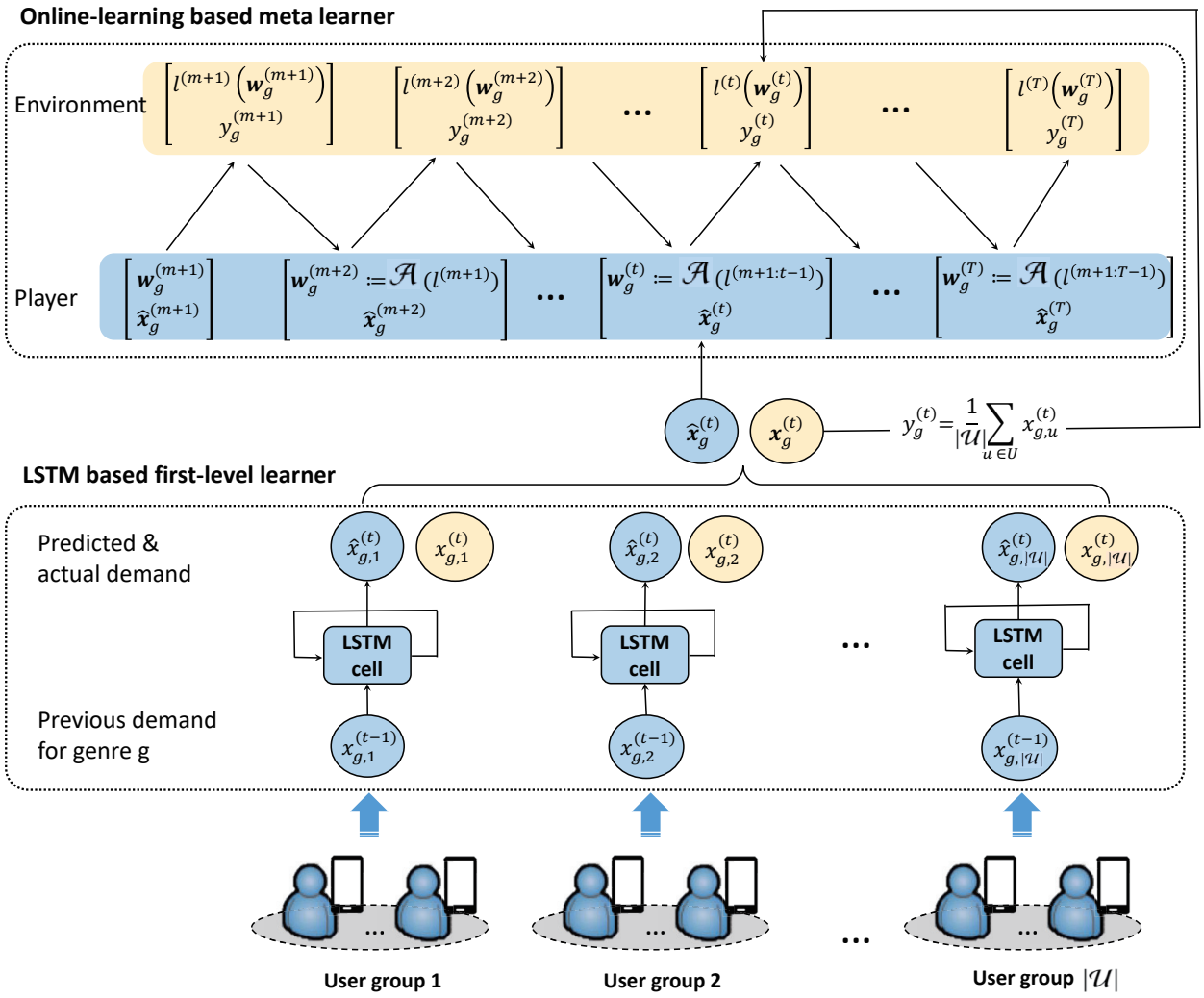


Fig. 4. Ensemble procedure for predicting user demand on movie genre g across user groups.

values. Next, in the renormalization step, the generated predictions are transformed back to their original scale by adding the last value of the trend inside an input window and finally taking the exponent of the values. After the postprocessing step, we obtain a prediction vector at the original scale, i.e., $\hat{\boldsymbol{x}}_{g,u} = \left[\hat{x}_{g,u}^{(m+1)}, \ldots, \hat{x}_{g,u}^{(T)}\right]^{\top}$.

C. Meta-Learner Training

In this subsection, we describe the proposed ensemble learning in detail.

1) Data Preparation: As shown in Fig. 4, the prediction vectors of the first-level learners associated with the same movie genre are aggregated and used to form the input data for the meta-learner, as follows

$$\left(\boldsymbol{X}_g^{meta}, \boldsymbol{y}_g^{meta}\right) = \left(\left[\{\hat{\boldsymbol{x}}_{g,u}\}_{u\in\mathcal{U}}\right], \left[y_g^{(m+1)}, \ldots, y_g^{(T)}\right]\right), \quad (21)$$

where $\boldsymbol{X}_g^{meta}$ and $\boldsymbol{y}_g^{meta}$ are the input and output for training the meta-learner, respectively; $y_g^{(t)}$ is the average (actual) demand of all user groups for movie genre g at time t, which is computed by

$$y_g^{(t)} = \frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}} x_{g,u}^{(t)}, \quad \forall t \in \{m+1, ..., T\}, \forall g \in \mathcal{G}, \quad (22)$$

where $|\mathcal{U}|$ indicates the number of user groups in $\mathcal{U}$, which is thirty including two gender groups, seven age groups, and twenty-one occupation groups for the MovieLens datasets; $x_{g,u}^{(t)}$ is the actual demand of a user group u for movie genre g at time t. We also denote $\boldsymbol{x}_g^{(t)} = [x_{g,u}^{(t)}]_{u\in\mathcal{U}}$ to represent the collection of actual demands for genre g across user groups (illustrated in Fig 4).

2) Stacking regression model: Here, a second-level learner is trained to combine the prediction values. At each time step t, for a given movie genre g, weighted averaging is performed to predict the number of views for the specific movie genre across user groups in $\mathcal{U}$, as follows

$$\hat{y}_g^{(t)} = \sum_{u\in\mathcal{U}} w_{g,u}^{(t)} \hat{x}_{g,u}^{(t)}, \quad \forall g \in \mathcal{G}. \quad (23)$$

By denoting the vectors $\boldsymbol{w}_g^{(t)} = [w_{g,u}^{(t)}]_{u\in\mathcal{U}}$ and $\hat{\boldsymbol{x}}_g^{(t)} = [\hat{x}_{g,u}^{(t)}]_{u\in\mathcal{U}}$, (23) becomes

$$\hat{y}_g^{(t)} = \left(\boldsymbol{w}_g^{(t)}\right)^\top \hat{\boldsymbol{x}}_g^{(t)}, \quad \forall g \in \mathcal{G}. \tag{24}$$

Normally, the parameter vector $\boldsymbol{w}_g^{(t)}$ is estimated by minimizing the sum of square errors

$$\min_{\left\{\boldsymbol{w}_g^{(t)}, \forall t\in\{m+1,\ldots,T\}\right\}} \quad \sum_{t=m+1}^{T} \left(y_g^{(t)} - \hat{y}_g^{(t)}\right)^2. \tag{25}$$

If the all values of $y_g^{(t)}$ over $t \in [m+1,\ldots,T]$ are all exploited together, common convex optimization solvers can be employed to solve (25). However, for a sequential prediction problem, when predicting the value of $\hat{y}_g^{(t)}$, it is reasonable to assume that only the information on $y_g^{(t)}$ over $t \in [m+1,\ldots,t-1]$ is available at the time t. Therefore, solving Problem (25) in an offline learning fashion is intractable. To resolve this issue, we apply *online convex optimization* (OCO) theory [58]–[60]. The fundamental framework of OCO [58] is described in the following.

Definition 3. ***(Online convex optimization framework).***
In the online convex optimization problem, there is a convex set $C \subseteq \mathbb{R}^{|\mathcal{U}|}$, a sequence of convex functions $\{l^{(t)}\}$, where $l^{(t)} : C \to \mathbb{R}$. At each time step t, an online algorithm $\mathcal{A}$ chooses a vector $\boldsymbol{w}^{(t)} \in C$. After that, the cost of $l^{(t)}(\boldsymbol{w}^{(t)})$ is revealed.

The OCO problem is naturally considered as a repeated game [58]. In particular, we build our linear regression model using the following game framework repeated between a player (learner) and environment (see the online-learning based meta-learner in Fig. 4). At each round t, an instance described by a feature vector $\hat{\boldsymbol{x}}_g^{(t)}$ arrives. A player proposes a model parameter vector $\boldsymbol{w}_g^{(t)}$ by using an algorithm, denoted by $\mathcal{A}$, and predicts $\hat{y}_g^{(t)} = \left(\boldsymbol{w}_g^{(t)}\right)^\top \hat{\boldsymbol{x}}_g^{(t)}$. Then, the environment reveals the true value $y_g^{(t)}$, and a loss (cost) function of the form $l^{(t)}(\boldsymbol{w}) := \left(\boldsymbol{w}^\top \hat{\boldsymbol{x}}_g^{(t)} - y_g^{(t)}\right)^2$. The algorithm $\mathcal{A}$ chooses $\boldsymbol{w}_g^{(t)}$ based on the previous information including $l^{(m+1:t-1)} := \left[l^{(m+1)}, ..., l^{(t-1)}\right]$ (this will be explained later). To measure the performance of the algorithm $\mathcal{A}$, we use a standard measure, called *regret*. The regret is the difference between the accumulative loss of the algorithm and that of the best single solution in hindsight. Here, the "hindsight" knows all the loss functions and the best single solution over all rounds. For our problem, it can be defined as follows.

Definition 4. ***(Regret).***
The player suffers a loss, which is the discrepancy between the predicted and true value

$$l^{(t)}(\boldsymbol{w}_g^{(t)}) = \left(\left(\boldsymbol{w}_g^{(t)}\right)^\top \hat{\boldsymbol{x}}_g^{(t)} - y_g^{(t)}\right)^2. \tag{26}$$

The accumulative loss that the player with the algorithm $\mathcal{A}$ suffers after $T-m$ rounds is defined by

$$L_{\mathcal{A}} = \sum_{t=m+1}^{T} \left(\left(\boldsymbol{w}_g^{(t)}\right)^\top \hat{\boldsymbol{x}}_g^{(t)} - y_g^{(t)}\right)^2. \tag{27}$$

The loss of a specific vector $\boldsymbol{w}_g$ is defined by

$$L_{\boldsymbol{w}_g} = \sum_{t=m+1}^{T} \left(\boldsymbol{w}_g^\top \hat{\boldsymbol{x}}_g^{(t)} - y_g^{(t)}\right)^2. \tag{28}$$

Then, the player aims to minimize the regret, which is given by

$$\textit{Regret} = L_{\mathcal{A}} - \min_{\boldsymbol{w}_g} L_{\boldsymbol{w}_g}. \tag{29}$$

The regret is the difference between the suffered loss and minimum loss, which is obtained by the best vector $\boldsymbol{w}_g$. We could have chosen this vector if we knew the data offline.

Our *goal* is to find the algorithm $\mathcal{A}$ that possesses a sequence of model parameters $\{\boldsymbol{w}_g^{(t)}\}$ that minimizes the player's regret. In general, the algorithm $\mathcal{A}$ generates the sequence $\{\boldsymbol{w}_g^{(t)}\}$ as

$$\boldsymbol{w}_g^{(t+1)} = \arg\min_{\boldsymbol{w_g}} \left(l^{(t)}(\boldsymbol{w}_g)\right). \tag{30}$$

In this study, we utilize a typical example of algorithm $\mathcal{A}$, *follow-the-proximally-regularized-leader* (FTRL-Proximal) algorithm [61], [62] that updates the model's parameter iteratively

$$\boldsymbol{w}_g^{(t+1)} = \arg\min_{\boldsymbol{w_g}} \left\{\left(\boldsymbol{h}_g^{(m+1:t)}\right)^\top \boldsymbol{w}_g + \frac{1}{2}\sum_{s=m+1}^{t} \sigma^{(s)} \|\boldsymbol{w}_g - \boldsymbol{w}_g^{(s)}\|_2^2 + \lambda_1\|\boldsymbol{w}_g\|_1 + \frac{1}{2}\lambda_2\|\boldsymbol{w}_g\|_2^2\right\}, \tag{31}$$

where $\boldsymbol{h}_g^{(m+1:t)} = \sum_{s=m+1}^{t} \boldsymbol{h}_g^{(s)}$, with $\boldsymbol{h}_g^{(s)} = \nabla l^{(s)}(\boldsymbol{w}_g^{(s)}) = 2\left((\boldsymbol{w}_g^{(s)})^\top \hat{\boldsymbol{x}}_g^{(s)} - y_g^{(s)}\right)\hat{\boldsymbol{x}}_g^{(s)}$ denotes the gradient vector of the loss function with respect to $\boldsymbol{w}_g^{(s)}$ at time s; $\sigma^{(s)}$ is the learning rate schedule such that $\sum_{s=m+1}^{t} \sigma^{(s)} = \frac{1}{\eta^{(t)}}$. It applies the linearization form of the loss function added with the ℓ_1 and ℓ_2 regularization terms to handle the over-fitting problem and model complexity [63]. The notations $\|\cdot\|_1$ and $\|\cdot\|_2$ denote the ℓ_1-norm and ℓ_2-norm, respectively.

By definition, we have $\|\boldsymbol{w}_g - \boldsymbol{w}_g^{(s)}\|_2^2 = \left(\boldsymbol{w}_g - \boldsymbol{w}_g^{(s)}\right)^\top \left(\boldsymbol{w}_g - \boldsymbol{w}_g^{(s)}\right) = \|\boldsymbol{w}_g\|_2^2 - 2\left(\boldsymbol{w}_g^{(s)}\right)^\top \boldsymbol{w}_g + \|\boldsymbol{w}_g^{(s)}\|_2^2$. Substituting this into problem (31), we have the following equivalent problem

$$\boldsymbol{w}_g^{(t+1)} = \arg\min_{\boldsymbol{w_g}} \left\{\left(\boldsymbol{h}_g^{(m+1:t)} - \sum_{s=m+1}^{t} \sigma^{(s)}\boldsymbol{w}_g^{(s)}\right)^\top \boldsymbol{w}_g + \frac{1}{2}\left(\lambda_2 + \sum_{s=m+1}^{t}\sigma^{(s)}\right)\|\boldsymbol{w}_g\|_2^2 + \lambda_1\|\boldsymbol{w}_g\|_1 + \frac{1}{2}\sum_{s=m+1}^{t}\sigma^{(s)}\|\boldsymbol{w}_g^{(s)}\|_2^2\right\}. \tag{32}$$

Let $\boldsymbol{\vartheta}^{(t)} = \boldsymbol{h}_g^{(m+1:t)} - \sum_{s=m+1}^{t} \sigma^{(s)} \boldsymbol{w}_g^{(s)}$, then the relationship between $\boldsymbol{\vartheta}^{(t)}$ and $\boldsymbol{\vartheta}^{(t-1)}$ is represented as follows

$$\boldsymbol{\vartheta}^{(t)} = \boldsymbol{\vartheta}^{(t-1)} + \boldsymbol{h}_g^{(t)} - \left(\frac{1}{\eta^{(t)}} - \frac{1}{\eta^{(t-1)}}\right)\boldsymbol{w}_g^{(t)}. \tag{33}$$

This means that we only need to store $\boldsymbol{\vartheta}^{(t-1)}$ and use it to update $\boldsymbol{\vartheta}^{(t)}$ at the beginning of the round t. Then, problem (32) can be rewritten as

$$\boldsymbol{w}_g^{(t+1)} = \arg\min_{\boldsymbol{w}_g} \left\{ \left(\boldsymbol{\vartheta}^{(t)}\right)^T \boldsymbol{w}_g + \frac{1}{2}\left(\lambda_2 + \sum_{s=m+1}^{t} \sigma^{(s)}\right)\|\boldsymbol{w}_g\|_2^2 + \lambda_1 \|\boldsymbol{w}_g\|_1 \right\}, \tag{34}$$

where the last component of problem (32) is omitted in problem (34) since it is a constant with respect to $\boldsymbol{w}_g$. Problem (34) can be decoupled into $|\mathcal{U}|$ per-coordinate-based problems as follows

$$w_{g,u}^{(t+1)} = \arg\min_{w_{g,u}} \left\{ \vartheta_u^{(t)} w_{g,u} + \frac{1}{2}\left(\lambda_2 + \sum_{s=m+1}^{t} \sigma_u^{(s)}\right)(w_{g,u})^2 + \lambda_1 |w_{g,u}| \right\}, \quad \forall u \in \mathcal{U}, \tag{35}$$

where $w_{g,u}^{(t)}$, $h_{g,u}^{(t)}$, and $\vartheta_u^{(t)}$ is the u^{th} component of the vector $\boldsymbol{w}_g^{(t)} = [w_{g,u}^{(t)}]_{u\in\mathcal{U}}$, $\boldsymbol{h}_g^{(t)} = [h_{g,u}^{(t)}]_{u\in\mathcal{U}}$ and $\boldsymbol{\vartheta}^{(t)} = [\vartheta_u^{(t)}]_{u\in\mathcal{U}}$, respectively. Let $\sum_{s=m+1}^{t} \sigma_u^{(s)} = \frac{1}{\eta_u^{(t)}}$, where $\eta_u^{(t)} = \frac{\alpha}{\beta + \sqrt{\sum_{s=m+1}^{t} (h_{g,u}^{(t)})^2}}$, where α, β are chosen to obtain good learning performance [61]. The solution of $w_{g,u}^{(t+1)}$ can be obtained in closed using the following theorem.

Theorem 1. *The closed-form solution of problem (35) can be represented as follows*

$$w_{g,u}^{(t+1)} = \begin{cases} 0 & \text{if } |\vartheta_u^{(t)}| \le \lambda_1, \\ \frac{\vartheta_u^{(t)} - sgn(\vartheta_u^{(t)})\lambda_1}{\frac{1}{\eta_u^{(t)}} + \lambda_2} & \text{otherwise}, \end{cases} \quad \forall u \in \mathcal{U}. \tag{36}$$

Proof. Because problem (35) is a convex and unconstrained problem, the optimal value $w_{g,u}^{(t+1)}$ can be obtained by setting the first-order derivative of the corresponding objective function to zero as follows:

$$\vartheta_u^{(t)} + \left(\lambda_2 + \sum_{s=m+1}^{t} \sigma_u^{(s)}\right) w_{g,u}^{(t+1)} + \lambda_1 \phi = 0, \quad \forall u \in \mathcal{U} \tag{37}$$

where ϕ is defined as

$$\phi = \begin{cases} -1 & \text{if } w_{g,u}^{(t+1)} < 0, \\ 1 & \text{if } w_{g,u}^{(t+1)} > 0, \\ [-1, 1] & \text{if } w_{g,u}^{(t+1)} = 0, \end{cases} \tag{38}$$

to denote the subdifferential of the non-differential function $|w_{g,u}|$ at $w_{g,u} = 0$. Then, solving for $w_{g,u}^{(t+1)}$, for all u, we obtain the solution to problem (35) as expressed in (36). □

We propose the Algorithm 1, which captures the ensemble learning with FTRL algorithm presented above.

Algorithm 1 Ensemble algorithm with Per-Coordinate FTRL-Proximal method

Input: Parameters $\alpha, \beta, \lambda_1, \lambda_2, \boldsymbol{\vartheta}_g^\top = \boldsymbol{n}_g^\top = \boldsymbol{0} \in \mathbb{R}^{|\mathcal{U}|}$
Output: Ensemble weight vector $\boldsymbol{w}_g^{(T+1)}$
for $t = m+1, ..., T$ **do**
  Receive data $\hat{\boldsymbol{x}}_g^{(t)}$;
  For $u \in \mathcal{U}$, compute $w_{g,u}^{(t)}$ based on (36);
  Predict $\hat{y}_g^{(t)} = (\boldsymbol{w}_g^{(t)})^\top \hat{\boldsymbol{x}}_g^{(t)}$;
  Observe the true value $y_g^{(t)}$;
  for $u \in \mathcal{U}$ **do**
    $h_{g,u}^{(t)} = 2((\boldsymbol{w}_g^{(t)})^\top \hat{\boldsymbol{x}}_g^{(t)} - y_g^{(t)})\hat{x}_{g,u}^{(t)}$;
    $\sigma_{g,u}^{(t)} = \frac{1}{\alpha}\sqrt{n_{g,u}^{(t-1)} + (h_{g,u}^{(t)})^2} - \sqrt{n_{g,u}^{(t-1)}}$;
    $\vartheta_{g,u}^{(t)} = \vartheta_{g,u}^{(t-1)} + h_{g,u}^{(t)} - \sigma_{g,u}^{(t)} w_{g,u}^{(t)}$;
    $n_{g,u}^{(t)} = n_{g,u}^{(t-1)} + (h_{g,u}^{(t)})^2$;
  end for
end for

On the other hand, by the Corollary 1 in [64], the upper bound for the regret of the proposed ensemble learning can be presented as in the following theorem:

Theorem 2. *Let $C \subseteq \mathbb{R}^{|\mathcal{U}|}$ be the convex set of problem (31). Define $D = \max_{\boldsymbol{w},\boldsymbol{w}'\in C} \|\boldsymbol{w} - \boldsymbol{w}'\|$. Let $\{l^{(t)}\}_{t=1}^{K}$ be a sequence of convex loss functions such that $\|\nabla l^{(t)}(x)\| \le G$, for all $t \in \{1, \ldots, K\}$ and all $x \in C$. Then, the upper bound on the regret is as follows*

$$Regret \le DG\sqrt{2K} \tag{39}$$

The bound on the regret guarantees that algorithm 1 asymptotically converges to the optimal solution, i.e., $\lim_{K\to\infty} \frac{\text{Regret}}{K} = 0$. In other words, over the long-term average, the accumulative loss that the player suffers under the algorithm $\mathcal{A}$ asymptotically approaches the loss of the best weight vector in hindsight.

D. Edge Proactive Caching Algorithm

In this section, we present the caching algorithm, as described in Algorithm 2. The MECS sets a monitoring $\mathcal{T} = \{T+1, ..., T+M\}$, i.e., the duration for evaluating the prediction algorithm. At each time slot $t \in \mathcal{T}$, the MECS records requested movies up to time slot $t-1$ and extracts genres from these movies. Movies with the same genre, $g \in \mathcal{G}$, are stored and sorted in ascending order of view count in a list denoted as $\mathcal{L}_g$. Training and test data for the first level learner, $\left(\boldsymbol{X}_{g,u}^{LSTM}, \boldsymbol{y}_{g,u}^{LSTM}\right)$, $\left(\tilde{\boldsymbol{X}}_{g,u}^{LSTM}, \tilde{\boldsymbol{y}}_{g,u}^{LSTM}\right)$, are archived using the data preprocessing procedure in Section IV-B. The train data $\left(\boldsymbol{X}_{g,u}^{LSTM}, \boldsymbol{y}_{g,u}^{LSTM}\right)$ is used for training first-level learners and the test data $\left(\tilde{\boldsymbol{X}}_{g,u}^{LSTM}, \tilde{\boldsymbol{y}}_{g,u}^{LSTM}\right)$ is used for generating training data for meta learners, where the predictions from the first-level learners are considered as the inputs. Specifically, the training phase includes two parts: first-level learner training and meta-leaner training. In the first part, for user group u, for each movie genre g, one LSTM-based first-level learner $\phi_{g,u}$ is trained using its associated training set $\boldsymbol{X}_{g,u}^{LSTM}$. In the second part, one meta-learner ψ_g is trained for each movie genre g using the prediction from the first-level learner $\phi_{g,u}$

Algorithm 2 Proposed Edge Caching Algorithm

Input: Demand matrix $\boldsymbol{A}_u^{(1:T)}$ in (1)
Output: Overall cache hit
for $t = T+1, \ldots, T+M$ **do**
  % Generating list of popular movies in the previous time slots
  Record requested movies up to time slot $t-1$, extract associated genres in $\mathcal{G}$;
  Store movies with the same genre $g \in \mathcal{G}$ in list $\mathcal{L}_g$;
  Compute the total number of views for each movie in the list, and then sort them in ascending order of view count;

  % Training and Predicting Phase
  % First-level learner training
  Obtain $\left(\boldsymbol{X}_{g,u}^{LSTM}, \boldsymbol{y}_{g,u}^{LSTM}\right), \left(\tilde{\boldsymbol{X}}_{g,u}^{LSTM}, \tilde{\boldsymbol{y}}_{g,u}^{LSTM}\right)$ from $\boldsymbol{A}_u^{(1:t-1)}$ in Section IV-B;
  for $u \in \mathcal{U}$ **do**
    for $g \in \mathcal{G}$ **do**
      Train first-level learner using LSTM model in (19):
      $$\phi_{g,u} = \mathfrak{L}_u\left(\boldsymbol{X}_{g,u}^{LSTM}, \boldsymbol{y}_{g,u}^{LSTM}\right);$$
    end for
  end for

  % Meta-learner training
  for $g \in \mathcal{G}$ **do**
    $D_g = \emptyset$;
    for $u \in \mathcal{U}$ **do**
      $\hat{\boldsymbol{x}}_{g,u} = \phi_{g,u}(\tilde{\boldsymbol{X}}_{g,u}^{LSTM})$;
      $D_g = D_g \cup \left(\left[\hat{\boldsymbol{x}}_{g,u}\right]_{u\in\mathcal{U}}, \boldsymbol{y}_g^{meta}\right)$, where $\boldsymbol{y}_g^{meta}$ is defined in (21);
    end for
    Train meta-learner ψ_g by Algorithm 1 for stacking regression model to the new dataset D_g: $\psi_g = \mathfrak{L}(D_g)$, where $\mathfrak{L}$ denotes the regression model in (23).
  end for
  Predict $\hat{y}_g^{(t)} = \psi_g\left(\left[\hat{x}_{g,u}^{(t)}\right]_{u\in\mathcal{U}}\right)$, for $g \in \mathcal{G}$;

  % Caching Phase
  Calculate $v_g = \frac{\hat{y}_g^{(t)}}{\sum_{g\in\mathcal{G}} \hat{y}_g^{(t)}}, \quad \forall g \in \mathcal{G}$;
  for $g \in \mathcal{G}$ **do**
    Calculate $N_g = C v_g, \ \forall g \in \mathcal{G}$;
    Cache a set of movies of length N_g from the top of list $\mathcal{L}_g$;
  end for
  Observe the user requests, compute cache hit and cache miss;
  Update $\boldsymbol{A}_u^{(1:t)}$, for $u \in \mathcal{U}$.
end for

on the test data $\tilde{\boldsymbol{X}}_{g,u}^{LSTM}$ as an input feature, and the average (actual) demand for movie genre g over user groups as a target output. As a result of the training phase, the trained meta-learner ψ_g predicts the user demands for different move genres, $\hat{y}_g^{(t)} = \psi_g\left(\left[\hat{x}_{g,u}^{(t)}\right]_{u\in\mathcal{U}}\right)$. Then, the percentage of user demands for a specific genre over all other movie genres is calculated according to $\hat{y}_g^{(t)}$ as follows

$$v_g = \frac{\hat{y}_g^{(t)}}{\sum_{g\in\mathcal{G}} \hat{y}_g^{(t)}}, \quad \forall g \in \mathcal{G}. \tag{40}$$

If the cache size is C, the number of movies to be cached for each genre g can be calculated as $N_g = C v_g$. Then, N_g movies from the top of the list $\mathcal{L}_g$ are extracted and stored in a local cache of MECS. Finally, the user request demand matrix $\boldsymbol{A}_u^{(t)}$ for $u \in \mathcal{U}$ is updated after new user requests are observed.

V. LSTM-Ensemble-Learning based Edge Caching Architecture

This section describes the caching architecture in edge computing, and how the proposed learning-based caching algorithm works with MECS. They are shown in Fig. 5.

A. Cloud-Edge Computing Architecture

The proposed overall MEC framework consists of three layers: *User& IoT-device layer*, *MEC layer*, and *Cloud layer*.

1) User & IoT-device layer: This layer includes IoT devices such as cameras/sensors deployed in the smart home and all possible cache service users: 5G/6G radio users, Wi-Fi users, and 3rd-party wired and wireless users. By using CNN-based deep learning model as decribed in [45], [46], users' information such as age, gender, etc. can be obtained from facial images captured by the cameras in the house. This information and related additional information (e.g., occupation, etc) are compressed and sent to the edge node. Through the *User Interface*, users send their service requests to a MECS and receive the services from the *Local Cache*.

2) MEC layer: This layer has BSs (e.g., edge node) that are equipped with MECSs. This layer plays the role of data analysis, AI/ML processing, policy management, and storage. Data analysis, AI/ML processing, and policy management can be core components of network data analytic function (NWDAF) at MEC, which is a big data processing module in 5G/6G networks [65].

- *Data Analyzing:* Its main components are *Content Database (DB)*, *Request DB*, and *User DB* modules. The content-related information (e.g., contents list, cloud server id, etc.), user-related information (e.g., user ID, age, gender, occupation, etc.), and user's request information (e.g., watching time, movie name, movie genre, etc.) are saved in the *Content DB*, *Request DB*, and *User DB*, respectively. The collected data is classified and analyzed according to the purpose of each DB.
- *AI/ML Processing:* Its main components are *Preprocessing*, *Learning DB*, and *Learning Kernel* modules. First, the *Preprocessing* is responsible for data preparation for the generation of training and testing data. According to the purpose of AI/ML analysis, *Preprocessing* forms different groups or clusters based on the collected user information such as age, gender, occupation, etc. Once user groups are generated, time-series data for each group is extracted. The *Preprocessing* decomposes, normalizes, and deseasonalizes the time-series data. This process was mentioned in Section III. Second, the *Learning DB* is responsible for creating, managing and storing various training data and test data for the *Learning Kernel*. Third, the *Learning Kernel* is responsible for predicting the popularity of movie genres across groups of users. The proposed LSTM and ensemble learning models learn complex patterns of user preferences in various user groups. Although content popularity prediction is a computation-intensive tasks, it can be efficiently handled with the strong computing capability of the MECS. This is described in detail in Section IV.

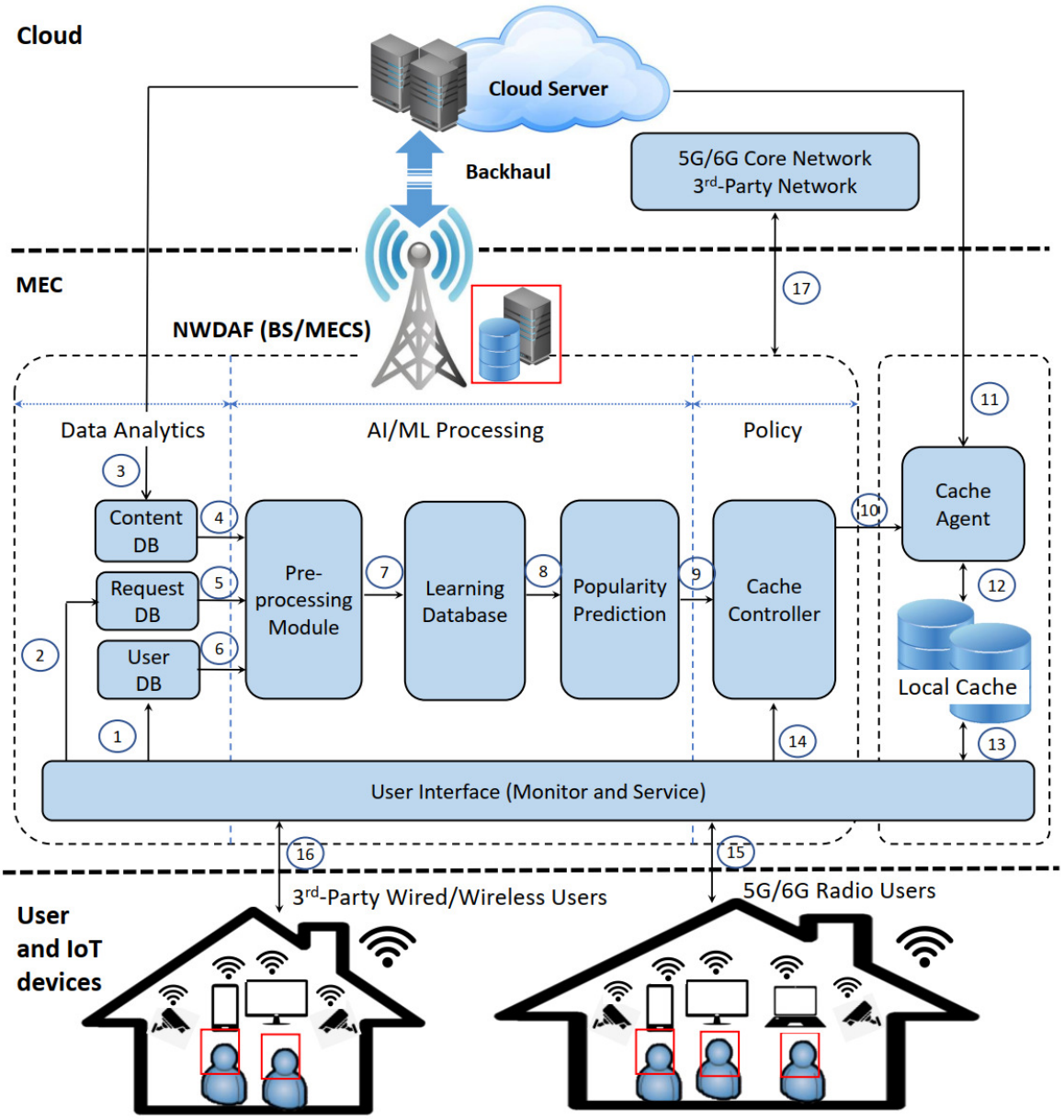


Fig. 5. LSTM-Ensemble-based IoT-MEC caching architecture.

- *Policy:* The main component is a *Cache Controller* module, which is responsible for determining whether or not to cache contents by the popularity prediction. Caching usually consists of *content placement phase* and *content delivery phase*. This paper focuses on the *content placement phase*.
- *Storage:* The main component is a *Cache Agent* module, which is responsible for replacing contents interacting with *Cloud Server*, and *Local Cache*. According to the caching policy based on LSTM-ensemble-based learning at the MECS, the contents with high predicted popularity will be pre-cached before user requests arrive. If the requested content already exists in the *Local Cache* of the MECS, then it can be served directly without retrieving from the *Cloud Server*. The details of the caching scheme are provided in Subsection IV-D.

3) Cloud layer: This layer has content cloud servers that store all contents, which can be requested from the users. This layer also includes 5G/6G core networks and 3rd-party networks.

B. Main Interfaces and Procedures

The activities of the main modules in the proposed proactive caching framework are summarized in Fig. 5. ①-③: The user information, content request information, content server information are periodically or aperiodically monitored and stored in the *User DB*, *Request DB*, and the *Content DB*, respectively. ④-⑥: After each time slot, the *Content DB*, *Request DB*, and *User DB* send the collected information to the *Preprocessing Module* for AI/ML processing. ⑦: The *Preprocessing* delivers its preprocessed data to the *Learning*

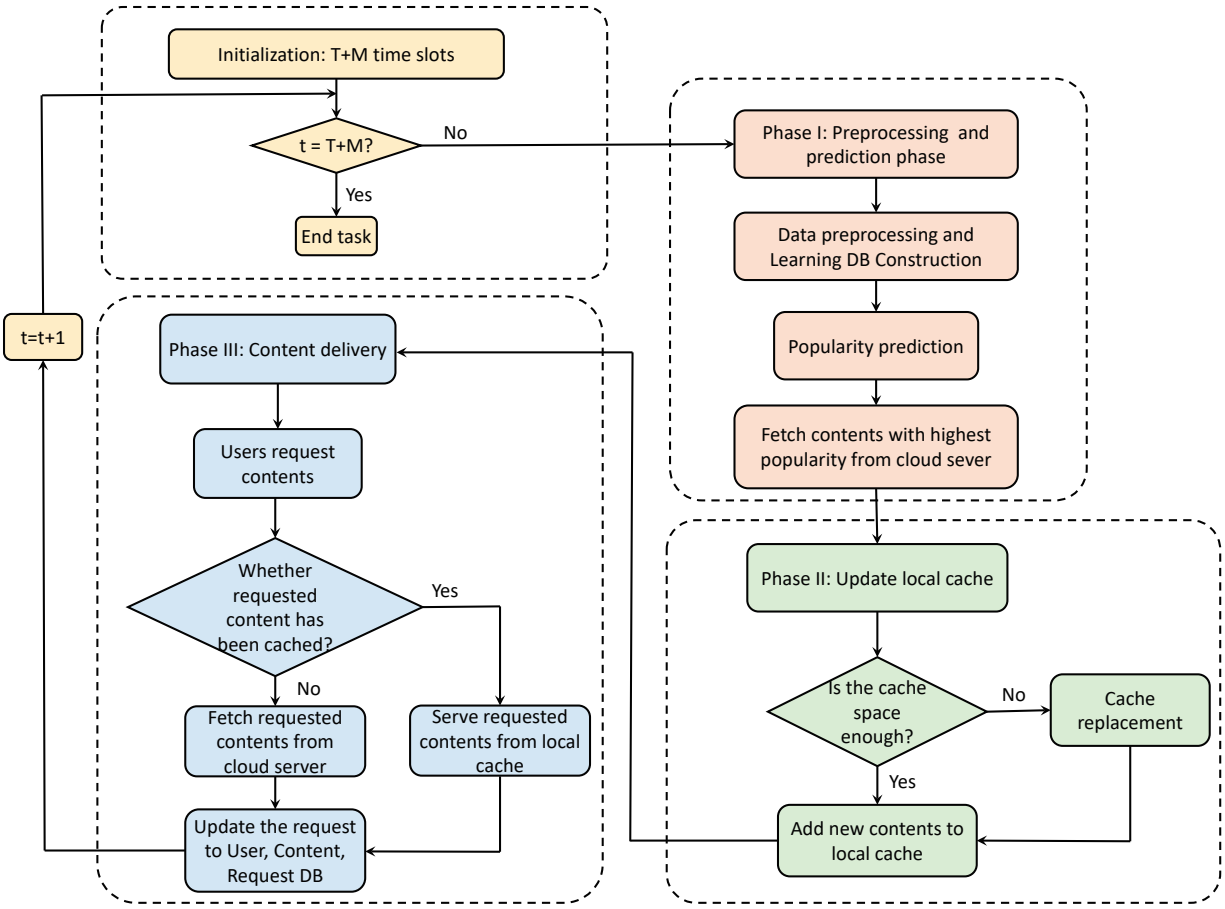


Fig. 6. LSTM-Ensemble-based IoT-MEC caching architecture flow chart.

DB. ⑧: Training or testing data is forwarded to the *Learning Kernel*. ⑨: The predicted popularity information of each content genre is delivered to the *Cache Controller*. ⑩: The *Cache Controller* decides which content should be cached based on the popularity of the genres and the ranks in the popular content list. Then, it delivers its control to the *Cache Agent*. ⑪: If the requested content is not in the *Local Cache*, the *Cache Agent* retrieves the contents from the *Cloud Server* and stores it in the *Local Cache*. ⑫: The *Cache Agent* stores or deletes the contents in/from the *Local Cache*. ⑬: The stored content is delivered to users through the *User Interface* module. ⑭: The users demand their contents to the *Cache Controller* through the *User Interface* module. Combining with Algorithm 2, overall flows in the proposed proactive caching framework can be summarized as a flow chart in Fig. 6.

VI. Performance Evaluation

A. DataSets and Hyperparameter Setting

1) Dataset Analysis: To evaluate the proposed algorithm, we conducted our experiments on two real-world datasets including the MovieLens 100K [49] and MovieLens 1M [50]. They are popular movie datasets released by the GroupLens research, for movie recommendations [66]. The MovieLens 100K dataset consists of $100,000$ ratings from 943 users on 1682 movies, collected from 1997 to 1998. The MovieLens 1M dataset consists of $1,000,209$ ratings from 6040 users on approximately 3900 movies, collected from 2000 to 2003. The datasets include anonymous user ID, movie ID, movie genre, and timestamp. Additionally, they include demographic information of users, such as gender, age, and occupation. This additional information aids the segregation of users into different user groups for user preference prediction. We also assumed that the caching entity at MECS updates its cache content daily. Content can be refreshed during off-peak hours (such as at night) in a day without affecting normal network activities. To simulate the movie requests, we assumed that movies on a day (corresponding to a timestamp of the dataset) are those requested by the users.

Remark 3. In our experiments, the main reason that we chose these datasets [49], [50] is that they include various users' demographic information such as age, gender, and occupation. Also, as stable benchmark datasets, they have been widely used in recent studies [36], [45], [67].

2) Hyperparameter Setting: For model hyperparameter tuning, it is important to choose the appropriate hyperparameters for the LSTM model. As shown in Table II, the hyperparameters were chosen automatically based on the Bayesian

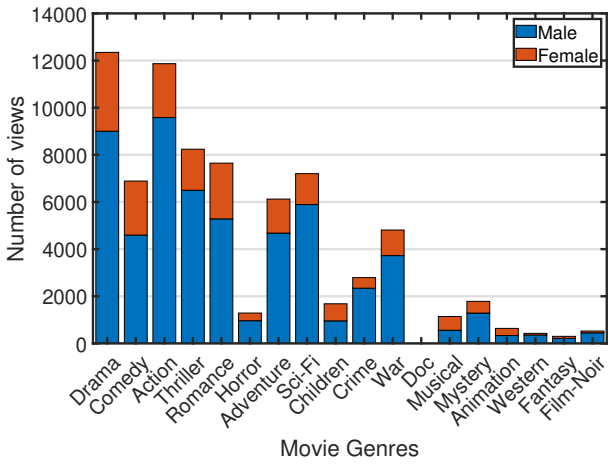


Fig. 7. Movie genre distribution of gender group in the 100K dataset.

optimization method for hyperparameter tuning [68]. The hyperparameter values were between the minimum and maximum values.

TABLE II
HYPERPARAMETER SETTINGS

Hyperparameter	Minimum value	Maximum value
Batch size	10	50
Number of epochs	200	250
Number of hidden layers	1	1
Number of neurons	50	100
Dropout	0.0001	0.1
L2-regularization parameter	5×10^{-4}	5×10^{-3}

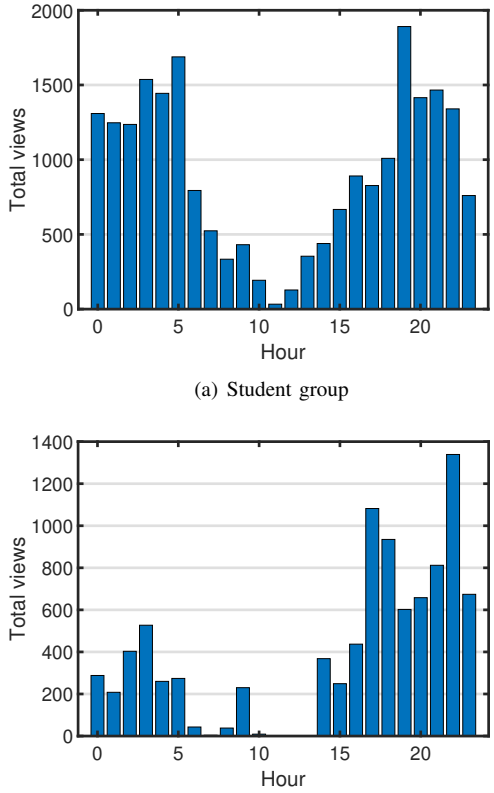
B. User Group's Preference

Fig. 7 illustrates the preference on different movie genres in two gender groups. It can be seen that male viewers prefer action and science fiction movies, while female viewers favor drama and romance movies. In addition, we can see that male viewers watch movies more often than female viewers. The aforementioned diversity in the preference of movie genres in these user groups varies the popularity of a movie across user groups.

Moreover, the popularity of a movie changes over time because different user groups watch it at different times during a day or week. To demonstrate this fact, we investigated the educator group and the student group based on the MovieLens 100K dataset [49]. In Fig. 8(a), it can be observed that the watching time of the student group tends to be in the early morning and evening, while the watching time of the educator group in Fig. 8(b) tends toward the evening. Here, it can be seen that different groups tend to have different preferences in watching time. Therefore, the patterns in user preferences can be leveraged to improve the prediction performance of the neural networks.

C. Prediction Evaluation

To evaluate the accuracy of the proposed prediction method, we can apply commonly-used metrics, such as root mean



(a) Student group

(b) Educator group

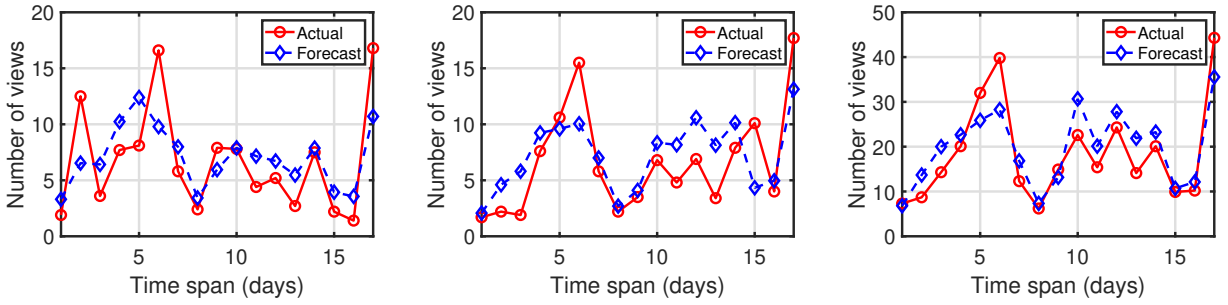
Fig. 8. Total of views in hour basis subjected to the student group and educator group in the 100K dataset.

squared error (RMSE), mean absolute error (MAE), mean absolute percentage error (MAPE), symmetric mean absolute percentage error (sMAPE), or mean absolute scaled error (MASE) [69]. However, the first two metrics are scale-dependent measures, which depend on the scale of the data; the next two pose infinite or undefined problems because they involve division by a value equal or close to zero. For our datasets, there are numerous zero observations; thus, we did not use these metrics for our evaluation. The final metric, MASE, is one of the best candidates as it overcomes the aforementioned issues.

$$MASE = \frac{\frac{1}{n}\sum_{t=1}^{n}|F_t - Y_t|}{\frac{1}{m-S}\sum_{t=S+1}^{m}|Y_{t-S} - Y_t|}, \tag{41}$$

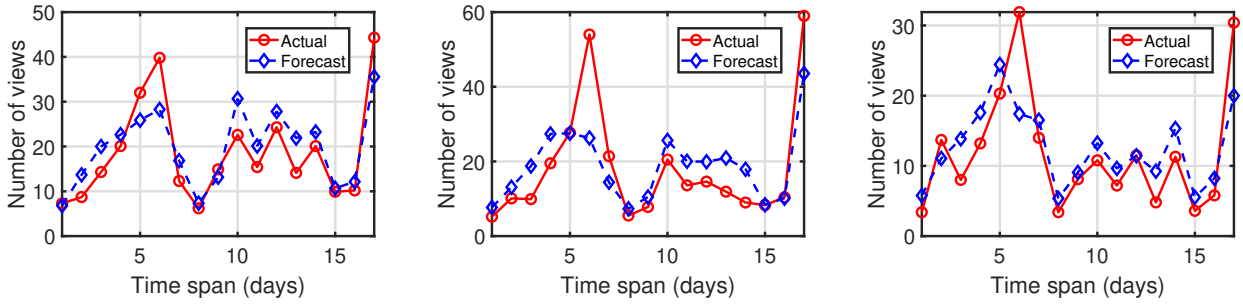
where Y_t represents actual observation at time t, and F_t refers to the prediction at time t. The parameters n and m are the number of predicted values in the test set and the number of observations in the training set, respectively. The seasonal period of a time series is represented as S. In this experiment, S indicates weekly the seasonal period, i.e., $S = 7$.

MASE is a scale-independent metric that measures the possible improvement by the proposed prediction method compared to a benchmark prediction method. Here, the benchmark prediction method is the seasonal naïve method [51], in which the prediction value is equal to the last observation, i.e., $F_t = Y_{t-S}$, where S is a seasonal period. MASE is the mean absolute error between the predicted value and actual observation in the



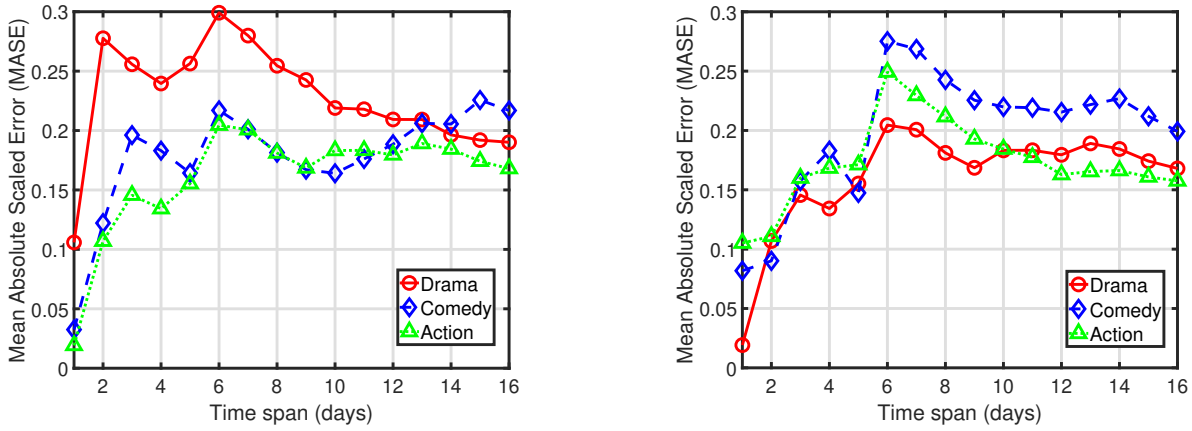
(a) Drama (b) Comedy (c) Action

Fig. 9. Actual and predicted values for 17 days for three movie genres with the highest view counts over user population (Movielens 100K): (a) Drama, (b) Comedy, and (c) Action.



(a) Drama (b) Comedy (c) Action

Fig. 10. Actual and predicted values for 17 days for three movie genres with the highest view counts over user population (Movielens 1M): (a) Drama, (b) Comedy, and (c) Action.



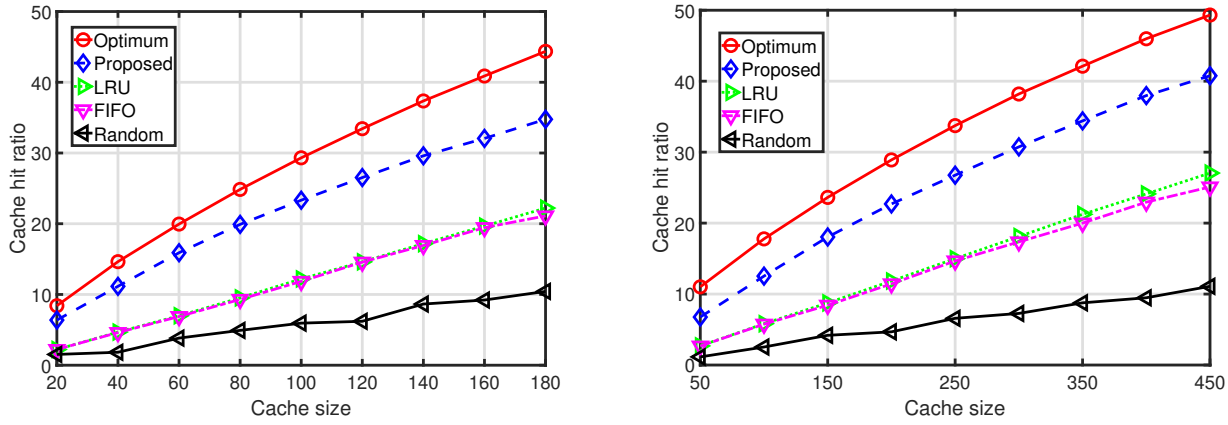
(a) MovieLens 100K (b) MovieLens 1M

Fig. 11. Cumulative Mean Absolute Scaled Error (MASE) for three movie genres (drama, comedy, action).

test set, scaled by the mean absolute error between seasonal-naïve based predicted values and observations in the training set. When MASE < 1, the proposed method, on average, is better than the seasonal-naïve-based method, and when MASE > 1, the proposed method is worse than the seasonal-naïve-based method [69].

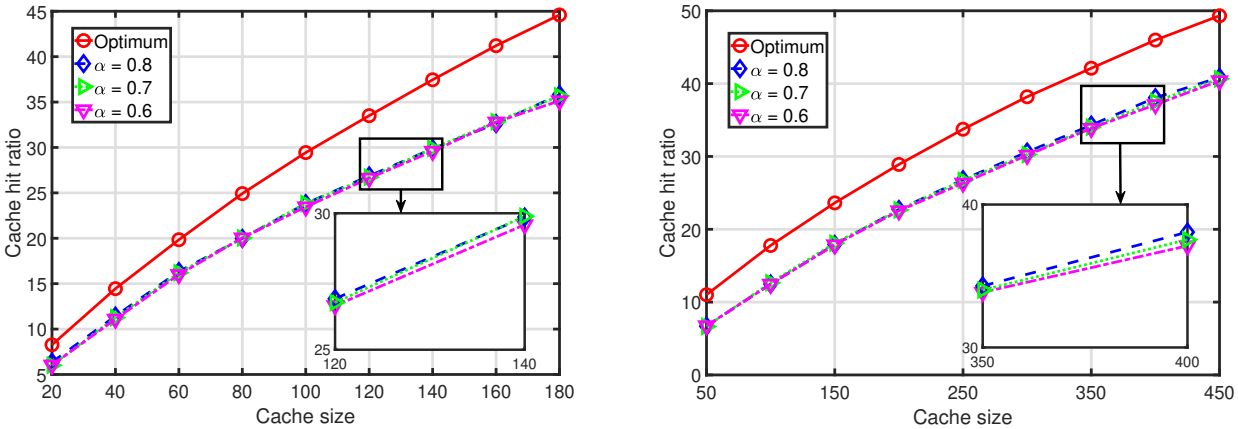
In our experiment, we applied the LSTM neural network for making a prediction one day ahead, which uses seven days' observation history, i.e., the size of the input window equals 7, and the size of the output window equals 1. In each user group, the prediction model was performed over the time series of each movie genre. In both MovieLens 100K and 1M datasets, within the T time slots from the beginning, $\left(\boldsymbol{X}_{g,u}^{LSTM}, \boldsymbol{y}_{g,u}^{LSTM}\right)$ accounts for a proportion of α of the data points (see Eq. (18)), and $\left(\tilde{\boldsymbol{X}}_{g,u}^{LSTM}, \tilde{\boldsymbol{y}}_{g,u}^{LSTM}\right)$ accounts for the remaining proportion of $(1-\alpha)$ of the data points, where $\alpha \in \{0.6, 0.7, 0.8\}$. Figs. 9-11, and 14 illustrate experimental results with $\alpha = 0.8$. In addition, Fig. 13 was added to illustrate the caching performance with different proportions of α.

Figs. 9 and 10 show the predicted user demand in comparison to the actual user demand for 17 days for both datasets.



(a) MovieLens 100K (b) MovieLens 1M

Fig. 12. Cache ratio vs cache size with different datasets: (a) MovieLens 100K, (b) MovieLens 1M.



(a) MovieLens 100K (b) MovieLens 1M

Fig. 13. Cache ratio vs cache size with different proportions of training set: (a) MovieLens 100K, (b) MovieLens 1M.

We show the results of the three movie genres that attract much attention from the viewers: drama, comedy, and action. It can be observed that the error between the predicted and actual demands for different movie genres is relatively small in Fig. 9 and Fig. 10. This reveals the potential advantages of our edge caching algorithm with respect to the change in user preference over time.

In Fig. 11, we show the cumulative MASE error of our proposed prediction algorithm for the above three movie genres to each date over 17 days. We can observe that the error values are less than 1, i.e., the proposed method is better than naïve seasonal prediction on both datasets.

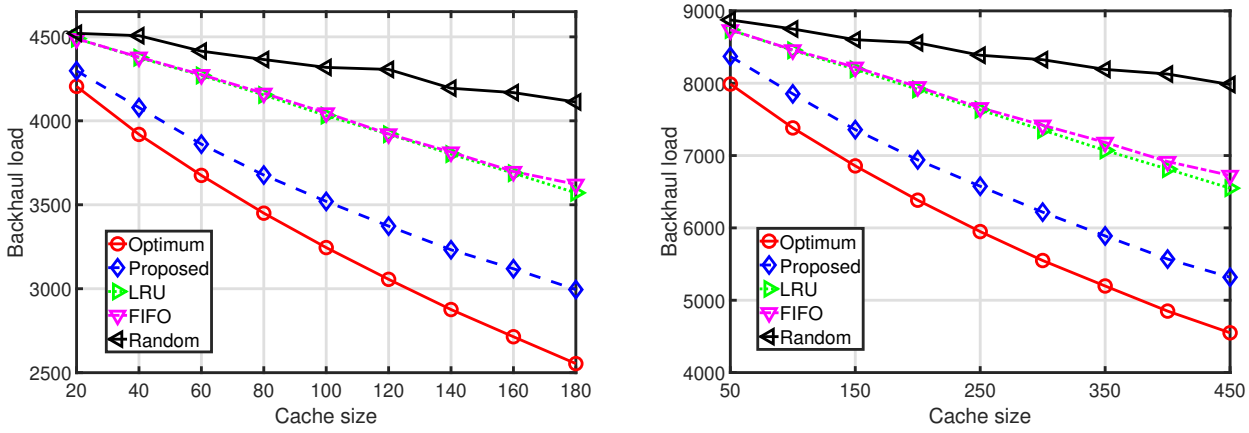
D. Caching Performance Results

To evaluate the performance of the proposed caching algorithm, we compared it with four prevalent schemes.

- *Optimum:* The optimal policy with perfect prior information about the popularity of movies provides the best cache decision.
- *Least recently used (LRU) [70]:* The reactive caching policy fetches a movie from the content server and caches it in the event of a cache miss. In case the cache is full, LRU replaces the movie that has been least recently requested to make room for newly requested movies.
- *First-in-first-out (FIFO) [71]:* The reactive caching policy fetches movies from the content server and caches it in the event of a cache miss. In case the cache is full, FIFO replaces the movie that has been stored for the longest time in the cache to make room for newly requested movies.
- *Random:* This caching policy caches movies randomly.

In Figs. 12(a) and (b), we show the overall cache hit ratio of our proposed policy with different cache sizes in 100K and 1M datasets, respectively. In Fig. 12(a), the cache size varies from 20 to 180 movies out of the total 1682 movies, which corresponds to 1.18% - 10.7% of the total number of movies. The cache size in Fig. 12(b) varies from 50 to 450 movies out of the total 3952 movies, which corresponds to 1.26% - 11.39% of the total number of movies.

It can be observed in both the figures that the cache hit ratio of all the policies increases as the cache size increases and the proposed caching policy shows better performance than other policies. In Fig. 12(a), the proposed caching policy achieves approximately 14%, 15%, and 25% higher cache hit ratio



(a) MovieLens 100K (b) MovieLens 1M

Fig. 14. Backhaul load vs cache size with different datasets: (a) MovieLens 100K, (b) MovieLens 1M.

than LRU, FIFO, and Random, respectively. In Fig. 12(b), the proposed caching policy can archive approximately 15%, 16%, and 30% higher cache hit ratio than those with LRU, FIFO, and Random, respectively. This is because the proposed policy can predict user demand and popularity for movie genres using information from long-term history and decide to cache those movies that meet most of the users' preferences. However, LRU, FIFO, and Random do not consider the popularity of movies or predict the future popularity of movies. Moreover, compared to the optimal policy, the proposed method achieves near-optimal cache hit ratio of within 11% and 9% for the 100K and 1M datasets, respectively. In addition, Figs. 13(a), (b) were sketched to illustrate the influence of these proportions on the cache hit ratio on both datasets. It can be seen that the cache hit ratio with higher proportion of $\alpha = 0.8$ is slightly higher than that of $\alpha = 0.6$ and $\alpha = 0.7$. However, in general, excessive use of training data can lead to longer training time or overfitting problems.

If a movie is stored in a cached entity, it can be served to the users directly without any traffic on the backhaul link. Otherwise, the movie has to be retrieved from the content server, which adds to the backhaul load. In Fig. 14, we show the comparison between the backhaul load of our proposed caching algorithm and that of the benchmark algorithms in terms of varying cache sizes based on the two datasets. It can be observed that when the cache sizes increases, the backhaul load decreases for all algorithms. In Fig. 14(a), the backhaul load of the proposed algorithm is approximately 14%, 15%, and 30% lower than those of LRU, FIFO, and Random algorithm, respectively. In Fig. 14(b), the backhaul load of the proposed algorithm is approximately 14%, 15%, and 27% lower than those of LRU, FIFO, and Random algorithm, respectively. Moreover, our algorithm can approach near-optimal backhaul-load reduction within 12%, and 10% for the 100K and 1M datasets, respectively.

VII. Conclusions

In this paper, we proposed an efficient content caching policy in edge networks that implements dynamic prediction. It exploits a hierarchical deep learning architecture: LSTM-based local learning and ensemble-based meta-learning. First, as a local learning model, we employed the LSTM method using STL-based preprocessing. It identified real-time content preferences in each demographic user group. Second, as a meta-learning model, we employed a regression-based ensemble method. For optimal ensemble learning, we developed an online convex optimization approach that provides sublinear 'regret' performance. It effectively orchestrated the obtained multiple demographic user preferences into a unified caching strategy. Extensive experiments were conducted on the popular MovieLens dataset to verify the caching performance of the proposed algorithm against various benchmark algorithms, including an optimal caching algorithm with perfect prior knowledge of content popularity. The proposed control provides up to a 30% higher cache hit ratio than conventional representative algorithms. Moreover, the proposed control has a near-optimal cache hit ratio within approximately 9% of the optimal caching scheme. The proposed learning and caching control algorithms can be implemented as a core function of the 5G/6G standard. As part of our future work, we will study advanced caching algorithms that consider the popularity of cold-start content [72] and on-device coded caching algorithms that leverage distributed learning techniques in a dynamically changing network.

References

[1] "Ericsson mobility report," Report, Ericsson, Nov. 2020. [Online]. Available: https://www.ericsson.com/en/mobility-report/dataforecasts

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1657–1681, 2017.

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

[5] J. Yao, T. Han, and N. Ansari, "On mobile edge caching," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2525–2553, 2019.

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[7] A. S. Gomes, B. Sousa, D. Palma, V. Fonseca, Z. Zhao, E. Monteiro, T. Braun, P. Simoes, and L. Cordeiro, "Edge caching with mobility prediction in virtualized lte mobile networks," *Future Generation Computer Systems*, vol. 70, pp. 148–162, 2017.

[8] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," *IEEE Communications Magazine*, vol. 51, no. 4, pp. 142–149, 2013.

[9] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5g wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.

[10] Q. Li, L. Changlong, B. Cao, and Q. Zhang, "Caching resource management of mobile edge network based on stackelberg game," *Digital Communications and Networks*, vol. 5, 10 2018.

[11] S. Safavat, N. Sapavath, and D. B. Rawat, "Recent advances in mobile edge computing and content caching," *Digital Communications and Networks*, vol. 6, 09 2019.

[12] Y.-L. Chen and C.-L. Chang, "Early prediction of the future popularity of uploaded videos," *Expert Systems with Applications*, vol. 133, pp. 59–74, 2019.

[13] R. Fares, B. Romoser, Z. Zong, M. Nijim, and X. Qin, "Performance evaluation of traditional caching policies on a large system with petabytes of data," in *Proc. IEEE Seventh International Conference on Networking, Architecture, and Storage*, 2012, pp. 227–234.

[14] Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas, "Federated learning based proactive content caching in edge computing," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[15] B. Blaszczyszyn and A. Giovanidis, "Optimal geographic caching in cellular networks," in *Proc. IEEE international conference on communications (ICC)*, 2015, pp. 3358–3363.

[16] M. Ji, G. Caire, and A. F. Molisch, "Wireless device-to-device caching networks: Basic principles and system performance," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 1, pp. 176–189, 2015.

[17] S. H. Chae, J. Y. Ryu, T. Q. Quek, and W. Choi, "Cooperative transmission via caching helpers," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[18] J. Song, H. Song, and W. Choi, "Optimal caching placement of caching system with helpers," in *Proc. IEEE International Conference on Communications (ICC)*, 2015, pp. 1825–1830.

[19] H. J. Kang, K. Y. Park, K. Cho, and C. G. Kang, "Mobile caching policies for device-to-device (d2d) content delivery networking," in *Proc. IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, 2014, pp. 299–304.

[20] S. H. Chae and W. Choi, "Caching placement in stochastic wireless caching helper networks: Channel selection diversity via caching," *IEEE Transactions on Wireless Communications*, vol. 15, no. 10, pp. 6626–6637, 2016.

[21] D. Malak, M. Al-Shalash, and J. G. Andrews, "Optimizing content caching to maximize the density of successful receptions in device-to-device networking," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4365–4380, 2016.

[22] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[23] Z. Zheng, L. Song, Z. Han, G. Y. Li, and H. V. Poor, "A stackelberg game approach to proactive caching in large-scale mobile edge networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5198–5211, 2018.

[24] K. Hamidouche, W. Saad, and M. Debbah, "Many-to-many matching games for proactive social-caching in wireless small cell networks," in *Proc. 12th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2014, pp. 569–574.

[25] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[26] A. C. Güngör and D. Gündüz, "Proactive wireless caching at mobile user devices for energy efficiency," in *Proc. International Symposium on Wireless Communication Systems (ISWCS)*, 2015, pp. 186–190.

[27] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM*, vol. 1, 1999, pp. 126–134.

[28] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *Proc. 7th ACM SIGCOMM*, 2007, pp. 1–14.

[29] B. Bharath, K. G. Nagananda, and H. V. Poor, "A learning-based approach to caching in heterogenous small cell networks," *IEEE Transactions on Communications*, vol. 64, no. 4, pp. 1674–1686, 2016.

[30] K. N. Doan, T. Van Nguyen, T. Q. Quek, and H. Shin, "Content-aware proactive caching for backhaul offloading in cellular network," *IEEE Transactions on Wireless Communications*, vol. 17, no. 5, pp. 3128–3140, 2018.

[31] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. S. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Transactions on Multimedia*, vol. 21, no. 4, pp. 915–929, 2018.

[32] K. Thar, N. H. Tran, T. Z. Oo, and C. S. Hong, "Deepmec: Mobile edge caching using deep learning," *IEEE Access*, vol. 6, pp. 78 260–78 275, 2018.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, "Learning distributed caching strategies in small cell networks," in *Proc. 11th International Symposium on Wireless Communications Systems (ISWCS)*, 2014, pp. 917–921.

[35] M. S. ElBamby, M. Bennis, W. Saad, and M. Latva-Aho, "Content-aware user clustering and caching in wireless small cell networks," in *Proc. 11th International Symposium on Wireless Communications Systems (ISWCS)*. IEEE, 2014, pp. 945–949.

[36] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 2, pp. 1024–1036, 2016.

[37] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Smart caching in wireless small cell networks via contextual multi-armed bandits," in *Proc. IEEE International Conference on Communications (ICC)*, 2016, pp. 1–7.

[38] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[39] N. Nguyen-Thanh, D. Marinca, K. Khawam, S. Martin, and L. Boukhatem, "Multimedia content popularity: Learning and recommending a prediction method," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[40] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: multiaccess edge computing for 5g and internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6722–6747, 2020.

[41] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[42] D. Antonogiorgakis, A. Britzolakis, P. Chatziadam, A. Dimitriadis, S. Gikas, E. Michalodimitrakis, M. Oikonomakis, N. Siganos, E. Tzagkarakis, Y. Nikoloudakis *et al.*, "A view on edge caching applications," *arXiv preprint arXiv:1907.12359*, 2019.

[43] G. Ruggeri, M. Amadeo, C. Campolo, A. Molinaro, and A. Iera, "Caching popular transient iot contents in an sdn-based edge infrastructure," *IEEE Transactions on Network and Service Management*, 2021.

[44] L. Zanzi, F. Cirillo, V. Sciancalepore, F. Giust, X. Costa-Perez, S. Mangiante, and G. Klas, "Evolving multi-access edge computing to support enhanced iot deployments," *IEEE Communications Standards Magazine*, vol. 3, no. 2, pp. 26–34, 2019.

[45] A. Ndikumana, N. H. Tran, D. H. Kim, K. T. Kim, and C. S. Hong, "Deep learning based caching for self-driving cars in multi-access edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 5, pp. 2862–2877, 2021.

[46] A. Dehghan, E. G. Ortiz, G. Shu, and S. Z. Masood, "Dager: Deep age, gender and emotion recognition using convolutional neural network," *arXiv preprint arXiv:1702.04280*, 2017.

[47] X. Zhang, L. Liang, C. Luo, and L. Cheng, "Privacy-preserving incentive mechanisms for mobile crowdsensing," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 47–57, 2018.

[48] Z. Duan, L. Tian, M. Yan, Z. Cai, Q. Han, and G. Yin, "Practical incentive mechanisms for iot-based mobile crowdsensing systems," *IEEE Access*, vol. 5, pp. 20 383–20 392, 2017.

[49] GroupLens, "Movielens 100k dataset." [Online]. Available: https://grouplens.org/datasets/movielens/100k/

[50] GroupLens, "Movielens 1m dataset." [Online]. Available: https://grouplens.org/datasets/movielens/1m/

[51] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[52] R. B. Cleveland *et al.*, "STL: A seasonal-trend decomposition procedure based on loess," *Journal of Official Statistics*, pp. 3–73, 1990.

[53] M. Nelson, T. Hill, W. Remus, and M. O'Connor, "Time series forecasting using neural networks: Should the data be deseasonalized first?" *Journal of Forecasting*, vol. 18, no. 5, pp. 359–367, 1999.

[54] S. Zhang, N. Zhang, X. Fang, P. Yang, and X. S. Shen, "Cost-effective vehicular network planning with cache-enabled green roadside units," in *Proc. IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[55] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[56] S. Smyl and K. Kuber, "Data preprocessing and augmentation for multiple short time series forecasting with recurrent neural networks," in *Proc. 36th International Symposium on Forecasting*, 2016.

[57] K. Bandara, C. Bergmeir, and S. Smyl, "Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach," *Expert Systems with Applications*, vol. 140, p. 112896, 2020.

[58] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. 20th international conference on machine learning (ICML)*, 2003, pp. 928–936.

[59] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.

[60] T. Chen, Q. Ling, Y. Shen, and G. B. Giannakis, "Heterogeneous online learning for "thing-adaptive" fog computing in iot," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4328–4341, 2018.

[61] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin *et al.*, "Ad click prediction: a view from the trenches," in *Proc. 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 1222–1230.

[62] Y. Jiang, M. Ma, M. Bennis, F.-C. Zheng, and X. You, "User preference learning-based edge caching for fog radio access network," *IEEE Transactions on Communications*, vol. 67, no. 2, pp. 1268–1283, 2018.

[63] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[64] H. B. McMahan, "A unified view of regularized dual averaging and mirror descent with implicit updates," *arXiv preprint arXiv:1009.3240*, 2010.

[65] "3GPP TR 23.791: Study of Enablers for Network Automation for 5G," 3GPP Standard, 2019.

[66] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems*, vol. 5, no. 4, pp. 1–19, 2015.

[67] B. Yi, X. Shen, H. Liu, Z. Zhang, W. Zhang, S. Liu, and N. Xiong, "Deep matrix factorization with implicit feedback embedding for recommendation system," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 8, pp. 4591–4601, 2019.

[68] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[69] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[70] H. Ahlehagh and S. Dey, "Video caching in radio access network: Impact on delay and capacity," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, 2012, pp. 2276–2281.

[71] C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the world wide web," *IEEE Transactions on Knowledge and data Engineering*, vol. 11, no. 1, pp. 94–107, 1999.

[72] D. Ralph, Y. Li, G. Wills, and N. Green, "Recommendations from cold starts in big data," *Computing*, January 2020.

The-Vi Nguyen received the B.S. degree in Mathematics from University of Science, Ho Chi Minh City, Viet Nam in 2016, and M.S. degree in Computer Science and Engineering from Chung-Ang University, South Korea in 2021. He is currently pursuing Ph.D. in Big Data at Chung-Ang University, South Korea. His research interests include machine learning, optimization, and their applications in wireless communications.

Nhu-Ngoc Dao received the B.S. degree in electronics and telecommunications from the Posts and Telecommunications Institute of Technology, Hanoi, Vietnam, in 2009, and the M.S. and Ph.D. degrees in computer science from the School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea, in 2016 and 2019, respectively. From 2019 to 2020, he was a Postdoctoral Researcher with the Institute of Computer Science, University of Bern, Switzerland. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Sejong University, Seoul. His research interests include intelligent systems, network softwarization, mobile cloudization, and the Internet of Things.

Van Dat Tuong received the B.S. degree in Mechatronics from Hanoi University of Science and Technology, Vietnam, in 2012, and M.S. degree in Computer Science and Engineering from Chung-Ang University, South Korea, in 2021. From 2012 to 2018, he was a Software Engineer with the Mobile R&D Center, Samsung Electronics Vietnam, Hanoi, Vietnam. From 2018 to 2021, he was a recipient of the Global Korea Scholarship sponsored by the Korean Government. He is currently pursuing his Ph.D. degree in Big Data at Chung-Ang University, South Korea. His research interests include wireless communication, mobile edge computing, reinforcement learning, and Internet of Things.

Wonjong Noh received the B.S., M.S., and Ph.D. degrees from the Department of Electronics Engineering, Korea University, Seoul, South Korea, in 1998, 2000, and 2005, respectively. From 2005 to 2007, he conducted the Postdoctoral Research with Purdue University, West Lafayette, IN, USA, and the University of California at Irvine, Irvine, CA, USA. From 2008 to 2015, he was a Principal Research Engineer with the Samsung Advanced Institute of Technology, Samsung Electronics, South Korea. After that, he worked as an Assistant Professor with the Department of Electronics and Communication Engineering, Gyeonggi University of Science and Technology, South Korea, and since 2019, he has worked as an Associate Professor with the School of Software, Hallym University, South Korea. He received the Government Postdoctoral Fellowship from the Ministry of Information and Communication, South Korea, in 2005. He was also a recipient of the Samsung Best Paper God Award in 2010, the Samsung Patent Bronze Award in 2011, and the Samsung Technology Award in 2013. His current research interests include fundamental analysis and evaluations on machine learning-based 5G and 6G wireless communications and networks.

Sungrae Cho received B.S. and M.S. degrees in Electronics Engineering from Korea University, Seoul, South Korea, in 1992 and 1994, respectively, and Ph.D. degree in Electrical and Computer Engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2002. He is a Professor with the School of Computer Science and Engineering, Chung-Ang University (CAU), Seoul, South Korea. Prior to joining CAU, he was an Assistant Professor with the Department of Computer Sciences, Georgia Southern University, Statesboro, GA, USA, from 2003 to 2006, and a Senior Member of Technical Staff with the Samsung Advanced Institute of Technology (SAIT), Kiheung, South Korea, in 2003. From 1994 to 1996, he was a Research Staff Member with Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. From 2012 to 2013, he held a Visiting Professorship with the National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA. His current research interests include wireless networking, ubiquitous computing, and ICT convergence. He has served as the Organizing Committee Chair for numerous international conferences, such as IEEE SECON, ICOIN, ICTC, ICUFN, TridentCom, and the IEEE MASS, and as a Program Committee Member for conferences such as IEEE ICC, MobiApps, SENSORNETS, and WINSYS. He has been a Subject Editor for IET Electronics Letter since 2018 and an Editor for Ad Hoc Networks Journal (Elsevier) from 2012 to 2017.