Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 5568935, 18 pages
https://doi.org/10.1155/2021/5568935

Research Article

A Novel Cooperative Cache Policy for Wireless Networks

Lincan Li,[1] **Chiew Foong Kwong**,[1] **Qianyu Liu**,[2] **Pushpendu Kar**,[3] **and Saeid Pourroostaei Ardakani**[3]

[1]*Department of Electrical and Electronic Engineering, University of Nottingham Ningbo China, 315100 Ningbo, China*
[2]*International Doctoral Innovation Centre, University of Nottingham Ningbo China, 315100 Ningbo, China*
[3]*School of Computer Science, University of Nottingham Ningbo China, 315100 Ningbo, China*

Correspondence should be addressed to Chiew Foong Kwong; chiew-foong.kwong@nottingham.edu.cn

Received 11 February 2021; Revised 16 July 2021; Accepted 27 July 2021; Published 10 August 2021

Academic Editor: Vishal Sharma

Mobile edge caching is an emerging approach to manage high mobile data traffic in fifth-generation wireless networks that reduces content access latency and offloading data traffic of backhaul links. This paper proposes a novel cooperative caching policy based on long short-term memory (LSTM) neural networks considering the characteristics between the features of the heterogeneous layers and the user moving speed. Specifically, LSTM is applied to predict content popularity. Size-weighted content popularity is utilised to balance the impact of the predicted content popularity and content size. We also consider the moving speeds of mobile users and introduce a two-level caching architecture consisting of several small base stations (SBSs) and macro base stations (MBSs). To avoid content requests of fast-moving users affecting the content popularity distribution of the SBS since fast-moving users frequently handover among SBSs, fast-moving users are served by MBSs no matter which SBS they are in. SBSs serve low-speed users, and SBSs in the same cluster can communicate with one another. The simulation results show that compared to common cache methods, for example, the least frequently used and least recently used methods, our proposed policy is at least 8.9% lower and 6.8% higher in terms of the average content access latency and offloading ratio, respectively.

1. Introduction

Wireless networks are undergoing exponential growth in mobile data traffic due to the massive utilisation of mobile devices and the tendency for high data rates and low-latency applications [1, 2]. Based on [3], the global mobile data traffic has increased sevenfold from 2016 to 2021, which causes the current network capacity to be not enough. Moreover, massive low-latency applications, such as real-time monitoring and intelligent driving, aggravate the network overload [4]. The most common approach is to deploy ultra-dense BSs to increase network capacity and overcome this problem. However, upgrading these infrastructures comes at a high cost for mobile operators [5]. And even if the capacity is expanded by upgrading the infrastructure, the latency cannot meet the requirements of low-latency applications due to the long distance between the users and the remote core network [6].

To meet the increasing demand for data traffic and provide low-latency services, mobile edge caching is introduced in wireless networks [7]. Mobile edge caching deploys popular content at edge nodes close to users, either in base stations (BSs) and/or user terminals (UTs). Users can directly access their requested content from the edge server rather than the remote core network via backhaul links [8], reducing content access latency due to decreased content transmission distances. In addition, data congestion in the backhaul links can be efficiently alleviated as many repeated content requests are avoided in the backhaul links [9, 10].

However, the cache capacity is limited, that only a part of rather than all the contents can be cached. The requests for the uncached contents still need to be satisfied at the core network [11]. Designing an efficient cache policy based on the limited cache capacity is a challenge. To date, the predictive cache scheme is proposed, where the content is precached before the content is requested [12]. Once the cached content

is requested, it can be directly delivered from the cache devices to the corresponding requester. The performance of the predictive cache highly depends on the accurate content popularity [13]. However, it is hard to accurately predict the content popularity when the cache scenario is dynamic.

Besides, the cache-enabled heterogeneous networks (HetNets) have attracted much attention. If the contents are cached at the small base stations (SBSs), the requested content can be delivered from the SBSs directly. Otherwise, the SBS needs to retrieve the requested content from the upper level, e.g., macro base stations (MBSs), and then send it to the corresponding user. The existence of the cache-enabled SBS provides more chances to reduce the content access latency since SBSs are highly close to the users [14]. Additionally, the SBSs can be empowered with collaborative ability. Under this situation, adjacent BSs can communicate with one another. Once the requested content is not cached at the serving SBS, the serving SBS retrieves the content from its nearby SBSs directly [15]. However, the above cache policies in HetNets consider the stationary user while do not consider user mobility. Actually, the users are mobile moving, and the local content popularity varies due to mobile users frequently moving in and out of the coverage area of the SBS. Without considering user mobility can cause the cache policy cannot adapt to the actual scenario [16]. The reason is that the motion of users increases the uncertainty of users arriving at the specific area, which complexes the local content popularity distribution.

To accurately predict the content popularity and make sure our policy can adapt to the actual scenario, this paper proposes a content popularity prediction-based cooperative caching (CPP-CC) policy and jointly considers the heterogeneous layers' features and user mobility. This paper's main contributions are summarised as follows:

(1) LSTM neural networks with time sequence prediction abilities are utilised to predict the time-varying content popularity. As a result, CPP-CC can track variations in the dynamic content popularity to achieve a high cache performance. Compared to the common neural networks in time-series prediction, e.g., recurrent neural network (RNN) and echo state network (ESN) [17], the advantage of LSTM in predicting dynamic content popularity comes from its self-forgetting ability [18]. Specifically, in RNN and ESN, the content requested a long time ago is still used to predict the future content popularity, which may result in the pollution of future content popularity by these outdated content requests. In contrast, the LSTM can forget these outdated content requests to enable the accuracy of future content popularity

(2) A size-weighted content popularity algorithm is applied to balance the effect of the content size and predicted content popularity to increase the cache performance of content popularity-based cache policies

(3) A two-level cache architecture consisting of MBSs and SBSs is proposed. The MBS should only serve fast-moving users in this work, and the SBS serves slow-moving users. As the MBS has a relatively large cache capacity and its cached content does not need to be updated frequently, this configuration ensures that fast-moving users can acquire their desired content from the close MBS and will not affect the content popularity distribution of the SBS. This configuration ensures that slow users can receive their desired content from the SBS with low content access latency

(4) The caching storage of each SBS is divided into two parts, which are the global caching and local caching modules. The local caching module caches popular content evaluated from the local perspective and the global module caches popular content evaluated from the entire cluster's perspective. Global caching modules in the same clusters can share their cached content, which further reduces content access latency

The remainder of this paper is organised as follows: Section 2 introduces the related works. Section 3 introduces a system model and problem formulation. Section 4 presents content caching and delivery for cooperative caching frameworks. Simulation results and analyses are provided in Section 5, and a conclusion is given in Section 6.

2. Related Works

To date, considerable research has been devoted to mobile edge caching. In [19], mobile edge caching at macro base stations (MBSs) was introduced, providing higher cache storage and coverage areas that serve more users. In [20], the small base station (SBS) caching architecture based on the user preference, e.g., the personal content request probability distribution, is proposed to reduce latency. However, popular content cached at noncooperative BSs can result in duplicated content to be cached and waste limited cache capacity resources.

To fully utilise limited cache capacity, cooperation among BSs is applied in cache policies. In [21, 22], a cooperative cache structure to reduce content access latency was introduced, in which adjacent SBSs can communicate with one another. In [23], Ye et al. presented a collaborative cache policy based on BS clustering to reduce content transmission costs. In [7], a predictive cache scheme was applied in the heterogeneous network, where the user's future path is predicted to decide the cache policy for each SBS. In [24], the user mobility was utilised to predict the contact time, which is when the user is in the coverage area of a BS. The cache content placement policy is based on the contact time since the contact time can affect the successful retrieval of the content by the user even if it is cached at the BS. However, unlike our proposed CPP-CC policy, these cooperative cache policies assume that content popularity is static and cannot perform effectively in dynamic cache scenarios with varying content popularity because they cannot track changes in time-varying content popularity.

To address this issue, the predictive cache scheme is applied. In [25], a deep learning-based content popularity

Table 1: The comparison of the related work policies.

Policy category	Weakness	The improvement by CPP-CC policy
(1) Zhao et al. [19] (2) Cheng et al. [20]	These policies do not include BS cooperation, which results in duplicate content being cached and further wasting limited cache capacity.	The SBSs in the CPP-CC policy can exchange information with each other. In particular, the SBSs in the same cluster can share their cached content.
(1) Xu et al. [21] (2) Zhou et al. [22] (3) Ye et al. [23] (4) Poularakis and Tassiulas [7] (5) Somesula et al. [24]	These policies assume that content popularity is static, and therefore they cannot be used for the dynamic cache scenario.	CPP-CC policy utilises LSTM neural network to predict the dynamic content popularity.
(1) Thar et al. [25] (2) Yan et al. [27]	These policies do not consider user mobility.	The MBS should only serve fast-moving users in this work, and the SBS serves slow-moving users.
(1) Mou et al. [26] (2) Rathore et al. [28]	These policies do not consider the relationship between HetNet layer properties and user mobility.	

prediction algorithm was used to reduce the cost of cache networks. The authors of [26] utilised a long short-term memory (LSTM) network to predict the location-related content popularity and minimise latency. In [27], big data analytics were applied to predict content popularity maximise the cache hit ratio, that is, the ratio of content requests hit in cache nodes to the entire content requests. In [28], a collaborative filter was used to predict the content popularity to reduce the pressure of backhaul links. However, unlike our proposed CPP-CC policy, these predictive cache policies have the following problems: (1) reference [25, 27] do not consider that user mobility can change content popularity since content popularity is location-dependent. (2) Although [26, 28] have considered user mobility, their works have not jointly considered the characteristics of different layers in heterogeneous cache networks and user moving speed. For instance, if fast-moving users are served by an SBS whose coverage area is small, fast-moving users will frequently handover among adjacent SBSs. In this case, the content popularity distribution of the SBS frequently varies, and the cached content of the SBSs needs to be updated frequently; hence, content popularity-based cache policies could be inefficient. The differences between the above cache policies and our proposed CPP-CC policy are summarised in Table 1.

3. The System Model and Problem Formulation

This section describes a heterogeneous network, the content model, and the cooperative caching model. We show how to calculate content access latency and offloading ratio to produce a caching problem formulation. Table 2 is the list of notations mainly used in this paper.

3.1. Heterogeneous Networks. In this paper, we consider a heterogeneous network that consists of a core network, Q MBS, W SBSs, and E users with different moving speeds, as shown in Figure 1. MBS, SBS, and User represent the set of Q MBSs, W SBSs, and E users, respectively. Each MBS or SBS is equipped with limited cache resources to store popular content. Each BS is connected to the cache device through a user plane function (UPF). The interface between the BS and UPF is $N3$, and the interface between the UPF and cache device is $N6$. The MBS has a large cache capacity, and the SBS has a relatively small cache capacity. The adjacent SBSs are clustered in the same group, and each SBS in this group can communicate with one another. High-speed users are served by the serving MBS, and low-speed users are served by the serving SBS. The SBS will retrieve the requested content from nearby SBSs or its serving MBS once the content request is missed locally (the content is not locally cached). The MBS will retrieve the requested content from the core network if the content request is missed in the local MBS. We assume that the core network has enough capacity to hold all the contents provided by the content provider.

Table 2: The notations.

Symbol	Description
MBS	Macro base station
SBS	Small base station
BS	Base station
{MBS},{SBS}, {User}	The set of MBSs, SBSs, and users
MBS_q, SBS_w, and User_e	The q^{th} MBS, w^{th} SBS, and e^{th} user
SBS^q	The set of SBSs in the coverage area of MBS_q
$[\text{User}]_q$, $[\text{User}]_w$	The set of users who are served by the q^{th} MBS and w^{th} SBS
$\mathbb{M}$	The connection status matrix
$\mathscr{F}$	Content library
$\mathcal{S}$	The set of the content size
UPF	The user plane function
$N3$	The interface between the radio access network and the UPF
$N6$	The interface between the UPF and the data network
CPP-CC	Our proposed cache policy
UPC	The uncollaborative predicted cache policy
CLFU	The collaborative reactive least frequently used cache policy
CLRU	The collaborative reactive least recently used cache policy

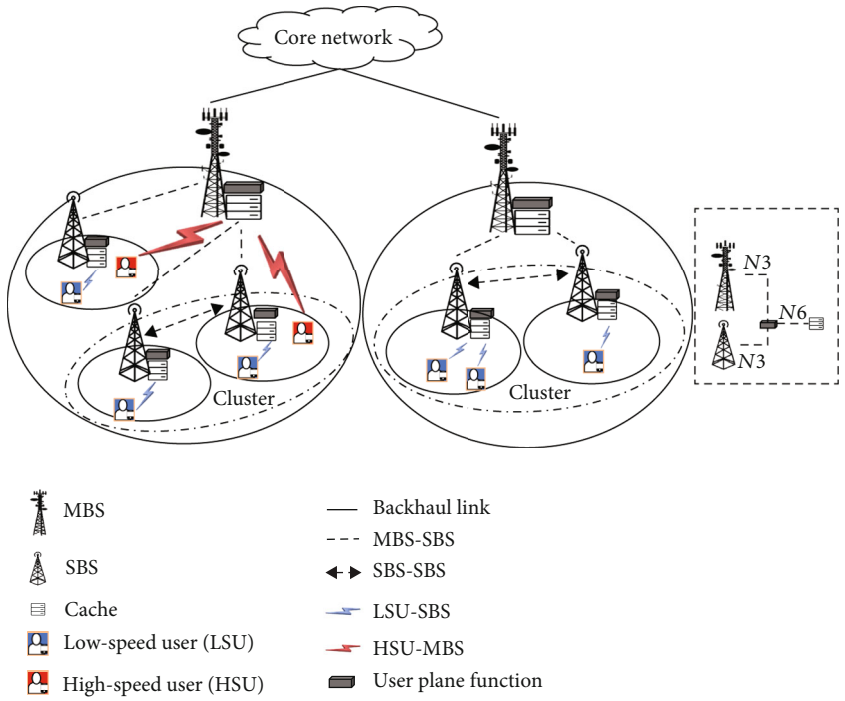


Figure 1: Heterogeneous cache enabled network.

We utilise MBS_q to represent the q^{th} MBS when $1 < q < Q$, SBS_w to represent the w^{th} SBS when $1 < w < W$, and User_e to represent the e^{th} user when $1 < e < E$. The SBS^q is used to denote the set of SBSs in the coverage area of MBS_q. We utilise $[\mathrm{User}]_q$ and $[\mathrm{User}]_w$ to represent the set of users who are served by the q^{th} MBS and w^{th} SBS, respectively. The connection status between E users and W SBSs is modelled as a $E \times W$ matrix $\mathbb{M}$, in which $\mathbb{M}_{ew}$ =1 if User_e is served by SBS_w, and $\mathbb{M}_{ew} = 0$ if not. Similarly, the connection status between E users and Q MBSs is modelled as an $E \times Q$ matrix $\mathbb{N}$.

3.2. Content Model. $\mathscr{F} = \{1, 2, \cdots, F\}$ represents all the F content that is requested by E users, and we assume that the core network has enough capacity to cache all of the F content. $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_F\}$ denotes the amount of F content. $\mathscr{F}^q$ and $\mathscr{F}^w$ denote the cached content in the q^{th} MBS and w^{th} SBS, respectively. C_t^e represents the content requested by User_e in time t whose duration is one hour, and $C^e = [C_1^e, C_2^e, \cdots, C_T^e]^T$ denotes the database with the requested content of User_e. Similarly, C^q and C^w represent the set of the requested content in the q^{th} MBS and w^{th} SBS, respectively. The content requests follow Zipf distribution laws [29], and the probability of r^{th} content c is denoted as P_c and calculated as follows:

$$P_c = \frac{1/r^g}{\sum_1^{\mathscr{A}} (1/\mathscr{A}^g)}, \tag{1}$$

where r is the popularity rank of content c, g is the Zipf parameter, and $\mathscr{A}$ is the amount of content. The higher g is, the more concentrated the content request distribution is around the most popular content [30].

3.3. System Model. To provide a stable and low-content latency service, we propose a cooperative caching policy to manage content caching and delivery. In detail, high-speed users are clustered in the MBS and their requested content is cached there. There are two content delivery scenarios for these users. In scenario 1, fast-moving users directly retrieve their desired content from the MBS if the content is cached at the serving MBS, that is, the requested content is hit at the serving MBS. In scenario 2, if the desired content is not cached at the serving MBS, that is, content is missed, the serving MBS retrieves the requested content from the core network and then delivers it to the high-speed users.

Low-speed users' requested content is cached at the serving SBS, that is, cached locally. Specifically, we divide the SBS capacity into two parts, the local and global modules. The local module caches the most popular content that has the highest value in the local content popularity distribution, and the global module caches the popular content that is not cached in the local module but still has the relatively high value in the cluster content popularity distribution. There are three scenarios for low-speed users. In scenario 3, if the requested content is cached locally, it can be directly delivered from the serving SBS to requesting users. In scenario 4, if the requested content is missed locally, the serving SBS needs to retrieve the content from its adjacent SBSs that are in the same group. In scenario 5, if the adjacent SBSs do not cache the content, the requested content needs to be delivered from the core network to the serving users. As the content delivery distance increases, the content access latency increases. Consequently, $\ell 3 < \ell 4 < \ell 1 \ll \ell 2 < \ell 5$, where $\ell 1$, $\ell 2$

, $\ell 3$, $\ell 4$, and $\ell 5$ are the average content access latency of scenarios 1, 2, 3, 4, and 5, respectively.

3.4. Problem Formulation. In this work, by designing an optimal scheme for cache content update and cache storage allocation, massive content requests from users are satisfied at the SBS or at the MBS, rather than from the core network over the backhaul links [31]. This can reduce latency for accessing content [32]. Two definitions are introduced here: the average content access latency and traffic offloading ratio, i.e., the ratio of the amount of data satisfied on the BS (either MBS or SBS) to the total data volume [33]. For example, there are ten content requests, and only seven content requests are satisfied the BS. The traffic offloading ratio is then equal to $7 \times 1/10 \times 1 = 7/10$, where the size of each content is 1 bit. Based on the above five scenarios, these two definitions are described in detail.

For content c requested by the slow-moving users, the successful retrieval for content c goes through three scenarios. In scenario 3, the latency of the successful retrieval for content c can be expressed as $\ell 3 \times \gamma_{(i,c)}^{(i,q)}$, where $\gamma_{(i,c)}^{(i,q)}$ is the probability that the request for content c is processed by the serving SBS i connected to MBS_q ($q \in [\mathrm{MBS}]$). In scenario 4, the latency for content c is equal to $\ell 4 \times \sum_{i \neq j, j \in S^q} \gamma_{(i,c)}^{(j,q)}$, where SBS j is in the same cluster with the serving SBS i, S^q is the set of SBSs in the same group covered by MBS_q, and $\gamma_{(i,c)}^{(j,q)}$ is the probability that the request for content c is processed by SBS j connected to MBS_q. In scenario 5, the probability that content c is obtained from the core network is equal to $(\mathscr{R}_i^q \times P_{(i,c)}^q - \sum \gamma_{(i,c)}^{(j,q)})$, where $\mathscr{R}_i^q$ is the amount of content library $\phi_{(q,i)}$ in SBS i, $P_{(i,c)}^q$ is the popularity of content c in SBS i connected to MBS_q, and $\sum \gamma_{(i,c)}^{(j,q)}$ is the sum of the probabilities that the request for content c will be processed by either the serving SBS i or the adjacent SBS j. The latency of successful retrieval for contents c is equal to $\ell 5 \times (\mathscr{R}_i^q \times P_{(i,c)}^q - \sum \gamma_{(i,c)}^{(j,q)})$. Therefore, the total latency for the contents requested by the slow-moving users is expressed as follows:

$$\sum L_{\mathrm{slow}} = \sum_{q=1}^{Q} \sum_{i \in S^q} \sum_{c \in \phi_{(q,i)}} \left[\ell 3 \times \gamma_{(i,c)}^{(i,q)} + \ell 4 \times \sum_{i \neq j, j \in S^q} \gamma_{(i,c)}^{(j,q)} + \ell 5 \times \left(\mathscr{R}_i^q \times P_{(i,c)}^q - \sum \gamma_{(i,c)}^{(j,q)} \right) \right]. \tag{2}$$

After obtaining the probability that the request for content c is processed by the serving SBS i and the probability that the demand for content c is processed by SBS j, the probability that the request for content c is offloaded at the SBS is equal to $\gamma_{(i,c)}^{(i,q)} + \sum_{i \neq j, j \in S^q} \gamma_{(i,c)}^{(j,q)}$. Therefore, the total traffic for the content requested by the slow-moving users offloaded at SBSs is calculated as follows:

$$\varkappa_{\mathrm{slow}} = \sum_{q=1}^{Q} \sum_{i \in S^q} \sum_{c \in \phi_{(q,i)}} \left(\gamma_{(i,c)}^{(i,q)} + \sum_{i \neq j, j \in S^q} \gamma_{(i,c)}^{(j,q)} \right) S_c, \tag{3}$$

where S_c is the size of the content c.

For the content c requested by the fast-moving users, the successful retrieval for content c goes through two phases. In scenario 1, the latency for successful retrieval for the content c can be expressed as $\ell 1 \times \gamma_c^q$, where γ_c^q is the probability that MBS_q will process the request for content c. If the request for content c is missed at MBS_q, it needs to be obtained from the core network, as described in scenario 2. The latency for successfully receiving the content c from the core network is equal to $\ell 2 \times (1 - \gamma_c^q)$. Consequently, the total latency for the content requested by the fast-moving users is expressed as follows:

$$\sum L_{\mathrm{fast}} = \sum_{c \in \varphi_q} [\ell 1 \times \gamma_c^q + \ell 2 \times (1 - \gamma_c^q)], \tag{4}$$

where φ_q is the content library of MBS_q.

Considering that the data traffic can only be offloaded at the MBS, the probability that the request for content c is equal to γ_c^q. Therefore, the total traffic for the content requested by the fast-moving users offloaded at SBSs is calculated as follows:

$$\varkappa_{\mathrm{fast}} = \sum_{c \in \varphi_q} \gamma_c^q S_c. \tag{5}$$

Based on $\sum L_{\mathrm{fast}}$ and $\sum L_{\mathrm{fast}}$, the average content access latency $\bar{L}$ can be derived as follows:

$$\bar{L} = \frac{\sum L_{\mathrm{slow}} + \sum L_{\mathrm{fast}}}{\sum_{q=1}^{Q} \left(\sum_{i \in S^q} \mathscr{R}_i^q + \mathscr{R}_q \right)}, \tag{6}$$

where $\mathscr{R}_q$ is the amount of content library φ_q.

Base on the $\varkappa_{\mathrm{slow}}$ and $\varkappa_{\mathrm{fast}}$, the system traffic offloading ratio $\vec{\varkappa}$ is calculated as follows:

$$\vec{\varkappa} = \frac{n_{\mathrm{slow}} \varkappa_{\mathrm{slow}} + n_{\mathrm{fast}} \varkappa_{\mathrm{fast}}}{\sum D}, \tag{7}$$

where n_{slow} is the number of content requests by the slow-moving users, n_{fast} is the number of content requests by fast-moving users, and $\sum D$ is the total system data traffic.

The caching problem minimising the average content access latency is expressed as follows:

$$P_1 : \min \bar{\boldsymbol{L}}, \tag{8}$$

$$\text{s.t.} \sum_{c \in \psi_x} S_{(c,x)} \leq M_x, \forall x \in \left(\mathrm{MBS}_q \cup \mathrm{SBS}_w \right), \tag{9}$$

where $S_{(c,x)}$ is the size of content c in BS x, ψ_x is the set of the cached content in BS x, M_x is the storage capacity of BS x.

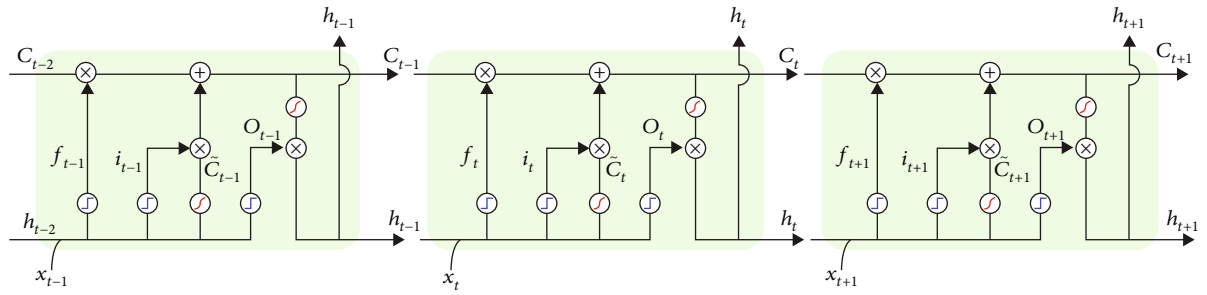


Figure 2: Structure of the LSTM neural network with three modules.

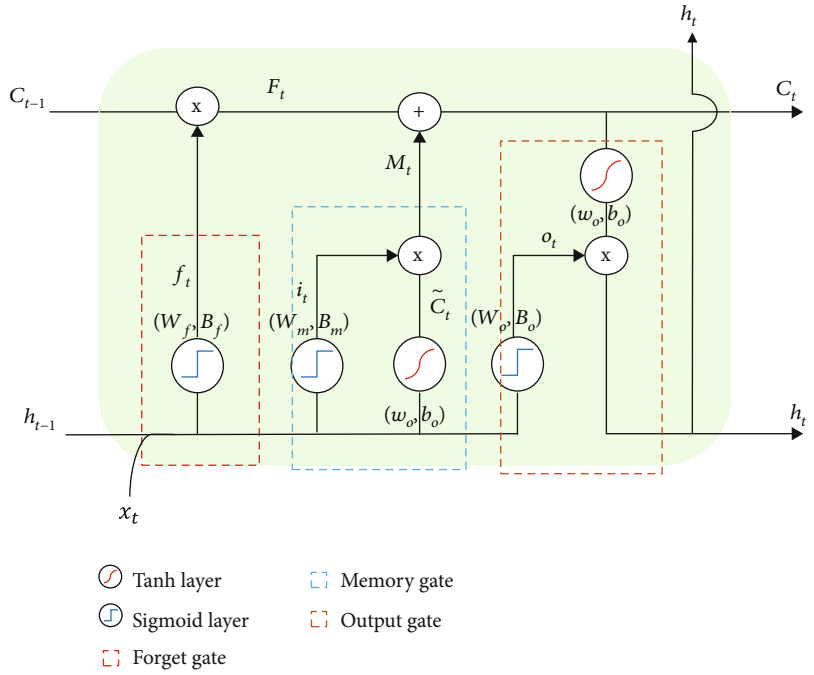


Figure 3: One module of the LSTM neural network.

The constraint in Equation (9) demonstrates that the total cached content should not exceed the cache capacity.

As $\ell 3 < \ell 4 < \ell 1 \ll \ell 2 < \ell 5$ is to minimise the average latency of accessing content, we should maximise the amount of successful content retrievals from the local BS. Specifically, we try to ensure that the desired content can be obtained from the MBS for high-speed users. For low-speed users, we try to ensure that the desired content can be obtained from the local SBS or nearby SBSs in the same cluster.

4. The Cooperative Caching Policy

In this section, we introduce a cooperative caching policy to efficiently manage content caching and delivery. First, a content popularity prediction based on the LSTM neural network is introduced. We then present the size-weighted content popularity. Finally, SBS caching and MBS caching are introduced to improve the cache performance.

4.1. The Long Short-Term Memory Neural Network. This paper utilises a 3-dimensional LSTM neural network that has three modules with the same structures, as shown in Figure 2.

To clearly explain the LSTM, a 1-dimensional LSTM neural network is shown in Figure 3. The output state h_t of current time t can be weighted combined with the previous hidden state h_{t-1} which is also known as the previous output state, the previous cell unit state C_{t-1}, and the current input state $\mathcal{X}_t$. The tanh layer is used to avoid gradient explosions, and the sigmoid layer forgets the recurrent information from the previous module. The LSTM neural network procedure is as follows:

(1) The LSTM determines the previous information that should be discarded, and this is achieved by the sigmoid layer in the forget gate. The forget value f_t is

$$f_t = \text{sigmoid}\left(W_f \bullet (h_{t-1}, \mathcal{X}_t) + B_f\right). \quad (10)$$

(2) The memory gate adds new information to the LSTM. The memory gate's sigmoid layer determines

the new output value i_t by selectively deleting some information on $h_{t-1}\mathcal{X}_t$, and the tanh layer generates a new value $\tilde{C}_t$ as follows

$$i_t = \text{sigmoid}(\text{W}_\text{m}\bullet(h_{t-1}, \mathcal{X}_t) + \text{B}_\text{m}),. \tag{11}$$

$$\tilde{\text{C}}_\text{t} = \tanh\,(\text{w}_\text{m}\bullet(h_{t-1}, \mathcal{X}_t) + \text{b}_\text{m}).. \tag{12}$$

(3) The new cell unit state C_t is updated by combining F_t and M_t as follows:

$$F_t = \text{f}_t * C_{t-1},. \tag{13}$$

$$M_t = i_t * \tilde{C}_t,. \tag{14}$$

$$C_t = F_t + M_t,. \tag{15}$$

where F_t determines the discarded information and M_t determines the new information being added.

(4) In the output gate, the sigmoid layer is utilised to decide which part of $h_{t-1}\mathcal{X}_t$ is output. The tanh layer is then utilised to normalise the cell unit state C_t. The current hidden state h_t and predicted output of LSTM Y_p are expressed as follows:

$$o_t = \text{sigmoid}(W_o(h_{t-1}, \mathcal{X}_t) + B_o,. \tag{16}$$

$$h_t = o_t * \tanh\,(C_t),. \tag{17}$$

$$Y_p = \sigma\left(Y_p\right),. \tag{18}$$

where σ is the activation function. The Relu function is applied since predicting the content popularity is a fitting problem.

4.2. Content Popularity Prediction Based on the Long Short-Term Memory Neural Network. The requesting probability of content c is denoted as $p_c = (p_{c1}, p_{c2}, \cdots, p_{ct}, \cdots p_{cT})$, where p_{ct} is the probability of content c being requested at time t whose time duration is 4 hours. Because this paper applies a 3-D LSTM neural network, the training data S^u is in the form of $(X^u_{-2}, X^u_{-1}, X^u_0, Y^u_1)$ where u is the training step, $\{X^u_{-2}, X^u_{-1}, X^u_0, Y^u_1\}$ is a continuous sequence chosen from p_c, $\{X^u_{-2}, X^u_{-1}, X^u_0\}$ is the input data, and Y^u_1 is the real upcoming output value as shown in Table 3. For each training step u, the predicted output Y^u_{pre} is obtained by inputting the sequence $\{X^u_{-2}, X^u_{-1}, X^u_0\}$ into the LSTM neural network. The mean squared error (MSE) is utilised to generate the loss value ξ^u which is

$$\xi^u = \left(Y^u_1 - Y^u_{pre}\right)^2.. \tag{19}$$

The LSTM has the automatic ability to minimise the loss value as the training steps increase. After several training steps, the LSTM with low loss can predict the content popularity. In a properly trained LSTM neural network, the content popularity p_{cT+1} of content c at future time $T+1$ can be predicted by inputting the newest content popularity sequence $\{p_{cT-2}, p_{cT-1}, p_{cT}\}$ into the LSTM neural network. The content popularity prediction based on the LSTM neural network is shown in Algorithm 1.

Table 3: Training data in the 3-D LSTM neural network.

Training step u	Training sequence	X_{t-2}, X_{t-1}, X_t	Y
1	$\{p_{c1}, p_{c2}, p_{c3}, p_{c4}\}$	$\{p_{c1}, p_{c2}, p_{c3}\}$	p_{c4}
2	$\{p_{c2}, p_{c3}, p_{c4}, p_{c5}\}$	$\{p_{c2}, p_{c3}, p_{c4}\}$	p_{c5}
3	$\{p_{c3}, p_{c4}, p_{c5}, p_{c6}\}$	$\{p_{c3}, p_{c4}, p_{c5}\}$	p_{c6}

4.3. Size-Weighted Content Popularity. Edge caching stores popular content since it is often requested. Caching the popular content can improve the cache performance when the cache capacity is finite [13]. However, if the caching decision only considers the content popularity but ignores the content size, it will reduce the cache performance in a practical scenario in which different content has different sizes. For example, as shown in Figure 4, the cache device in case 1 stores the most popular content, that is, content A, while it has no additional capacity to store other content. The cache device in case 2 stores relatively less popular content with a relatively small size, that is, content B and C. As a result, case 2 has better content popularity than case 1, and the cache content placement in case 2 outperforms case 1.

To maximise the utilisation of limited cache capacity and balance the effect of the content size and popularity on the cache performance, we introduce a size-weighted content popularity P_{s_x}, which is calculated as:

$$P_{s\_x} = \frac{\rho_x}{s_x}, \tag{20}$$

where ρ_x is the content popularity of content x and s_x is the size of content x.

4.4. Cooperative Small Base Station Cache. In this subsection, we introduce the cache policy for the cache-enabled SBS. For convenience, we classify the requested content into two categories, popular and unpopular categories. Specifically, we rank the content based on the size-weighted content popularity in descending order, and we classify the set of content with the top K values into the popular category. Only the content in the popular category has a chance of being cached.

We divide the cache of each SBS into two modules, local and general modules as shown in Figure 5(a). General modules in the same cluster can communicate with one another while the local modules cannot. The local module caches the most popular content, and the general module caches the relatively less popular content that has not been cached. The cached content in each local module in the same cluster can be repeated, while the cached content in each general module in the same cluster is unique. More specifically, the caching content in the local module is based on the predicted size-weighted content popularity distribution of the corresponding SBS, and the caching content in the local module

Input: The content library $\mathcal{F}$ and number of the training time slots T.
Initialise the predicted content popularity ϕP.

For c =1,…, F, ***do***
  Obtain p_c
  For u =1, T, ***do***
    Input $\{p_{cu}, p_{cu+1}, p_{cu+2}\}$
    Obtain Y^u_{pre}
    Loss value $\xi^u=(Y^u_1 - Y^u_{pre})^2$
    Backpropagates ξ^u
  Adjust the weights of all of the neural layers
  Obtain the trained LSTM neural network
End for
  Input$\{p_{cT-2}, p_{cT-1}, p_{cT}\}$ into the trained LSTM neural network
  Obtain Y^c_{T+1}
End for
$\phi P=(Y^1_{T+1}, Y^2_{T+1}, \cdots, Y^c_{T+1}, \cdots, Y^F_{T+1})$

Algorithm 1: The content popularity prediction based on the LSTM.

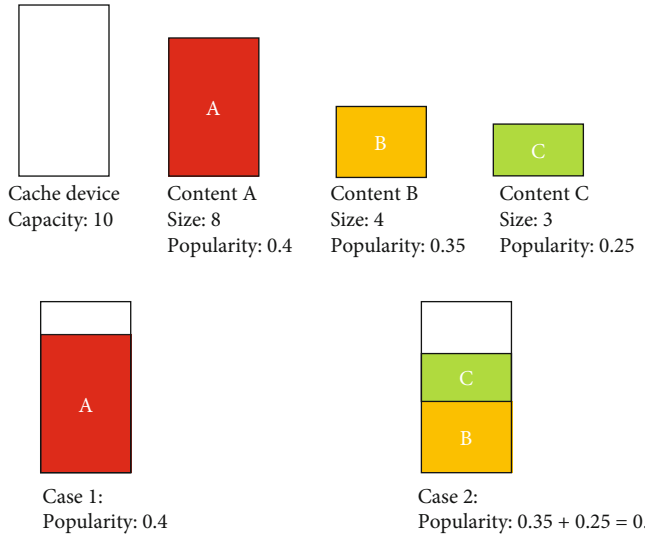


Figure 4: The two cache content placements. The capacity of the cache device, content A, content B, and content C is 10, 8, 4, and 3, respectively. The popularity of content A, content B, and content C is 0.4, 0.35, and 0.25, respectively.

is based on the predicted size-weighted content popularity distribution of the entire SBS in the same cluster. For example, there are three cache-enabled SBSs in the same cluster, as shown in Figure 5(b). The purple part is the local module that stores the most popular content in the serving SBS, and the yellow part represents the general module that caches the relatively less popular content that is not yet cached. The content with the highest local size-weighted predicted content popularity is cached at the corresponding local module. The three local modules cache {content A and B}, {content B and C}, and {content D and B}, respectively. Their cached content is repeated. The general module caches the less popular content that has not been cached in the local module. Because general modules in the same cluster can exchange their cached content, each content will only be cached at one general module in a cluster. The cached content in each general module is {content E and F}, {content I and G}, and {content J and K}, respectively. Each general module in the same cluster has no repeated content.

To achieve our target, we ensure that low-speed users in the SBS coverage areas have minimal average cache content access latency. The average cache content access latency for low-speed users is denoted as $\overline{L_{Lsp}}$ and calculated as follows:

$$\overline{L_{Lsp}} = \sum_{x\in[\text{User}]}\sum_{i\in r_x}\left(\ell_3\times\mathcal{P}_x^{(i,3)} + \ell_4\times\mathcal{P}_x^{(i,4)} + \ell_5\times\mathcal{P}_x^{(i,5)}\right)/\sum_{x\in[\text{User}]} N_{r_x}, \tag{21}$$

where r_x is the set of the requested content of user x, $\mathcal{P}_x^{(i,3)}$ is the requested probability of content i requested by user x in scenario 3, $\mathcal{P}_x^{(i,4)}$ is the requested probability of content i requested by user x in scenario 4, $\mathcal{P}_x^{(i,5)}$ is the requested

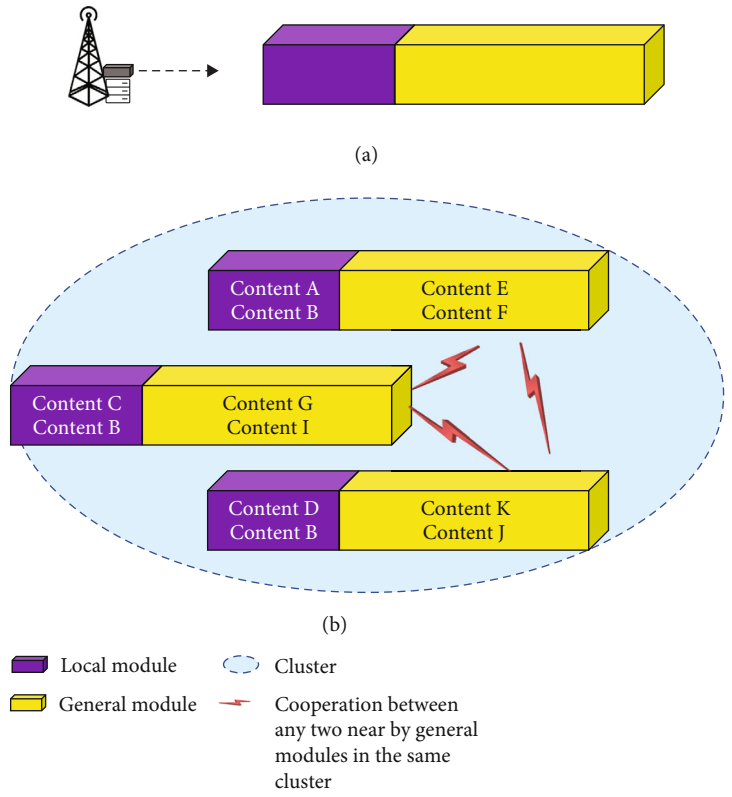


Figure 5: Detailed content placement in the local and general modules.

probability of content i requested by user x in scenario 5, and N_{r_x} is the amount of r_x.

4.5. Small Base Station Cache Capacity Allocation. To minimise the average latency $\overline{L_{Lsp}}$, we should ensure that the requested content is cached from the SBS perspective, that is, either in the local SBS or nearby SBSs in the same cluster as any additional requested content missing at the SBS and retrieved from the core network results in a high average content access latency. Due to the limited caching capacity of each SBS, we need to determine the capacity allocation for the local and general modules. Hence, we introduce an algorithm that is called the popularity maximising algorithm (PMA). The PMA allocates the caching capacity to the local and general modules. The local module stores content based on the local SBS's size-weighted content popularity, and the general module stores content that is not cached from the entire cluster's perspective. The content can be cached at one general module at most in the entire cluster. The PMA finds an optimal caching capacity allocation that has the maximal overall content popularity of cluster x, which is calculated as:

$$\wp_x = \sum_{i \in n(x)} \left[q(i_l) + q(i_g) \right], \tag{22}$$

where $n(x)$ is the number of SBSs in cluster x, $q(i_l)$ is the content popularity of local module i in cluster x, and $q(i_g)$ is the content popularity of general module i in cluster x.

For simplicity, we assume that each SBS in the same cluster x when $x \in Q$ has the same cache capacity M^x, and the capacity of each local and general module in cluster x is M_l^x and M_g^x, respectively. The PMA's specific operation is shown in lgorithm 2. We introduce the capacity allocation algorithm in one cluster, and it is the same in the rest of the clusters. First, the cache capacities of each local and general module are updated by increasing the cache capacity of each local module M_l^x by one percentage. Based on the updated capacity, the local module chooses κ content that has the top κ values from the local size-weighted content popularity distribution, and the general module chooses N content that has the top N values from the content popularity distribution in the entire cluster. The content is sorted in descending order based on the size-weighted content popularity in the local content popularity and the entire cluster. Furthermore, κ content and N content should not exceed the capacity of the local and general modules, respectively. If increasing the local module can produce a better $\wp_x$ than before, we keep this increase; otherwise, we reduce the local module's capacity by one percentage. The process is finished when $\wp_x$ barely changes regardless of how the capacity of the local module increases or decreases.

4.6. Macro Base Station Cache. The MBS cache is for high-speed users who do not have enough time in their corresponding SBSs. Once their requested content is cached at the MBS, regardless of how frequently they switch among the SBSs, they can still retrieve their desired content from

Input: M^x, M_l^x, and M_g^x.
Initialise: $\wp_x = 0$ and $t = 1$.
Randomly generate a value m for $0 < m < M^x$
  $M_l^x = m$
  $M_g^x = M^x - M_l^x$
  Obtain a new $\wp_x$ based on the initial capacity of the local module M_l^x.
  $\nabla = new\wp_x - initial\wp_x$
  For $t = 1$, T, ***do***
  If $|\nabla| \leq 5\%$, ***then***
    Output M_l^x and M_g^x
  End if
  End for
  Elif $\nabla > 5\%$, ***then***
  New $M_l^x = M_l^x + 1$
  $\nabla = new\wp_x - initial\wp_x$
  $t = t + 1$
  Elif $\nabla < 5\%$, ***then***
  New $M_l^x = M_l^x - 1$
  $\nabla = new\wp_x - initial\wp_x$
  $t = t + 1$

Algorithm 2: Capacity allocation of the SBS cache.

the MBS, which has a large coverage area. As the MBS has a relatively large capacity, the effect of the content size on the cache performance is ignored. Therefore, in the MBS cache, we cache the contents that have the highest content popularities to maximise their effect on the cache performance.

5. The Simulation Results

5.1. The Simulation Settings. In this section, we analyse the average content access latency and the data offloading ratio of our proposed cooperative caching policy. Here, an LSTM neural network with three cells is proposed to predict the distribution of content popularity. Specifically, for each cell, the size of the input layer is equal to 3. The size of each hidden layer, i.e., the tanh layer and the sigmoid layer, is 4. The learning rate $\mathscr{E}$ is equal to 0.05. The cache scenario consists of 1 core network, 1 MBS, 5 SBSs within the same cluster, 20 slow-moving users, and 10 fast-moving users. Here, to alleviate the effect of frequent handover of the users on the content popularity distribution of SBSs, the users who move in and out of the coverage area of one SBS more than 2 times during one interval are defined as fast-moving users, and the rest are slow-moving users. The Zipf parameter varies from 1.1 to 1.6. The cache update interval is 4 hours. In each time slot, each slow-moving user within the coverage area of an SBS is moved. The fast-moving users are assumed to be able to move from one SBS to another adjacent SBS while always being covered by the same MBS. The capacities of the MBS, each SBS, and the core network are 200 units, 50~100 units, and infinite units, respectively. The number of content requests is in a range of 3000-30000, and the size of each content varies from 1 to 5 units. $\ell 1$, $\ell 2$, $\ell 3$, $\ell 4$, and $\ell 5$ are 4 ~ 6 ms, 8 ~ 15 ms, 0.5 ~ 1 ms, 2 ~ 4 ms, and 10~20 ms, respectively, which are the average content access latencies of our proposed 5 scenarios. The simulations are achieved by Pytorch language in Pycharm software, running at the personal computer whose central processing unit (CPU) is Intel Core i7-1165G7. The main simulation settings are shown in Table 4.

5.2. The Benchmark Policies. This subsection first briefly reviews the CPP-CC policy. To evaluate the performance of our proposed policy, we compare CPP-CC with the existing methods whose cache content update is described as follows:

(1) *CPP-CC Policy*. The CPP-CC policy is a predictive policy, i.e., the cache decision is made before the content request. At the end of time t, each BS updates its original cached content by precaching the new content that will be in high demand in the future time $t + 1$. Based on the predicted future distribution of content popularity, the new cached content can be decided in the MBSs and SBSs, as described in Sections 4.4 and 4.6

(2) *The Uncollaborative Predictive Cache (UPC) Policy*. The difference between the CPP-CC and UPC policies is that the SBSs in the UPC policy cannot communicate with each other. Therefore, once the content request is missed at the serving SBS, the content must be retrieved from the MBS and not from the nearby SBSs. Moreover, the SBS cache stores locally popular content. Apart from the above differences, the rest of the UPC policy is the same as that of the CPP-CC policy

(3) *Collaborative LFU (CLFU) Policy*. The CLFU policy is a reactive cache, i.e., the cache decision is made after the content request. More specifically, in the cache content update phase, at the end of time t, each BS updates its cache content based on the distribution of content popularity collected in the last time $t - 1$.

Table 4: The main simulation settings.

Amount of requested content	3,000~ 30,000
Caching capacity of the MBS	200 units
Caching capacity of the SBSs	50~ 100 units
Number of SBSs	5
Zipf parameters	1.1 ~ 1.6
Size of the content	1 ~ 5 units
Number of mobile users	30
Learning rate, $\mathscr{E}$	0.05
Average latency of scenario 1, $\ell 1$	4 ~ 6 ms
Average latency of scenario 2, $\ell 2$	8 ~ 15 ms
Average latency of scenario 3, $\ell 3$	0.5 ~ 1 ms
Average latency of scenario 4, $\ell 4$	2 ~ 4 ms
Average latency of scenario 5, $\ell 5$	10~ 20 ms

The content with the highest value in the content popularity distribution can be cached [34]. Except for the above difference, the rest of the CLFU policy is the same as the CPP-CC policy

(4) *The Collaborative Least Recently Used (CLRU) Policy*. The CLRU policy is also a reactive policy, hence, the cache decision is made after the content request. Specifically, in the cache update phase, at the end of time t, each BS updates its cache content based on the most recently requested content collected in the last time $t-1$ [35]. Except for the above difference, the rest of the CLRU policy is the same as that of the CPP-CC policy

5.3. The Effect of the SBS Cache Capacity Allocation on the Cache Performance. Figures 6 and 7 show the average system latency and offloading ratio under different proportions of local module capacity to the SBS cache capacity. The capacity of each SBS cache is 100 units, and the number of the whole content requests is 3000. First, as the local module's capacity increases, the average latency of our proposed policy decreases, and the offloading ratio improves. This occurs because more content can be retrieved from the serving SBS or nearby SBS caches than from the remote core network via the backhaul links. When the proportion reaches 30%, the average latency gradually increases, and the offloading ratio decreases. More content can be cached at the local module as its capacity increases. However, the new cached content in the local module gradually becomes less popular than the initial cached content, and the less popular content in the local module cannot be retrieved by the nearby general modules. Therefore, caching the less popular content in the local module wastes the local module's capacity, and the general module has insufficient capacity since the total capacity of each SBS cache is constant. As demonstrated, the proportion of 30% is an optimal percentage.

5.4. The Accuracy of Content Popularity Prediction Based on LSTM Neural Network. To evaluate the effect of the number of contents on the prediction of content popularity, 72 groups of training data and 720 groups of training data are applied for training the LSTM neural network. Each group has 20 raw data. Here, the difference ratio, $\alpha = |y_p - y_r|/y_r$, is proposed to measure the content popularity prediction, where y_p and y_r are the predicted Zipf value based on the LSTM neural network and the real Zipf value. The Zipf value can represent the distribution of content popularity. Figure 8 shows the difference ratio when selecting 72 groups of training data for training the LSTM neural network. The loss of the corresponding LSTM neural network is shown in Figure 9(a). When choosing 720 groups of training data, the corresponding difference ratio and loss of the LSTM neural network are shown in Figures 10 and 9(b), respectively. As shown in Figures 9(a) and 9(b), both loss functions gradually converge with the increase of training step, which means that both LSTMs are well trained and can accurately predict content popularity. However, when comparing Figures 8 and 10, the trends of the two curves are different. The curve in Figure 8 always fluctuates, and while in Figure 10, the curve gradually converges with a low value. The reason is that the content prediction based on 720 groups of training data is accurate, while the content prediction with 72 groups of training data is not accurate. This, in turn, reflects the importance of the number of training data on the content popularity. Therefore, 720 groups of training data are chosen for content popularity prediction in the following simulations.

The effect of learning rate $\mathscr{E}$ on content popularity is studied, as shown in Figure 11. More specifically, Figure 11(a) shows the different ratio α when the learning rate $\mathscr{E}$ is 0.05, and Figure 11(b) shows the different ratio α when the learning rate $\mathscr{E}$ is 0.5. Here, 720 groups' training data are chosen for content popularity prediction. As can be seen, the difference ratio of Figure 11(a) is gradually converging with a low value. In contrast, Figure 11(b) shows unstable fluctuations in the difference ratio. The reason is that the value of the learning rate 0.5 is too high, which leads to a large step length. The large step length causes careless exploration for the optimal solution, and hence, the optimal value cannot be achieved. Therefore, choosing a proper learning rate is crucial, and in this paper, the appropriate learning rate is 0.05.

5.5. The Effect of the SBS Cache Capacity on the Cache Performance. As shown in Figure 12, we investigated the effect of the SBS cache capacity on the average system latency. We assume that the capacity of the SBS cache varies from 50 to 150 units. There are 3,000 content requests. As the SBS cache's capacity increases, the 4 curves gradually decline. This occurs because, as the capacity of each SBS cache increases, more content can be cached at the SBS servers, that is, the local SBS or nearby servers, and more content retrievals can be processed at the SBS server than from the remote core network. Compared to UPC, CPP-CC has a lower average latency. The reason is that UPC does not consider the cooperation between SBSs. Therefore, if the requested content is not cached in the SBS, it has to be obtained from the remote network. However, the SBS of the CPP-CC policy can obtain the content from the nearby SBSs once the requested content is not cached locally. On the other

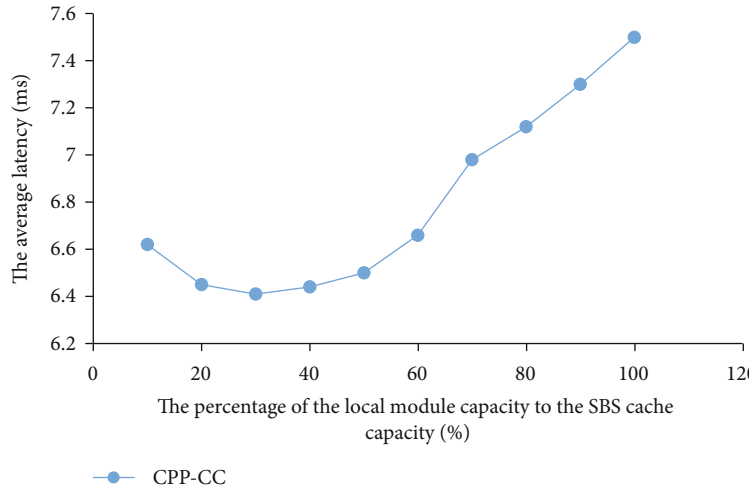


Figure 6: The average system latency vs. the percentage of the local module's capacity to the SBS cache capacity. The percentages are equal to 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100%, respectively.

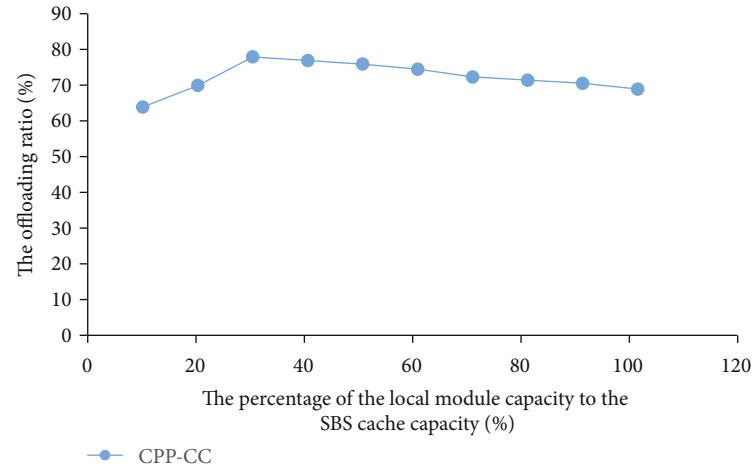


Figure 7: The offloading ratio vs. the percentage of the local module's capacity to the SBS cache capacity. The percentages are equal to 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100%, respectively.

hand, the CPP-CC policy has lower latency than the CLRU and CLFU policies. This is because CPP-CC is a predictive policy, where the content that will be in high demand in the next time $(t+1)$ is cached at the end of the current time t. This is the reason why CPP-CC has lower latency than CLRU and CLFU policies. In contrast, CLRU and CLFU are reactive policies. The content must be requested first, and then it can be decided whether to cache it or not. This means that the cache content update at the end of the current time t is the content requested by users at the previous time $(t-1)$. However, the popularity of the content is time-varying, which causes the cached content to be out of date at the next time $(t+1)$, and therefore, more contents need to be obtained from the core network. Moreover, CPP-CC considers the effect of content size on popularity, which further increases the usage of limited cache capacity.

The offloading ratio is plotted as a function of the SBS cache's capacity as demonstrated in Figure 13. The MBS cache's capacity is 200, and the SBS cache's capacity varies from 50 to 150 units. There are 3,000 content requests. As the SBS cache's capacity increases, the four methods' offloading ratios increase. This occurs because more content can be cached at the SBS cache, and the content requests that originally need to be satisfied at the core network via the backhaul links can be processed at the SBSs. Consequently, more data traffic can be offloaded from the backhaul links as the SBS cache's capacity increases.

Vertically, the UPC and CPP-CC policies have a higher traffic offloading ratio than the CLRU and CLFU policies. Achieving the high offloading ratio depends on predeployment of the content that is in high demand, which avoids the expiration of the cached content. For the two predictive caching policies, the CPP-CC policy performs better than the UPC policy in terms of offloading rate. This is because the UPC policy neglects SBS cooperation. In this situation, massively duplicated contents are cached in the adjacent

Figure 8: The difference ratio vs. the number of the training data.

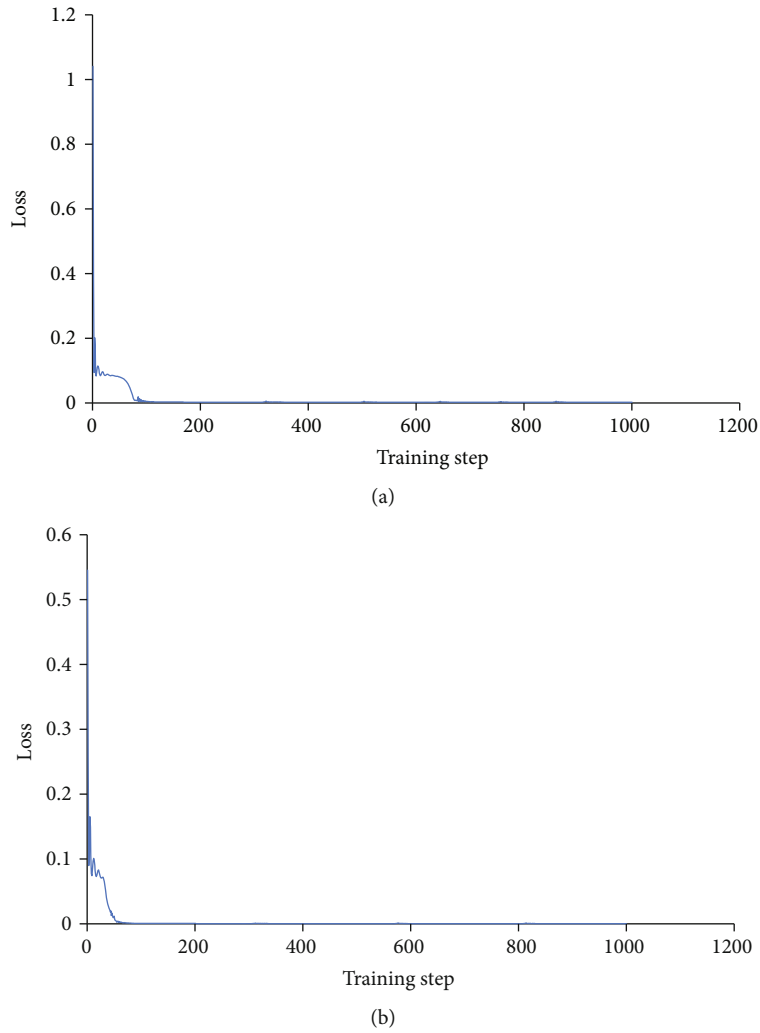


Figure 9: The loss of the LSTM neural network with (a) 72 groups of training data and (b) 720 groups of training data vs. the training step.

SBSs, which leads to the waste of the limited SBS cache storage resources. Therefore, more content cannot be cached and more content has to be obtained from the core network, resulting in a worse traffic offloading ratio than the CPP-CC policy.

5.6. The Effect of the Number of the Content Requests on the Cache Performance. Figure 14 displays the relationship between the average latency and the number of content requests. The MBS cache's capacity is 200 units, and each SBS cache's capacity is 50 units. The number of content

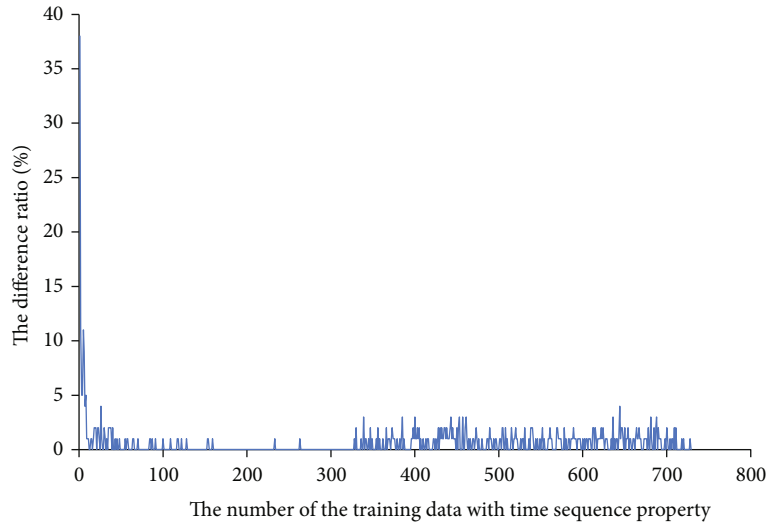
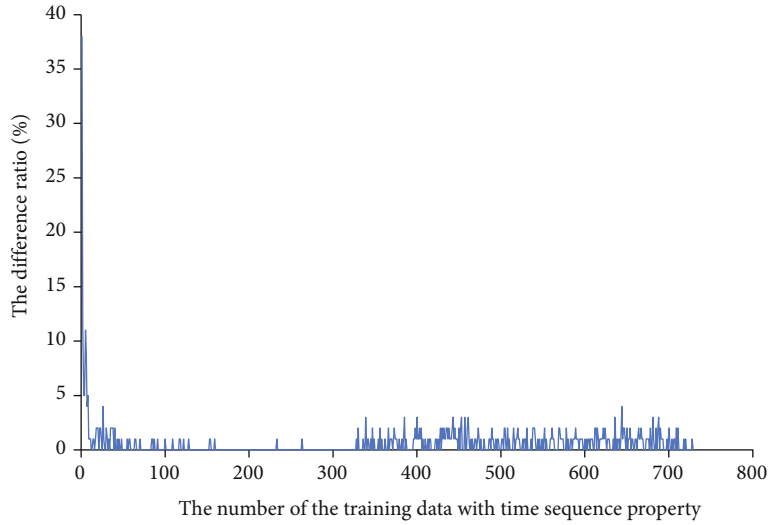


Figure 10: The difference ratio vs. the number of training data.



(a)

(b)

Figure 11: The difference ratio of the LSTM neural network whose learning rate $\mathscr{E}$ is equal to (a) 0.05 and (b) 0.5.

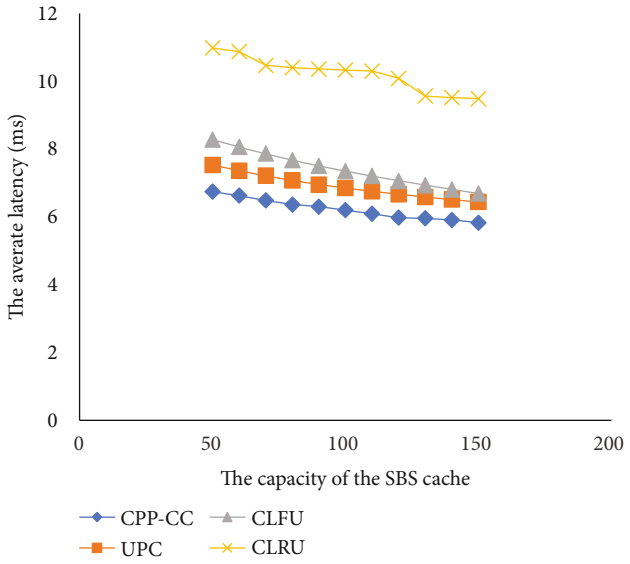


Figure 12: The average system latency vs. the SBS cache's capacity. Each SBS cache's capacity is equal to 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, and 150, respectively.

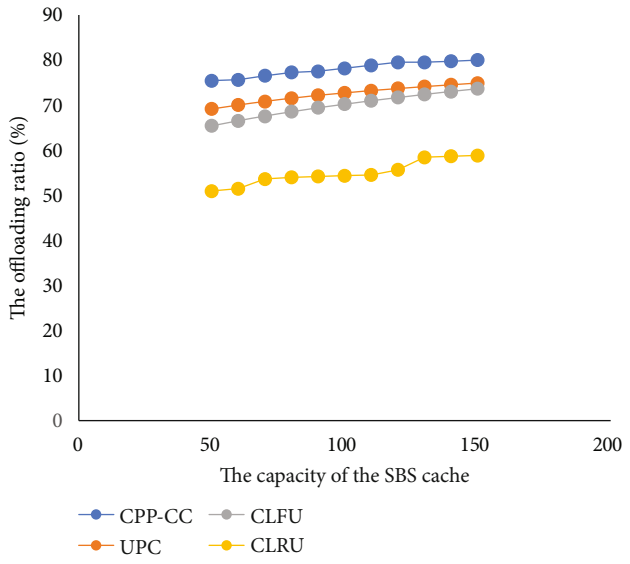


Figure 13: The offloading ratio vs. the SBS cache's capacity. Each SBS cache's capacity is equal to 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, and 150, respectively.

requests varies from 3,000 to 30,000. As the number of content requests increases, the four methods' average system latencies increase. This occurs because, according to the Zipf distribution laws, the number of content increases as the number of content requests increases, which changes the size-weighted content popularity distribution. The average system latency increases since the initial cache method cannot fully adapt to the changed size-weighted content popularity. The increase in the CLRU curve is lower than those of the CLFU, UPC, and CPP-CC methods. This occurs because the CLFU, UPC, and CPP-CC methods focus on storing content with the highest content popularity, while

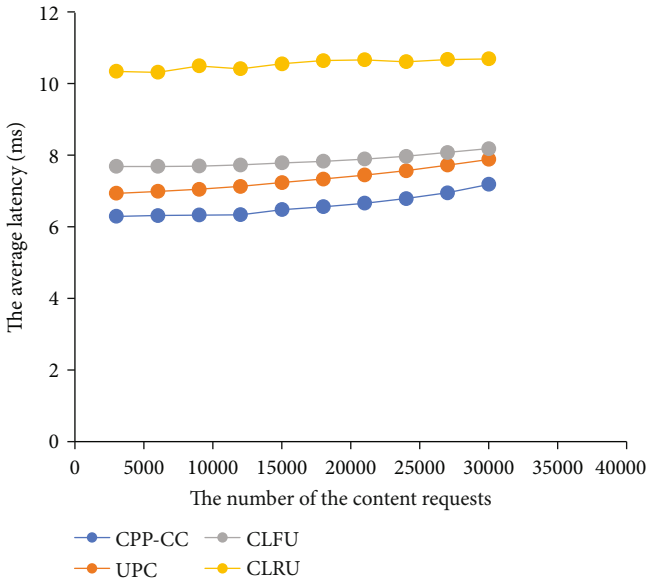


Figure 14: Average latency vs. the number of content requests.

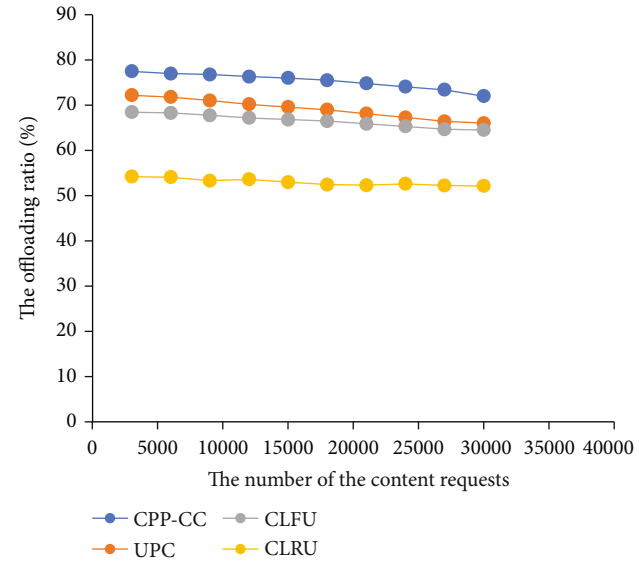


Figure 15: Offloading ratio vs. the number of content requests.

the CLRU policy stores the newest requested content rather than the content with the highest request probability; hence, the CLRU technique has a lower dependence on the content popularity than the other three methods. Moreover, the distance between the CLFU curve and the UPC curve gradually decreases as the number of content requests increases. This is because predicting content popularity becomes more difficult as the number of content queries increases. As the accuracy of content popularity prediction decreases, more content requests cannot be satisfied on the BS, which causes the increasing increased average latency of the UPC policy. In contrast, the effect of increasing content requests on the CPP-CC policy is not as significant as that on the UPC policy. The reason is that cooperation between neighbouring SBSs avoids caching massively repeated content, which means

Table 5: The summary of the reduction ratio of the average latency and the increase ratio of the offloading ratio by comparing CPP-CC policy with the UPC, CLFU, and CLRU policies.

	SBS cache capacity: 50~ 150		Number of content requests: 3000~ 3000	
	Average latency	Offloading ratio	Average latency	Offloading ratio
CPP-CC vs. UPC	9.2% ~ 10.4%	6.8% ~ 9.1%	8.9% ~ 11.1%	7.2% ~ 10.5%
CPP-CC vs. CLFU	12.8% ~ 18.5%	8.6% ~ 15.3%	12.2% ~ 18.2%	11.6% ~ 13.7%
CPP-CC vs. CLRU	37.7% ~ 40.9%	35.9% ~ 48.1%	32.8% ~ 39.7%	38.1% ~ 40.1%

more content can be cached in the SBS cluster. And each SBS can obtain the content from its nearby SBSs, which in turn increases the capacity of each SBS. Given the increasing content requests, more cache capacity means a higher chance that the requested content can be satisfied in the cacheable SBSs. Therefore, the average latency of the CPP-CC policy is lower than that of the UPC policy, and the increase in the average latency of the CPP-CC policy is slower than that of the UPC policy.

As shown in Figure 15, we investigated the relationship between the offloading ratio and the number of content requests. Similar to Figure 14, the MBS cache's capacity is 200 units, and each SBS cache's capacity is 50 units. The number of content requests varies from 3,000 to 30,000. As the number of content requests increases, the four methods' offloading ratios decrease. This occurs because more content requests are missed at the edge servers, that is, the MBS and SBS servers, and need to be obtained from the core network via the backhaul links as the number of content requests increases. Vertically, the CPP-CC policy has a higher offloading ratio than the CLRU and CLFU policies, regardless of how many content requests there are. This is because the cached content of the CLFU and CLRU policies have partially expired, which results in more content to be obtained from the core network through the backhaul links. Moreover, the higher offloading ratio of the CPP-CC policy compared to the UPC policy is achieved by the less frequent caching of content at the SBSs.

5.7. The Summary of the Simulation Results in terms of the Average Latency and the Offloading Ratio. As we can see, regardless of the SBS cache capacity, the CPP-CC policy is at least 9.2%, 12.8%, and 37.7% lower in terms of average latency and at least 6.8%, 8.6%, and 35.9% higher in terms of the offloading ratio compared to the UPC, CLFU, and CLRU policies, respectively. Moreover, the CPP-CC policy can reduce at least 8.9% more average latency and offload at least 7.2% more traffic compared to the UPC, CLFU, and CLRU policies. Therefore, the CPP-CC policy outperforms the other three policies in terms of average latency and traffic offload.

Table 5 summarises the reduction ratio in terms of the average latency and the increase ratio in terms of the offloading ratio of the CPP-CC policy over the UPC, CLFU, and CLRU policies. Here, the reduction ratio η is derived based on $\eta = L_{\text{existing}} - L_{\text{CPP\_CC}}/L_{\text{existing}}$, where $L_{\text{CPP\_CC}}$ and L_{existing} are the latency of the CPP-CC policy and any of the other three comparison methods, respectively. The increase ratio Γ is derived based on $\Gamma = \text{OFF}_{\text{CPP\_CC}} - \text{OFF}_{\text{existing}}/\text{OFF}_{\text{existing}}$, where $\text{OFF}_{\text{CPP\_CC}}$ and $\text{OFF}_{\text{existing}}$ are the offloading ratios of the CPP-CC policy and any of the other three policies, respectively. As we can see, regardless of the SBS cache capacity, the CPP-CC policy is at least 9.2%, 12.8%, and 37.7% lower in terms of average latency and at least 6.8%, 8.6%, and 35.9% higher in terms of the offloading ratio compared to the UPC, CLFU, and CLRU policies, respectively. Moreover, the CPP-CC policy can reduce at least 8.9% more average latency and offload at least 7.2% more traffic compared to the UPC, CLFU, and CLRU policies. Therefore, the CPP-CC policy outperforms the other three policies in terms of average latency and traffic offload.

6. Conclusion

In this study, we present a cooperative cache method with a two-level network that consists of a cache-enabled MBS and five cache-enabled SBSs. Specifically, an LSTM neural network is first applied to highly predict the time-varying content popularity. Then, the size-weighted content popularity is presented to balance the effect of the content size and predicted content popularity. Fast-moving users are connected to the cache-enabled MBS to ensure that the frequent handover of the fast-moving users does not change the predicted content popularity distribution of the SBS. Slow-moving users are connected to the cache-enabled SBS to provide low latency service. Cache-enabled SBSs in the same cluster can communicate with one another, which further increases the cache performance. The simulation results show that our proposed policy is at least 8.9% lower and 6.8% higher in terms of the average content access latency and offloading ratio, respectively, than the existing methods.

Data Availability

Content requests were described in the simulation section.

Conflicts of Interest

Lincan Li, Chiew Foong Kwong, Qianyu Liu, Pushpendu Kar, and Saeid Pourroostaei Ardakani declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] M. Radenkovic, V. S. H. Huynh, and P. Manzoni, "Adaptive real-time predictive collaborative content discovery and retrieval in mobile disconnection prone networks," *IEEE Access*, vol. 6, pp. 32188–32206, 2018.

[2] L. Chen, L. Song, J. Chakareski, and J. Xu, "Collaborative content placement among wireless edge caching stations with time-to-live cache," *IEEE Transactions on Multimedia*, vol. 22, no. 2, pp. 432–444, 2020.

[3] D. Xu, Y. Li, X. Chen et al., "A survey of opportunistic offloading," *IEEE Communication Surveys and Tutorials*, vol. 20, no. 3, pp. 2198–2236, 2018.

[4] B. Ai, A. F. Molisch, M. Rupp, and Z. D. Zhong, "5G key technologies for smart railways," *Proceedings of the IEEE*, vol. 108, no. 6, pp. 856–893, 2020.

[5] R. Wang, X. Peng, J. Zhang, and K. B. Letaief, "Mobility-aware caching for content-centric wireless networks: modeling and methodology," *IEEE Communications Magazine*, vol. 54, no. 8, pp. 77–83, 2016.

[6] S. M. A. Kazmi, T. N. Dang, I. Yaqoob et al., "Infotainment enabled smart cars: a joint communication, caching, and computation approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8408–8420, 2019.

[7] K. Poularakis and L. Tassiulas, "Code, cache and deliver on the move: a novel caching paradigm in hyper-dense small-cell networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 3, pp. 675–687, 2017.

[8] K. Thar, N. H. Tran, T. Z. Oo, and C. S. Hong, "DeepMEC: mobile edge caching using deep learning," *IEEE Access*, vol. 6, pp. 78260–78275, 2018.

[9] M. Furqan, C. Zhang, W. Yan, A. Shahid, M. Wasim, and Y. Huang, "A collaborative hotspot caching design for 5g cellular network," *IEEE Access*, vol. 6, pp. 38161–38170, 2018.

[10] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[11] L. Qiu and G. Cao, "Popularity-aware caching increases the capacity of wireless networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 1, pp. 173–187, 2020.

[12] Z. Piao, M. Peng, Y. Liu, and M. Daneshmand, "Recent advances of edge cache in radio access networks for internet of things: techniques, performances, and challenges," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 1010–1028, 2019.

[13] 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, 2019.

[14] Y. Niu, S. Gao, N. Liu, Z. Pan, and X. You, "Clustered small base stations for cache-enabled wireless networks," in *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6, Nanjing, China, 2017.

[15] J. Liao, K. Wong, Y. Zhang, Z. Zheng, and K. Yang, "Energy-efficient cooperative coded caching for heterogeneous small cell networks," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 468–473, May 2017.

[16] N. Gao, X. Xu, Y. Hou, and L. Gao, "A mobility-aware proactive caching strategy in heterogeneous ultra-dense networks," in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Istanbul, Turkey, 2019.

[17] Q. Ma, E. Chen, Z. Lin, J. Yan, Z. Yu, and W. W. Y. Ng, "Convolutional multitimescale echo state network," *IEEE Transactions on Cybernetics*, vol. 51, no. 3, pp. 1613–1625, 2021.

[18] Y. Yu, J. Cao, and J. Zhu, "An LSTM short-term solar irradiance forecasting under complicated weather conditions," *IEEE Access*, vol. 7, pp. 145651–145666, 2019.

[19] J. Zhao, S. Zhao, H. Qu, G. Ren, and Y. Shi, "Analysis and optimization of probabilistic caching in micro/millimeter wave hybrid networks with dual connectivity," *IEEE Access*, vol. 6, pp. 72372–72380, 2018.

[20] P. Cheng, C. Ma, M. Ding et al., "Localized small cell caching: a machine learning approach based on rating data," *IEEE Transactions on Communications*, vol. 67, no. 2, pp. 1663–1676, 2019.

[21] P. Xu, S. Cai, and H. Zhu, "Collaborative hierarchical caching strategy in D2D-enabled heterogeneous networks," in *2018 IEEE/CIC International Conference on Communications in China (ICCC)*, pp. 578–582, Beijing, China, 2019.

[22] Y. Zhou, Z. Zhao, R. Li, H. Zhang, and Y. Louet, "Cooperation-based probabilistic caching strategy in clustered cellular networks," *IEEE Communications Letters*, vol. 21, no. 9, pp. 2029–2032, 2017.

[23] Y. Ye, Z. Zhang, G. Yang, and M. Xiao, "Minimum cost based clustering scheme for cooperative wireless caching network with heterogeneous file preference," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, Paris, France, 2017.

[24] M. K. Somesula, R. R. Rout, and D. V. L. N. Somayajulu, "Contact duration-aware cooperative cache placement using genetic algorithm for mobile edge networks," *Computer Networks*, vol. 193, article 108062, 2021.

[25] K. Thar, T. Z. Oo, Y. K. Tun, D. H. Kim, K. T. Kim, and C. S. Hong, "A deep learning model generation framework for virtualized multi-access edge cache management," *IEEE Access*, vol. 7, pp. 62734–62749, 2019.

[26] H. Mou, Y. Liu, and L. Wang, "LSTM for mobility based content popularity prediction in wireless caching networks," in *2019 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, Waikoloa, HI, USA, 2019.

[27] M. Yan, C. A. Chan, W. Li et al., "Assessing the energy consumption of 5G wireless edge caching," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, Shanghai, China, 2019.

[28] S. Rathore, J. H. Ryu, P. K. Sharma, and J. H. Park, "Deepcachnet: a proactive caching framework based on deep learning in cellular networks," *IEEE Network*, vol. 33, no. 3, pp. 130–138, 2019.

[29] L. Yao, A. Chen, J. Deng, J. Wang, and G. Wu, "A cooperative caching scheme based on mobility prediction in vehicular content centric networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 5435–5444, 2018.

[30] D. Ren, X. Gui, K. Zhang, and J. Wu, "Mobility-aware traffic offloading via cooperative coded edge caching," *IEEE Access*, vol. 8, pp. 43427–43442, 2020.

[31] Z. Lu, X. Sun, and T. La Porta, "Cooperative data offload in opportunistic networks: from mobile devices to infrastructure," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3382–3395, 2017.

[32] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2651–2664, 2018.

[33] E. Bastug, M. Bennis, and M. Debbah, "Social and spatial proactive caching for mobile data offloading," in *2014 IEEE*

International Conference on Communications Workshops (ICC), pp. 581–586, Sydney, NSW, Australia, 2014.

[34] M. S. A. Khaleel, S. E. F. Osman, and H. A. N. Sirour, "Proposed ALFUR using intelegent agent comparing with LFU, LRU, SIZE and PCCIA cache replacement techniques," in *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, pp. 1–6, Khartoum, Sudan, 2017.

[35] D. Lee, J. Choi, J.-H. Kim et al., "LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Transactions on Computers*, vol. 50, no. 12, pp. 1352–1361, 2001.