Route-based Proactive Content Caching using Self-Attention in Hierarchical Federated Learning

SUBINA KHANAL, KYI THAR, EUI-NAM HUH†
Department of Computer Science and Engineering, Kyung Hee University, Yongin-si 17104, South Korea
†Corresponding author: Eui-Nam Huh (johnhuh@khu.ac.kr)

ABSTRACT The sheer unpredictability of content popularity, diversified user preferences and demands, and privacy concerns for data sharing all create hurdles to develop proactive content caching strategies in self-driving cars. Therefore, to address these concerns, we investigate in detail the role of proactive content caching methods in self-driving cars for improving quality-of-experience (QoE) and content retrieval cost in this work. We develop a low-complexity content popularity prediction mechanism in a hierarchical federated setting. In particular, we use a self-attention technique with an LSTM-based prediction mechanism to extract local content popularity patterns in self-driving cars. However, the local contents will not be sufficient to satisfy the passenger’s requirements. Hence, using the popular contents of other self-driving cars will solve the requirement constraint but poses some privacy issues. We use the privacy-preserving decentralized model training framework of Federated Learning (FL) to tackle this issue. Specifically, we deploy the hierarchical Federated Averaging (FedAvg) algorithm on local models obtained from self-driving cars to develop a regional and global content popularity prediction model at RSU and MBS, respectively. Extensive simulations on real-world datasets show the proposed approach improves cache space utilization by maximizing the local cache hit ratio, and further, minimizes the content retrieval cost for self-driving cars as compared with alternative methods.

INDEX TERMS proactive content caching, multi-access edge computing, federated learning, self-attention mechanism, recurrent neural network.

I. INTRODUCTION

With the advancement of self-driving car technology, the possibility of autonomous public vehicles operating on roadways is no longer a far-fetched reality. As a result, passengers in a driverless car will have a significant amount of free time with a released burden for manual navigation [1]. By utilizing the existing in-vehicle infotainment service delivered by an on-board unit (OBU) installed in self-driving cars, passengers can work or relax to spend their time while being entertained [2], [3]. In practice, to assist OBUs in delivering the freshly requested infotainment contents that are not available at the OBUs, self-driving cars can leverage the wireless network connectivity technologies (e.g., Wi-Fi or cellular networks) to reach the respective content servers. In this regard, if the content requested by the passenger is cached in the nearby Road Side Unit (RSU) with better connectivity, self-driving cars can access that content promptly from the associated RSU. If not, the self-driving cars will request those contents from other (nearby) self-driving cars, a next-hop base station (MBS), or the core cloud until found and retrieved; however, resulting in a higher content delivery delay. One way to reduce such uncertainties is to use OBU to cache the contents in advance. Because OBU can only store a portion of the content, not every passenger will receive the requested content right away. In summary, owing to OBU’s limited cache capacity, a practical, effective, proactive content caching strategy is imperative [4], [5].

In this regard, a common assumption made in the prediction schema is, the demand pattern of an infotainment content of a self-driving car is predictable to some extent [4]. Therefore, proactively caching the anticipated content requested by the passengers at the designated OBUs of the self-driving cars can massively diminish the peak load on edge networks. Then, whenever passengers send a content request to the self-driving car, the OBU of the car can retrieve the requested content directly from its cached
memory rather than downloading it from the wireless edge network. While the passengers in the self-driving car request different types of infotainment content, the OBU of the car captures and stores these patterns as historical data. Thereby, we interpret that OBU is solely responsible for identifying the request patterns of these historical content. Particularly, based on these content request patterns, OBU can determine the most popular content requested by the passengers [6], [7]. However, the popularity of content changes over time. Also, the limited number of historical contents requested by the passengers to a single OBU will not be enough to model the content popularity for proactive content caching. A solution to which, we can jointly take into account the dynamic content popularity patterns of local and other self-driving cars to develop proactive caching strategies; and hence, it ensures users Service Level Agreements (SLAs) and Quality-of-Experience (QoE) for seamless media streaming. However, on the other hand, self-driving cars will not share their data due to privacy concerns. In this regard, considering the privacy concerns of self-driving cars (clients) for sharing the historical content patterns with other self-driving cars or RSUs or MBS, employing the concept of federated learning is the best-suited idea [8]–[10]. In distributed scenarios, federated learning (FL) offers a viable solution for privacy-preserving edge intelligence. While all training data is collected at a centralized repository in the traditional machine learning setting resulting in privacy concerns for self-driving cars, FL overcomes these privacy concerns to develop a content popularity model without sharing their local data. Thus, FL reduces the cost incurred while transferring the data by dispersing the task of model training to the clients. In detail, clients execute the model training on their data locally, which typically adopts the gradient descent optimization algorithm. In an FL framework, clients do not share their raw data; instead, they send the local model parameters to the server for model aggregation, enabling a concurrent approach to build a global model while retaining their data privacy. As a result, FL provides edge intelligence while maintaining privacy by learning from decentralized data.

In this work, we propose a novel proactive content caching strategy combined with the idea of a privacy-preserving decentralized model training paradigm of FL [11] to address the issues highlighted above. In our proposed approach, the prediction model predicts the popularity pattern of the next popular content in a self-driving car using the local historical dataset. LSTM performs better on time-series data, and hence makes an obvious choice for predicting the expected popular contents. In this regard, given OBU’s limited cache size, we infer that each self-driving car contains a modest amount of local data, and it is reasonable then to adopt an FL approach to learn regional content popularity from the large distributed data instead. However, relying only on the sequential historical data of content request pattern ignores the correlated contextual information captured by the popularity count of each content, which is fundamental. To that end, the self-attention mechanism [12] allows inputs to interact with each other, i.e., the attention weights we obtain for each input helps to prioritize them accordingly. Therefore, we combine the concept of self-attention mechanism with the LSTM model in a FL setting to first capture (i) the contextual information of popularity count of each content, and (ii) the content request pattern for the model training process. Thenafter, we predict the content popularity pattern of next popular content and derive the decisions on which content to cache proactively to serve the request of the passengers. In particular, self-driving cars use the local dataset to train their model, leveraging the self-attention mechanism [12] with LSTM. Each self-driving car shares its local model parameters to the server (RSU and MBS) to get a regional and global model, respectively, using the hierarchical Federated Averaging (FedAvg) algorithm. The acquired global model is subsequently sent to the self-driving cars through associated RSUs for the next round of model training, resulting in an updated local model. The process of model training at the self-driving car and sending local model parameters to the server to downloading the global model is iterative. In this manner, the FL process continues until the global model converges to the expected level of accuracy.

The following are the major contributions of the paper:

- We proposed an efficient method that, to the best of our knowledge, is the first to combine self-attention mechanism with the LSTM model based FL technique for proactive content caching for infotainment contents in a self-driving car.
- The proposed method develops local and regional caching strategies based on the hierarchical federated learning architecture, where two aggregation levels are performed at the RSU and MBS, respectively.
- Extensive experimental evaluations on real-world datasets show that our proposed model outperforms other traditional caching strategies in terms of content retrieval cost, cache hit ratio, and user-satisfaction level.

We organize the rest of the paper as follows. In Section II, we review the related works, where we discuss various works on proactive content caching and its application. In Section III, we present our proposed system model of proactive content caching at self-driving cars using hierarchical federated learning. Here, we discuss an overview of the solution approach and present preliminaries of adopted federated learning, self-attention mechanism and LSTM model. Similarly, in Section IV, we discuss the details of our proposed approach and present a low-complexity algorithm to tackle the formulated problem. In Section V, we provide the performance evaluation of the proposed approach and compare it with other traditional approaches using real-world datasets. Finally, we conclude this work in Section VI.

II. RELATED WORKS

Several research works use deep learning methods [5], [13]–[20] in the area of proactive content caching to overcome problems mentioned in the introduction. The authors in [5]
exploit passenger’s attributes gathered through deep learning techniques to obtain caching decisions for infotainment content in self-driving cars. Likewise, in [21], a significant amount of data is used to determine content popularity. Then, strategic content is cached at base stations to reduce the cost of content retention and backhaul offloading for users. In [22], the authors suggest a self-attention-based sequential model that combines the concepts of Markov Chains (MCs) and Recurrent Neural Networks (RNNs), capturing long-term semantics (like an RNN) but making predictions based on relatively few actions (like an MC). Similarly, in [23], the authors describe a Multi-Factor Generative Adversarial Network for explicitly modelling the role of context information on the sequential recommendation. Furthermore, the authors of [24] leverage the self-attention technique to identify the item-item relationship from the historical interactions of the users. In [25], the authors use a weighted clustering strategy to optimize caching performance by using content caching oriented popularity prediction. Likewise, the authors in [26] propose a video content collaborative caching strategy for minimizing user response latency and easing backhaul stress in the cloud-edge cooperative setting. Authors in [27] propose a probabilistic dynamic factor analysis model to explain content demands for real-world time-varying scenarios to track and anticipate the evolution of content popularity. In [28], the authors consider a learning-based content caching strategy in which each small BS learns space-time popularity dynamics in a distributed manner by utilizing a multi-armed bandit learning agent.

In addition, the authors in [29] propose an LSTM Encoder-Decoder model for content caching, as well as a caching policy component that takes into consideration the expected object information to make smart caching decisions. The authors in [30] propose a social-aware vehicular edge caching mechanism that adaptively defines the cache capability of roadside units (RSUs) and smart vehicles based on similarity of the preference of users and accessibility of services. They incorporate digital twin technology to map the edge caching system into virtual space, making it easier to build the social relation model. Moreover, [31] employs a deep spatiotemporal residual network to estimate future vehicle service requirements to address the problem of providing high-quality services while ensuring resource efficiency through edge content caching. To examine the implications of restricted radio and storage resources, the authors in [32] offer a systematic and coherent content caching and delivery framework for the cloud augmented mobile edge with hierarchical radio access points. Likewise, in [33], the authors...
presented a developing learning-based content caching strategy in edge networks which flexibly analyzes time-varying content popularity and when the cache is filled, it determines which contents should be refreshed. Besides, in [34], authors present an incentive caching policy in which networks incorporate preferences of the users and group mobility together with Device-to-Device (D2D) communication for solving the caching problem in mobile devices. Also, authors in [18] use the echo state network and the long short-term memory to predict the content popularity, along with the mobility of the users in the D2D communication technology. In the same manner, D2D offloading reduces the cellular network traffic congestion, yet mobile nodes may refuse to participate in the offloading process. As a result, authors in [35] propose an incentive-driven Deep Q Network-based solution that uses reverse auction as an incentive mechanism to reduce content service provider costs.

The majority of the recent research and the ones mentioned earlier are based on centralized content caching. On the other hand, the conventional centralized system is sensitive to privacy, and not all self-driving cars may want to share their raw data with the core cloud or server (RSU and MBS). Even though some self-driving cars share their local data, the model obtained after model training may only perform efficiently for a limited number of self-driving cars, resulting in generalization constraints. That means the obtained local model may not be efficient enough to manage proactive content caching that fulfills the infotainment content request of every passenger of a self-driving car. Therefore, to address these concerns, we developed a proactive content caching scheme by leveraging the privacy-preserving, decentralized model training paradigm of FL [11], [36] in this work. In our proposed approach, self-driving cars iteratively train local learning models on their dataset using a self-attention mechanism with LSTM models. Self-driving cars then share the obtained local model parameters to the RSU and MBS to get a generalized regional and global model.

III. SYSTEM MODEL

In Fig. 1, we show a basic pictorial workflow of our proposed system model. Here, we consider that self-driving cars are public vehicles that travel on the same local routes on a daily basis, which is an intuitive assumption to make for the public transports. By local routes, we mean the routes are within certain areas. In this perspective, we use the term “route-based” throughout the manuscript. To that end, several self-driving cars share these routes regularly and can communicate with each other via RSUs. Considerably, in these local routes, it becomes convenient to manage and control historical data in self-driving cars and RSUs to develop proactive content caching strategies. Also, we assume that the MBS consists of a multi-access edge computing (MEC) server with a reliable backhaul link to the cloud server. In particular, a set of cache-enabled RSUs $\mathcal{R}$ of $|\mathcal{R}| = R$ connected to a MBS $\mathcal{M}$ via radio links as well as a set of self-driving cars $\mathcal{V}_r$ of $|\mathcal{V}_r| = V_r$ linked with RSU $r$ at time $t$ are defined. In addition, a set of contents $\mathcal{K}$ of $|\mathcal{K}| = K$ is also defined, where the size of each content $k \in \mathcal{K}$ is denoted as $z_k$. In the proposed model, OBUs of the self-driving car function as a mediator to fulfill the passenger’s demand by serving them the infotainment contents they requested. However, because of the limited cache space of the OBUs, not every content request of the passengers is immediately satisfied. As a result, the OBU will send the request for missing infotainment content to the associated RSUs and, if available, retrieve the requested infotainment content from the RSU to meet the passenger’s request. Because the RSU only stores the contents of a specific area, not all contents requested by the OBUs will be satisfied; as a result, RSU will fetch the requested contents from MBS and serve the OBUs$^1$

Let $C_v$ denotes the cache capacity of self-driving car $v, \forall v \in \mathcal{V}$ and $x_{vk} \in \{0, 1\}$ represents the binary variable such that

$$x_{vk} = \begin{cases} 0, & \text{if content } k \text{ is not cached at self-driving car } v, \\ 1, & \text{otherwise.} \end{cases} \tag{1}$$

In this regard, we consider $x_{vk} = 0$ a cache miss. Consequently, the self-driving car $v$ will incur the content retrieval cost denoted as $\delta_k^v$ to serve the passengers while retrieving requested contents from the associated RSU. In particular, we define the content retrieval cost as the backhaul usage function. That is, the content retrieval cost $\delta_k^v$ for a missed content request $v$ at a self-driving car $v$, is the function of the content size $z_k$ and the available link capacity $\Omega_k^v$ between the self-driving car $v$ and the RSU, i.e., $\delta_k^v = z_k/\Omega_k^v$. Mathematically, we define $\Omega_k^v$ as the standard Shannon rate function, which is dependent on the available wireless resources, such as the transmit power and allocated bandwidth, and the channel gain, i.e., $\Omega_k^v = B_k^v \log_2(1 + \frac{\rho^v|h|^2}{n_0^v})$, where $B_k^v$ is the available bandwidth to retrieve content $k$ from RSU, $\rho^v$ is the transmission power, $|h|^2$ is the channel gain, and $n_0^v$ is the Gaussian noise power density [4], [5]. Here, we consider a cache hit if $x_{vk} = 1$, in which the content requested by the passengers’ content would be delivered directly from the OBUs of the self-driving car. The cache capacity constraint at each self-driving car $v \in \mathcal{V}$ at any time $t$ can thus be explicitly defined as follows:

$$\sum_{k \in \mathcal{K}} x_{vk}(t)z_k \leq C_v, \forall v \in \mathcal{V}. \tag{2}$$

Similarly, for RSU $r, \forall r \in \mathcal{R}$, we define the cache capacity as $C_r$ and $y_{rk} \in \{0, 1\}$ represents the binary variable such that

$$y_{rk} = \begin{cases} 0, & \text{if content } k \text{ is not cached at RSU } r, \\ 1, & \text{otherwise.} \end{cases} \tag{3}$$

In the same manner, for RSU $r$, we consider a cache miss when $y_{rk} = 0$. Thus, RSU will incur an additional content

$^1$Note that we consider the self-driving cars are associated with the RSUs. Thus, the self-driving cars cannot fetch (request) the contents from the MBS. Instead, the link between RSUs and MBS readily serves as a backhaul for delivering the requested contents, similar to [5].
retrieval cost denoted as $\delta_k^r$ while fetching the requested contents either from the content server located at the remote cloud or the MBS to serve the self-driving cars. Similarly, this delay cost depends on the content size $z_k$ and the available link capacity between RSU $r$ and the MBS denoted as $\Omega_k^r$, where $\Omega_k^r = B_k^r \log_2(1 + \rho^r h^r r) r^2$, where $B_k^r$ is the available bandwidth, $\rho^r$ is the transmission power, $h^r$ is the channel gain and $n_0^r$ is the noise power density between RSU and MBS. Likewise, we consider a cache hit at the RSU $r$ if $y_k^r = 1$, whereby the content requested content by the self-driving car would be delivered directly from the RSU. The cache capacity constraint at each RSU $r \in \mathcal{R}$ at any time $t$ thus can be explicitly defined as

$$\sum_{k \in \mathcal{K}} y_k^r(t) z_k \leq C_r, \forall r \in \mathcal{R}. \quad (4)$$

At each self-driving car $v \in \mathcal{V}$ and RSU $r \in \mathcal{R}$, the primary purpose of proactive caching, is to cache a subset of contents for a time window $N$. Thus, it jointly improves the cache hit ratio, minimizes content retrieval costs, and optimizes cache utilization. Therefore, the network-wide delay associated with the cache miss at OBUs of the self-driving car and RSU for the next $t + N$ time slot is defined as

$$\phi(t + N) = \frac{R}{N} \sum_{r=1}^{N} \sum_{v=1}^{V} \sum_{k=1}^{K} \left(1 - x_k^r(t)\right) \delta_k^r + \left(1 - y_k^r(t)\right) \beta_k^r(t + N), \quad (5)$$

where $\beta_k^r(t + N)$ denotes the number of request counts for content $k$ at a $N$ time spaced window by the self-driving car $v$. In (5), we observe the overall network delay associated with the contents $k \in \mathcal{K}$ requested (either by the passengers to a self-driving car or self-driving car to the RSU) depends on the caching strategy $x_k^r(t)$ and $y_k^r(t)$, which is defined by self-driving car $v$ and RSU $r$, respectively, and the number of future request counts $\beta_k^r(t + N)$, which is an unknown quantity. Here we can estimate $\beta_k^r(t + N)$ using the historical information about the prior content request counts, or we can leverage the proactive content caching with a federated learning approach to predict $\beta_k^r(t + N)$, which will be discussed in the Section IV.

Therefore, we formulate our optimization problem to minimize the network-wide latency during content request for the time window $N$ as

$$\mathbf{P} : \text{minimize } x, y \quad \frac{1}{N} \sum_{t=1}^{N} \phi(t + N)$$

subject to

$$\sum_{k \in \mathcal{K}} x_k^r(t) z_k \leq C_v, \forall v \in \mathcal{V}, \forall t \in \mathcal{T}, \quad (6a)$$

$$\sum_{k \in \mathcal{K}} y_k^r(t) z_k \leq C_r, \forall r \in \mathcal{R}, \forall t \in \mathcal{T}, \quad (6b)$$

$$\phi(x_k^r(t)) + \phi(y_k^r(t)) \leq \phi_{th}, \forall r \in \mathcal{R}, \forall t \in \mathcal{T}, \quad (6c)$$

$$x_k^r(t) \in \{0, 1\}, \forall k \in \mathcal{K}, \forall v \in \mathcal{V}, \forall t \in \mathcal{T}, \quad (6d)$$

$$y_k^r(t) \in \{0, 1\}, \forall k \in \mathcal{K}, \forall v \in \mathcal{V}, \forall t \in \mathcal{T}, \quad (6e)$$

where $x$ and $y$ are the matrix that is used for mapping contents caching strategies for each self-driving car and RSU, respectively, over the time slots; $\phi(x_k^r(t))$ and $\phi(y_k^r(t))$ are shorthand representation of delays to quantify total network-wide latency requirements following (5), constraints (6b) and (6c) denotes the cache capacity at each self-driving car and RSU, respectively; constraint (6d) denotes the constraint imposed on the content retrieval cost to define the QoE for passengers when requesting contents to the self-driving cars. We consider the QoE in terms of content retrieval cost at the associated self-driving cars and the RSUs, and for all requested contents. In principle, the metric of QoE is reflected in relation to the caching strategies, as highlighted in the problem formulation $\mathbf{P}$. This translates constraint (6d) as the constraint imposed on the content retrieval cost to define the QoE for passengers when requesting contents to the self-driving cars. Therein, the term QoE equivalently translates into the proportion of passengers served by the self-driving car in the formulation, similar to [37]. We also observe that the MBS can accurately determine the content retrieval costs $\delta_k^r$ and $\delta_k^r$ in the downlink. Thus it satisfies the objective of minimizing (6) which usually entails first determining the contents request count and then applying the proactive content caching decision at both the self-driving cars and RSUs with their corresponding cache capacity. However, as stated previously, the optimization problem (6) is difficult to solve for two specific reasons: (i) the request count of infotainment contents $\beta_k^r(t + N)$ is unknown, and (ii) the decision variables are coupling in both the objective function and the constraints, while (6c) and (6f) are combinatorial in nature; thus making the problem difficult to solve in the polynomial time, i.e., NP-hard, as shown in [38]. Furthermore, the coupled constraints are stochastic, making it challenging to determine the optimal caching strategy beforehand.

Thus, to tackle the formulated optimization problem, we propose a decentralized, self-attention mechanism with an LSTM model-based hierarchical FL solution. Particularly, we decouple the problem to evaluate proactive content caching strategy at each RSU-level and incorporate a decentralized learning-based model training approach of FL
with self-driving cars (as clients) and associated RSUs (as an aggregator) in a hierarchical setting (with MBS aggregating the models sent by the RSUs) to derive networking-wide caching strategy. In this way, we obtain a low-complexity solution to the optimization problem \( P \) than the optimal solution.

In our considered system model, passengers of the self-driving cars send the infotainment contents request to the OBU installed in self-driving cars. These content requests are stored in OBU as historical data, as illustrated in Fig. 1. Self-driving cars use these historical data for the model training process exploiting the self-attention mechanism with the LSTM model. After the FL training process, we get a local model (personalized model) for each self-driving car, which predicts the popular content for that self-driving car and proactively cache those contents in OBU. After a set number of epochs of the model training process, self-driving cars upload the local model parameters to the nearby RSUs. Each RSU waits for the model parameters until the time \( t \) and starts model aggregation, creating a regional model (generalized model), predicting the popular content and proactively cache those contents in RSU. Each RSU then uploads these regional model parameters to MBS, where MBS again perform model aggregation, creating a global model (generalized model), i.e., a typical hierarchical FL setting. All RSUs then download this global model and pass it to all the randomly selected self-driving cars. This FL process is repeated until the global model reaches the desired accuracy.

In the following subsection, we explain the preliminaries used. We begin by discussing the self-attention mechanism and the processes that are involved in our proposed approach. After that, we go over the LSTM model and explain its use in our proposed approach. Finally, we describe the federated learning process and its benefits in our proposed approach.

A. PRELIMINARIES: SELF-ATTENTION MECHANISM

The concept of self-attention was first presented in [12]. Self-attention is an attention process that entails calculating an interpretation of a single sequence by combining several distinct parts of it. According to the generalized definition, each word’s embedding should have three separate vectors related to it: Key, Query, and Value. Matrix multiplications are a simple way to get these vectors. Whenever we need to calculate the attention of a target word to the input embedding, we should calculate a matching score using the target’s Query and the input’s Key, and then utilize these matching scores as the weights of the Value vectors during summation. There are three phases to calculating attention. To obtain a weight, we first compute the similarity between the query and each key. The dot product, splice, detector, and other similarity functions are frequently employed. The second step is to normalize these weights with a softmax function, then weigh these weights against the relevant values to get the final attention.

Here, the three vectors of the model: key \( K \), query \( Q \), and value \( V \) indicate the temporal content relationship sequences, i.e., the embedding representations, defined as \( X_t \) with a embedding vector of dimension \( d_o \) for each time step \( t \). We adopt ReLU as an activation function to link key and query in the same hidden layer space and with shared parameters, similar to [24], where key \( K \) and query \( Q \) can be calculated as

\[
K' = ReLU(X_t \times W_K),
\]

\[
Q' = ReLU(X_t \times W_Q),
\]

where \( W_K \in \mathbb{R}^{d_o \times d_o} \) and \( W_Q \in \mathbb{R}^{d_o \times d_o} \) are the weights of the two non-linear layers; hence \( W_K = W_Q \).

The self-attention mechanism primarily reflects the short-term dependency of the content’s sequential pattern. The scaled dot attention technique is used to calculate attention in the self-attention layer. Here, we have the scaling factor as \( \frac{1}{d_o} \), where \( d_o \) is the dimension of keys and queries, and the attention weight matrix as \( a_n \), which is calculated as follows:

\[
SelfAttention(Q, K, V) = \text{softmax}\left(\frac{Q'K'^T}{\sqrt{d_k}}\right)V.
\]

In our proposed approach, we use the attention weights obtained from the self-attention mechanism as the input of the LSTM cell of the LSTM model described below.

B. LSTM MODEL

LSTM is the type of Recurrent Neural Network (RNN) that can learn patterns from data with lengthy dependent periods [39]. The primary purpose of RNNs is to capture the dynamic behaviour of sequential and synchronous input data. As shown in Fig. 2, the LSTM cell consists of three gates: forget gate \( (f_{io}) \), input gate \( (i_{io}) \), and output gate \( (O_{p_l}) \). Here, the state information is stored by the memory cell \( CS_t \), which is accessed, written to, and cleared by various self-parameterized controlling gates. If the input gate is on, the information from each new input will be collected and stored in the cell. And if the forget gate is activated, the previous cell status \( CS_{t-1} \) may be “lost/forgotten” throughout this process. The output gate then determines whether the latest cell output \( CS'_t \) will be transmitted to the final state \( h_t \).

We can mathematically represent \( f_{io} \) as

\[
f_{io} = \sigma(W_f \cdot [h_{t-1}, d_t] + b_f),
\]

where \( h_{t-1} \) and \( d_t \) are the inputs, \( W_f \) is the weight matrix to be learned, and \( \sigma(\cdot) \) is the sigmoid activation function and \( i_{io} \) as

\[
i_{io} = \sigma(W_i \cdot [h_{t-1}, d_t] + b_i).
\]

The new cell state \( CS'_t \) can be obtained as

\[
CS'_t = \tanh(W_c \cdot [h_{t-1}, d_t] + b_c).
\]

Then, the memory cell \( CS_t \) can be updated as

\[
CS_t = f_{io} \ast CS_{t-1} + i_{io} \ast CS'_t.
\]

Next, we have \( O_{p_l} \) represented as

\[
O_{p_l} = \sigma(W_o \cdot [h_{t-1}, d_t] + b_o),
\]

\[
VOLUME \ .
\]
and finally, the final state $h_t$ can be obtained as

$$h_t = Op_t \ast \tanh(CS_t). \quad (9)$$

In our proposed approach, we first feed the attention weights acquired from the self-attention mechanism into the historical content of each self-driving car. We then pass these sequential historical contents along with their popularity counts as an input to the LSTM model, resulting in a local model of content popularity. Finally, using the local dataset of the self-driving car, the obtained local model predicts the popularity pattern of the next popular content. The OBU of a self-driving car can then make a proactive content caching decision based on the popularity pattern, delivering such infotainment content to passengers whenever they request it, to reduce the excessive delivery latency.

C. FEDERATED LEARNING

The term Federated Learning (FL) refers to a decentralized machine learning method that allows training on an extensive range of decentralized data stored on various IoT (Internet-of-Things) devices [9], [40], [41]. Traditional machine learning collects all training data in a centralized repository owing to privacy issues, whereas federated learning overcomes these privacy concerns. Clients perform model training on their local data in an FL setting adopting the gradient descent optimization process. The general optimization problem is as follows [8], [36]:

$$F(w) = \frac{1}{V} \sum_{v=1}^{V} F_v(w); F_v(w) = \frac{1}{n_v} \sum_{(x_i, y_i) \in D_v} l_v(w), \quad (10)$$

where $w$ is the model parameter, $V$ is the total number of clients, i.e., self-driving cars, $l_v(w)$ is the loss of prediction for some input-output pairs $(x_i, y_i)$ in the training data samples $x_i$ and labels $y_i$, respectively, $D_v$ is the set of indices of data points on client $v$ with $n_v = |D_v|$ and $D$ the total data samples in the considered region. Each client $v$ updates local model $w'_{v}$ following stochastic gradient descent to solve (10) in a distributed manner [8].

However, in our proposed approach, we employ a hierarchical federated learning strategy in which the Federated Averaging (FedAvg) [8] procedure to solve (10) is enforced on two tiers. First, we implement the FedAvg process in RSU (tier-one) to generate a regional model utilizing the local models obtained from the associated self-driving cars. Then, using the regional model obtained from the RSUs, we apply the model aggregation at the MBS (tier-two) to create a global model. In doing so, we incorporate a more extensive set of distributed data samples to obtain a generalized view of the popular contents in each area where the RSUs and self-driving cars are associated.

IV. PROPOSED SOLUTION APPROACH

This section proposes and discusses the deployment design of a self-attention mechanism with the LSTM model in route-based proactive content caching using hierarchical FL. Firstly, we present the steps required in predicting the popularity count of content in our proposed approach, which employs a self-attention mechanism with the LSTM model. Next, we explain the process of how will self-driving cars and RSUs proactively cache the predicted popular content exploiting the regional and global model acquired after hierarchical federated averaging.

In Fig. 2, we show an illustration of the proposed mechanism for proactive content caching at the self-driving cars following the system model, as in Fig 1. The OBU installed in the self-driving cars (clients) stores historical contents in it. This sequential historical content has a definite pattern in the self-driving cars (clients) stores historical contents in

![FIGURE 2: Client Side Model Training.](image-url)
Algorithm 1 Proactive Content Caching Strategy using Hierarchical Federated Averaging.

1: MBS executes:
2: Initialization: initialize model parameter $w_0^g$;
3: Output: global model $w^g$, content popularity count;
4: $R_t$: total number of RSUs associated with the MBS at time $t$;
5: for each round $t = 1, 2, ...$ do
6: for $\forall r \in R_t$ in parallel do
7: $w_{t+1}^r \leftarrow $ REGIONALUPDATE$(r, w^g_t)$;
8: end for
9: $w_{t+1}^g \leftarrow \sum_{r=1}^{R_t} \frac{w_{t+1}^r}{R_t}$;
10: end for
11: Execute Algorithm 2 and Algorithm 3;
12: end for
13: REGIONALUPDATE$(r, w)$: \texttt{// Implement regional model updates on RSUs $r$.}
14: $V_t^r$: total number of self-driving cars associated with the RSU $r$ at time $t$;
15: for each round $t = 1, 2, ...$ do
16: for $\forall v \in V_t^r$ in parallel do
17: $w_{t+1}^v \leftarrow $ LOCALUPDATE$(v, w^g_t)$;
18: end for
19: $w_{t+1}^v \leftarrow \sum_{v=1}^{V_t^r} \frac{w_{t+1}^v}{V_t^r}$;
20: return $w_{t+1}^v$ to MBS;
21: end for
22: end for
23: LOCALUPDATE$(v, w^g)$: \texttt{// Implement local model updates on self-driving car $v$.}
24: for each local epoch $E$ do
25: $B$: is the local minibatch size used for the self-driving car updates;
26: $n_v$: is the training data;
27: $\eta$: is the learning rate;
28: $B \leftarrow $ (split $n_v$ into mini-batches of size $B$);
29: for $b \in B$ do
30: $w_{t+1}^v \leftarrow w_{t+1}^v - \eta \nabla \ell (w^g_t; b)$;
31: end for
32: return $w_{t+1}^v$ to RSU;
33: end for

Our proposed approach, popular contents have the maximum number of request counts; in our case, popularity counts. This popularity count determines the subset of contents required to be cached in each client. First, for each client, we use sequential historical content, and its popularity counts as an input feature to train our model using a self-attention mechanism with the LSTM model to obtain a local model. This local model predicts the popularity pattern of the next popular content, which each client’s OBU may choose to cache. However, caching merely popular content will not suffice because the contents requested depend on passengers’ preferences that change over time. Furthermore, due to the significant mobility of clients and passengers, it is not viable to leverage content popularity patterns as a critical parameter in a particular OBU for designing proactive content caching strategies at the client-side. As a result, we will have to create a diversified observation of popular content from other clients’ OBUs. Clients, on the other hand, may not share their content request pattern owing to privacy concerns. As a result, we employ the FL idea, as previously discussed in the preliminaries of section III, to share the local model parameters of clients for a broad perspective of content request patterns. Secondly, after model training, each selected client sends the derived local model parameters to the RSU for model aggregation following the FedAvg algorithm to obtain a regional model. Finally, the regional model parameters are sent to the MBS for model aggregation using the FedAvg algorithm to get a global model. This, global model is sent back to the associated RSUs, which later forward the global model to each selected client for further local model updates. Hence, the FL process is repeated until the global model converges to desired model accuracy.

Each selected client updates their local model after receiving the global model from the RSU to predict the popularity pattern of the next popular content to be cached proactively. To that end, each client may get a generalized perspective of popular infotainment content, which they can choose to cache proactively if needed. Nevertheless, each selected clients make sure if the predicted contents are locally available in the OBU or not. Suppose the predicted content is not available in the OBU. In that case, it sends a request to nearby self-driving cars or the associated RSUs to proactively cache those contents so that OBU can serve the passenger’s requests instantly. And if the predicted content is already available in the OBU, then the content requested by passengers gets served immediately. Similarly, RSUs also use the obtained regional models to predict the popularity pattern of the next popular content to cache proactively. Likewise, each RSU checks if the predicted contents are available in their cached memory or not. If the predicted contents are available, it

Algorithm 2 Local Proactive Content Caching Strategy in Self-Driving Cars.

$V_t^r$: total number of self-driving cars associated with the RSU at time $t$;

2: $w^g$: Final global model;

Output: local content popularity count, top-$n$ contents to be proactively cached;

4: for $\forall v \in V_t^r$ do
5: Employ $w^g$ to obtain local content popularity count;
6: for each top-$n$ content prediction made by the local model do
7: if predicted contents in OBU are not available then
8: retrieve contents from RSU and cache them; end if
10: end for

Algorithm 3 Regional Proactive Content Caching Strategy in RSU.

1: \( R_t \): total number of RSUs associated with the MBS at time \( t \);
2: \( w^r \): Final regional model;
3: Output: regional content popularity count, top-\( n \) contents to be proactively cached;
4: for \( \forall r \in R_t \) do
5: 
6: 
7: end if
8: end for
9: end for

delivers the content requested by the self-driving cars immediately, whenever they ask for it. Else, it sends a request to the MBS to proactively cache those contents to serve the request of self-driving cars instantly.

Algorithm 1 presents the details of the proposed decentralized, self-attention mechanism with the LSTM model-based hierarchical FL solution approach to derive a low-complexity solution to the optimization problem \( P \), as discussed. Therein, Algorithm 2 and Algorithm 3 are executed to obtain the network-wide caching strategy while satisfying QoE constraints for passengers. In particular, we decompose the original problem to evaluate proactive content caching strategy at each self-driving car and RSU, following a decentralized learning-based model training approach of FL with self-driving cars and associated RSUs in a hierarchical setting. In doing so, the self-attention mechanism with the LSTM models is used to predict the content popularity pattern of next popular content. To that end, executing Algorithm 1, we derive decision variables for the content caching strategies as the solution to the formulated optimization problem \( P \). We used Tensorflow [42] for the implementation of the proposed algorithms.

To that end, following the proposed approach of self-attention mechanism with LSTM model in the FL setting, we note, the computational complexity of learning per weight and time of the LSTM model is \( O(1) \) with the SGD, while \( O(\log(d_n)) \) memory complexity with \( d_n \) temporal distance [12] per round of global iteration.

V. SIMULATION RESULTS

The results of experiments applied to analyze the performance of the proposed approach are presented in this section. For this, first of all, we divide the areas into the set of area \( \{a_1, ..., a_n\} \) as shown in Fig. 3. For each area, we randomly select the self-driving cars (clients) \( \{v_1, v_2, ..., v_n\} \) and start training the model by using the historical data stored in the OBU of each self-driving car. For a practical scenario, we consider a small network, where we divide the routes of self-driving cars into two areas, Area 1 and Area 2, with one MBS and two RSUs each, RSU X and RSU Y [5].

We show the summary of key parameters used while performing the experiments in Table 1. We use two features of the dataset which has the highest feature score; Fig. 4 shows the level of importance of each feature, as feature scores, used as input from the data samples. The higher the score more important or relevant is the feature towards the output variable. For the extraction of top features from the dataset, we use Extra Tree Classifier [43]. In fact, Tree-Based Classifiers include an inbuilt class called Feature Importance. As we observe in Fig. 4, the ‘rating’ feature has the highest feature score. However, we did not use this feature as the rating is of little to no significant feature when determining the popularity of content for proactive content caching, but for the recommendation systems. In this regard, we use features ‘timestamp’ and ‘title of the content’ to assess the ‘popularity count’ of the content, which is the third feature used in our experiments. Similarly, we use the well-accepted

Figure 3: Area Distribution

Figure 4: Level of importance of each feature based on feature scores.
algorithms as follows. We compare our proposed approach with other alternative schemes. Proposed hierarchical FL enabled proactive content caching methods used as baselines to demonstrate the efficacy of our graphic information is provided voluntarily by the users. MovieLens users who joined MovieLens in 2000. All demo-
mous ratings of approximately 3,900 movies made by 6,040 proposed approach [49], which contain 1,002,209 anony-
ms simulation experiments, to evaluate the performance of our as a loss function as it is the most natural and unambiguous best fit for our scenario [47]. Likewise, we employ ‘MAE’ a probability between 0 and 1, the sigmoid function is the activation function’ in the attention layer of the model as limited range of popular contents [46]. We utilize a ‘sigmoid ‘attention width’ in Table 1 controls the width of the local contents from a content server, either from the RSU, if ‘Adam’ optimizer during the model training process in self-

driving cars, as it is easy to implement compared to other model optimizers. Also, it requires very little memory, which makes it computationally efficient [44], [45]. The parameter ‘attention width’ in Table 1 controls the width of the local contents of a self-driving car, where it tend to focus on a limited range of popular contents [46]. We utilize a ‘sigmoid activation function’ in the attention layer of the model as the self-attention mechanism predicts the attention weights of the input in the form of probability. Because anything has a probability between 0 and 1, the sigmoid function is the best fit for our scenario [47]. Likewise, we employ ‘MAE’ as a loss function as it is the most natural and unambiguous measure of average error magnitude [48].

A. DATASET
We use the real-world movielens 1M (ml-1m) dataset for simulation experiments, to evaluate the performance of our proposed approach [49], which contain 1,002,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000. All demographic information is provided voluntarily by the users.

In the following subsection, we discuss the alternative methods used as baselines to demonstrate the efficacy of our proposed hierarchical FL enabled proactive content caching scheme.

B. ALTERNATIVE METHODS
We compare our proposed approach with other alternative algorithms as follows.

1) Distributed Collaborative Learning (DCoL): DCoL develops a proactive content caching technique at the network edge leveraging item-based collaborative filtering and long short-term memory (LSTM). DCoL develops a proactive content caching technique at the network edge leveraging item-based collaborative filtering and long short-term memory (LSTM). It integrates the local prediction models to build a regional content popularity database [13].

2) Bidirectional Long Short-term Memory (BI-LSTM): BI-LSTM learns the input sequence information of the neural network in both forward (past to future) and backward (future to past) directions, concatenating and embedding both interpretations in the hidden states [50].

3) Long Short-term Memory (LSTM only): LSTM is a type of RNN that is specifically designed to avoid long dependency periods. The model’s ability to remember information for lengthy periods is the fundamental characteristic of LSTM [39]. It is used to predict the next popular contents to be cached.

4) Item-based Collaborative Filtering (IBCF): Item-based collaborative filtering seeks out similar items based on user preferences derived from information about items they’ve already liked or positively interacted with. The algorithm then suggests items depending on the users’ preferences, i.e., previously consumed/requested content [51].

5) Least Recently Used (LRU): When making a caching decision, this approach eliminates the items that have been used the least recently. It always keeps a record of which item was retrieved from the database. The LRU technique is a very tedious and time-consuming process as it needs to discard the least recently used item [52], [53].

6) Random Replacement (RR): This strategy picks a candidate item randomly and deletes it as needed to clear up cache space. This approach does not require any previous content access information. Every other item in this procedure is essentially substituted with the same probability [52], [53].

C. PERFORMANCE EVALUATION
The results of Fig. 5, and Fig. 6 respectively, shows that our proposed approach performs better than the alternative baselines: DCoL, BI-LSTM, LSTM only, IBCF, LRU and RR in terms of three key metrics: (a) Cache Hit Ratio, (b) Content Retrieval Cost, and (c) User Satisfaction Level, for different cache sizes at the client’s OBU of RSU X and RSU Y, respectively. In this work, cache hit ratio is the total number of content requests served, either at the self-driving cars, or at the RSUs, respectively. Similarly, content retrieval cost is the latency incurred while fetching the contents from a content server, either from the RSU, if the content is available at the RSU, or otherwise, from the MBS to the OBU of self-driving cars, respectively. Finally, user satisfaction level refers to the number of satisfied passengers, i.e., the content request served following the proposed proactive content caching strategy. Clients can build efficient content caching decisions based on a global model and local data knowledge, resulting in reduced content retrieval costs than other alternative content caching techniques. Whereas DCoL offers a distributed, low-complexity solution to derive content caching strategy but does not consider important features when developing content popularity models; thus,
it performs poorly. In this regard, we also note that LSTM and BI-LSTM work well than IBCF, LRU and RR; however, LSTM performs poor than the proposed method and DCoL. Similarly, BI-LSTM also performs poor than the proposed method but performs hand-in-hand with DCoL. This is due to the fact that both the LSTM and BI-LSTM models work well when data samples have a specific pattern but cannot capture long-term, dynamic content popularity patterns required in a practical scenario. Table 2 summarizes the improvement of our proposed approach in the case of both cache hit efficiency and content retrieval cost. We observe a significant gain, respectively, of 64% to 6.3% and 46% to 10% in terms of the cache hit ratio and content retrieval cost against naive caching strategy (RR) and the competitive distributed algorithm (DCoL).

The results in Fig. 7 shows the performance of our proposed approach for the number of communication rounds in the FL process. Here, “communication rounds” define the total number of global iterations defined in a Hierarchical Federated Learning (HFL) process, similar to [54]. In particular, one communication round indicates the single global iteration between clients (self-driving cars), the RSU, and server.

**TABLE 2**: Comparison results of cache hit ratio and content retrieval cost

| Traditional Methods | Proposed: Improvements (%) |
|---------------------|----------------------------|
|                     | Cache Hit Ratio | Content Retrieval Cost |
| DCoL                | 6.3            | 10                      |
| BI-LSTM             | 7.81           | 11.26                   |
| LSTM only           | 11.22          | 22.2                    |
| IBCF                | 17             | 25                      |
| LRU                 | 33             | 37.78                   |
| RR                  | 64             | 46                      |
and the server located at the MBS during the global model training. For each client’s OBU, we evaluated the cache hit ratio using a 300MB cache size. We set the local model training epoch for each client to 20. We observe that the cache hit ratio increases as the number of communication rounds increases, indicating that multiple communication rounds improve global model performance with the obtained value of MAE 0.004.

Fig. 8 and Fig. 9 show the performance of regional and global models at the client level, respectively. As the communication round increases between clients and RSUs, and RSUs and MBS, the cache hit ratio also increases. This is intuitive as the algorithm exploits improved regional and global models to determine the proactive content caching strategy in the client level. In RSU X, we use five different clients for the model training process, and for RSU Y, we use four clients in total. In this regard, Fig. 10 illustrates the evolution of movie prediction over time, i.e., for each communication round in the FL process, the obtained local model after the model training process predicts a different set of movies for the self-driving car. Here, we also observe the local model predicts distinct popular contents for different communication rounds.

Similarly, Fig. 11 shows the average performance evaluation of regional models on distributed RSUs in terms of a cache hit ratio. This result demonstrates the efficacy of the proposed approach in improving content delivery time via improved proactive caching strategy at the RSUs following the distributed model training. In Fig. 12 we analyze the proposed approach for varying vehicle density in the range [2,10] for fixed, 20 local epoch and communication round, respectively, in terms of the cache hit ratio. In our case, we refer to vehicle density as the number of clients participating in the FL process. As vehicle density increases, so does the cache hit ratio because the increased participation of clients improves global model accuracy, lowering the content prediction error. As shown in Fig. 13, the cache hit ratio increases with the number of local model training epochs for a certain number of communication rounds, which is a common feature of the FL setting. This is because when the number of local epochs increases, the accuracy of the local model increases as well, increasing the cache hit ratio due to the high-quality global model.

Finally, we estimate average user satisfaction for varied cache sizes of the client’s OBU in Fig. 14. We randomly assigned nine clients to two RSUs for evaluating the satisfaction measurement using the proposed approach. User-satisfaction level captures the varied preferences of passengers for infotainment content. Here, we calculate the user satisfaction ratio by dividing the total number of requests served for passengers by the total number of requests. Here, we observe that increasing cache sizes boost user satisfaction since the OBU can proactively store more content due to sufficient storage.
VOLUME ,

VI. CONCLUSION
Proactive content caching is a potential strategy for dealing with the exponential growth of content requests from diverse passengers in self-driving cars, as OBUs frequently cache content for fast and repeated access. However, due to the limited cache capacity of OBU and the dynamic arrival of passengers’ requests for random contents, OBUs cannot effectively decide which contents to cache. We investigated the subject of proactive content caching in self-driving cars in this paper. As a result, we’ve presented a self-attention technique for proactive content caching in the self-driving cars using LSTM models in a hierarchical FL setup. We primarily used interleaved self-attention and LSTM-based mechanisms to predict local content popularity patterns as local models. Then leveraged the FL process to send these model parameters to the RSUs (and the MBS) for creating regional (and global) models via model aggregation at two different levels, namely at RSUs and the MBS. Compared to existing caching algorithms, extensive simulation results shows the proposed approach has substantial improvement in the cache hit ratio, reduce content retrieval cost, and optimize cache utilization.

REFERENCES
[1] M. Daily, S. Medasani, R. Behringer, and M. Trivedi, “Self-driving cars,” Computer, vol. 50, no. 12, pp. 18–23, 2017.
[2] G. Jarvis, “Keeping entertained in the autonomous vehicle,” TU-Automotive Detroit, Detroit, MI, USA, Tech. Rep. TU Auto-[Tele. Jarvis. 2017.12. 08], 2018.
[3] S. A. Kazmi, T. N. Dang, I. Yaqoob, A. Ndikumana, E. Ahmed, R. Hussain, and C. S. Hong, “Infotainment enabled smart cars: A joint communication, caching, and computation approach,” IEEE Transactions on Vehicular Technology, vol. 68, no. 9, pp. 8408–8420, 2019.
[4] 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.
[5] A. Ndikumana, N. H. Tran, K. T. Kim, C. S. Hong et al., “Deep learning based caching for self-driving cars in multi-access edge computing,” IEEE Transactions on Intelligent Transportation Systems, 2020.
[6] T. Taleb, S. Dutta, A. Ksentini, M. İşbil, and H. Flinck, “Mobile edge computing potential in making cities smarter,” IEEE Communications Magazine, vol. 55, no. 3, pp. 38–43, 2017.
[7] H. S. Goian, O. Y. Al-Jarrah, S. Muhaidat, Y. Al-Hammadi, P. Yoo, and M. Dianati, “Popularity-based video caching techniques for cache-enabled networks: A survey,” IEEE Access, vol. 7, pp. 27699–27719, 2019.
[54] S. R. Pandey, M. N. Nguyen, T. N. Dang, N. H. Tran, K. Thar, Z. Han, and C. S. Hong, “Edge-assisted democratized learning towards federated analytics,” IEEE Internet of Things Journal, 2021.

SUBINA KHANAL received the Bachelor in Information Management in Tribhuvan University, Nepal. After her undergraduate, she worked as a quality assurance (QA) engineer for two years. Currently, she is pursuing the M.E. degree with the Department of Computer Science and Engineering, Kyung Hee University, South Korea, for which she was awarded the prestigious Brain Korea 21st Century (BK21) scholarship and the graduate scholarship in 2020. She is the recipient of the best paper award in the International Conference on Ubiquitous and Future Networks (ICUFN) 2021. Her research interests are applied machine learning, federated learning, and the Internet of Things (IoT).

KYI THAR received the Bachelor of Computer Technology degree from the University of Computer Studies, Yangon, Myanmar, in 2007, and the Ph.D. degree in computer science and engineering from Kyung Hee University, South Korea, in 2019, for which he was awarded a scholarship for his graduate study in 2012. He is currently a postdoctoral research fellow with the Department of Computer Science and Engineering, Kyung Hee University, South Korea. His research interests include name-based routing, in-network caching, multimedia communication, scalable video streaming, wireless network virtualization, deep learning, and future Internet.

EUI-NAM HUH (Senior Member, IEEE) received the B.S. degree from Busan National University, South Korea, the master’s degree in computer science from The University of Texas at Arlington, TX, USA, in 1995, and the Ph.D. degree from Ohio University, Athens, OH, USA, in 2002. He is a member of the IEEE and a Review Board of the National Research Foundation of Korea. He has also served many community services for ICCSA, WPDRTS/IPDPS, APAN Sensor Network Group, ICUIMC, ICONI, APIC-IST, ICUFN, and SoICT as various types of chairs. He is the Vice-Chairman of the Cloud/Bigdata Special Technical Group of TTA, and an Editor of ITU-T SG13 Q17. He is currently with the Department of Computer Science and Engineering, Kyung Hee University, South Korea, as a Professor. His interesting research areas are cloud computing, the Internet of Things, future internet, distributed real-time systems, mobile computing, big data, and security.