TS-based Mobility-aware Multi-hierarchical Caching Model with Vehicle Clustering and Content Popularity Prediction

Radouane Baghiani[1], Lyamine Guezouli[2], and Ahmed Korichi[1]

[1]*LINATI Laboratory, Kasdi Merbah University, Ouargla, Algeria,*
[2]*LEREESI Laboratory, HNS-RE2SD, Batna, Algeria*

https://doi.org/10.26636/jtit.2024.3.1616

Abstract — This research is concerned with the fusion of artificial intelligence (AI) and machine learning within multi-hierarchical caching systems, specifically targeting vehicular and edge caching domains. This study introduces an innovative architecture harmonizing Thompson sampling learning-based caching policies with advanced vehicle clustering and content-popularity prediction methods (TS-MMCM). Simulations show substantial performance improvements and a big impact of the proposed approach on system efficiency in dynamic network environments. The proposal demonstrates a notable gain in cache hit rates and decreased latency levels, highlighting the potential of AI to improve caching techniques in dynamic network environments.

Keywords — clustering, Internet of Vehicles, machine learning, mobile edge computing, multi-hierarchical caching, Thompson sampling learning

1. Introduction

Caching is an essential computing approach that improves the performance and efficiency of a system by keeping frequently accessed data in fast access storage. It improves the rate at which data can be processed and greatly reduces the time needed to retrieve data for applications that require high performance and real-time capabilities. The cache handles recurring data requests, thereby minimizing the workload of the backend system, simultaneously enabling efficient scalability of services during high traffic periods.

Caching also enhances user experience by speeding up data retrieval, making it especially advantageous in web construction tasks, as it allows accelerated website loading. From an economic standpoint, data catching reduces the expenses associated with data transport and computations and facilitates offline data access, hence decreasing bandwidth utilization and alleviating network congestion. Furthermore, caching enhances scalability of the system by alleviating data access bottlenecks. Through the use of advanced techniques, caching also guarantees data consistency in dispersed systems, ensuring that the cached data remain accurate and valid.

Vehicular and edge caching are important developments in modern networking that aim to satisfy mobile users' growing need for data and services. Vehicular caching utilizes the storage capabilities of modern vehicles and the characteristics of vehicular networks to enhance both accessibility and availability of data. This approach is particularly beneficial in urban and mobile environments, as it minimizes access latency and improves user experience standards in applications such as video streaming, gaming, and augmented reality [1]. It enables real-time analysis and decision-making for IoT devices and services, allows for effective network expansion by distributing the data workload, and promotes sustainable networking by reducing the amount of energy used for transmitting data.

Vehicular and edge caching work together to achieve a decentralized and efficient networking model. This approach addresses such issues as high bandwidth requirements, minimal latency, and the necessity of ensuring uninterrupted connectivity in mobile and distributed environments.

Artificial intelligence (AI) and machine learning (ML) are transforming caching techniques in computing by improving efficiency and performance levels through predictive caching, adapting to changing data access patterns, as well as optimizing data storage. AI and ML enable caching systems to predict upcoming data queries by analyzing user behaviors and access patterns, thereby resulting in decreased latency. Furthermore, these technologies dynamically adapt caching algorithms in real time to optimize data delivery and resource allocation, which is particularly advantageous in content distribution networks (CDNs) and for handling the influx of data from IoT devices and edge computing. AI and ML enhance caching systems by identifying abnormalities that signal potential security risks or malfunctions, thereby enabling remedial measures to be taken in a timely manner. Tailored caching in web and mobile applications customizes content delivery based on individual user preferences, thus resulting in improved user experience.

In this study, a long short-term memory (LSTM) model is utilized to forecast content request numbers in a time series tailored for Internet of Vehicles (IoV) scenarios. It focuses on caching decisions and proposes a reinforcement learning-based algorithm called the Thompson-based mobility-aware multi-hierarchical caching model with vehicle clustering and content popularity prediction methods (TS-MMCM). This model aims to enhance cache hit ratios by predicting content requests, applying the k-means model to cluster cars depending on their speed and position in order to stabilize communication, forecasting request numbers via the LSTM model, estimating content popularity with the Zipf model, and using reinforcement learning to optimize caching decisions.

The combination of Thompson sampling (TS) with vehicle clustering and content-popularity prediction methods enhances caching rules in dynamic situations, such as CDNs, edge computing, and vehicular networks. This method integrates the adaptive learning of TS, a Bayesian approach renowned for effectively managing the trade-off between exploration and exploitation, with clustering to categorize comparable content or requests, and predicting content popularity. The objective is to improve the effectiveness of caching by utilizing real-time predictions and user behaviors in order to determine which information to cache.

The main benefits of this technique include the ability to make detailed caching decisions based on the popularity of content within certain groups, the flexibility to adapt caching policies in real time according to changes in content demand, and the emphasis on caching content that is expected to be required in the future. This method not only enhances resource utilization, but also amplifies user experience by customizing cached information based on the interests of specific groups or locations, resulting in expedited content delivery and reduced network congestion.

The remainder of this paper is organized as follows. In Section 2, we discuss the related literature. Section 3 outlines the proposed system model, while Section 4 presents the formulation of the problem. The proposed caching strategy is detailed in Section 5. Section 6 discusses the simulation results. Finally, Section 7 concludes the paper and suggests future research.

2. Related Works

Several studies have employed a reinforcement learning approach to enhance caching algorithms. Reference [2] addressed the device-to-device (D2D) caching issue as a multi-agent multi-armed bandit (MAB) learning problem. It employed Q-learning to determine the optimal coordination of caching decisions among multiple agents to maximize caching benefits. Paper [3] utilized a recurrent neural network (RNN) to predict user behavior and content popularity. A Q-learning strategy that employs learning automata was proposed for cooperative caching. This considers the popularity of content and the positions of mobile users to select the best options in a random and unchanging environment.

In [4], the authors investigated user terminal (UT) edge caching in D2D-enabled cellular networks, emphasizing content popularity and UT positioning. The problem of stochastic games was modelled and solved using the proposed multi-agent cooperative alternating Q-learning (CAQL) algorithm, allowing UTs to update caching placement policies for better performance. Article [5] explored the tidal effect in a mobile edge computing (MEC) network with multi-user and multicast data. It formulated the problem as an infinite-horizon average cost Markov decision process, aiming to optimize bandwidth usage and reduce data transmission. The problem was restated in reinforcement learning to learn file popularity and user requests. The design of Q-learning and a deep Q-network (DQN) was proposed, providing insights into the network caching design.

2.1. Caching in Vehicles

Caching-related issues arising from vehicle mobility are different from those found in cellular networks. The authors of [6] investigated the idea of automobiles acting as moving cache nodes and creating an automobile cloud to distribute the requested content to consumers using 6G. To prepare edge smart technologies for an upcoming 6G vehicle network, work [7] suggested utilizing parked vehicles as additional edge nodes. Such a solution will provide abundant resources in conjunction with existing ground infrastructure.

The architecture proposed in paper [8] consists of three layers: an airship, UAVs, and vehicles, all of which are used for vehicular caching. The authors employed the DQN method to obtain the most advantageous caching approach. In study [9], a sophisticated machine learning method was applied to information-centric networking-based vehicular networks. This technique expects future user requests by predicting their future evaluations of videos.

In [10], deep reinforcement learning (DRL) and federated learning (FL)-based content caching methods were proposed for efficient transmission tasks meeting the applicable latency requirements. These methods optimize latency by considering regional preferences and latency constraints. Multi-agent reinforcement learning (MARL) and FL were used to forecast content popularity and determine cached content in each area.

Another study in [11] proposed a deep learning system for content caching in autonomous vehicles that makes use of MEC servers to cache high-probability content and relies on a multi-layer perceptron (MLP) to predict content demands in certain regions. A convolutional neural network (CNN) was employed for the prediction of passenger profiles, while the self-driving car utilized binary classifications and the k-means approach to choose pertinent infotainment content for downloading and caching. In [12], researchers focused on issues pertaining to automated vehicle control and proactive caching in roadside units. This study employed deep reinforcement learning to enhance efficiency and quality of experience (QoE) by implementing proactive caching actions.

The authors of [13] proposed caching for infotainment content in self-driving cars based on MEC servers and utilizing CNN-

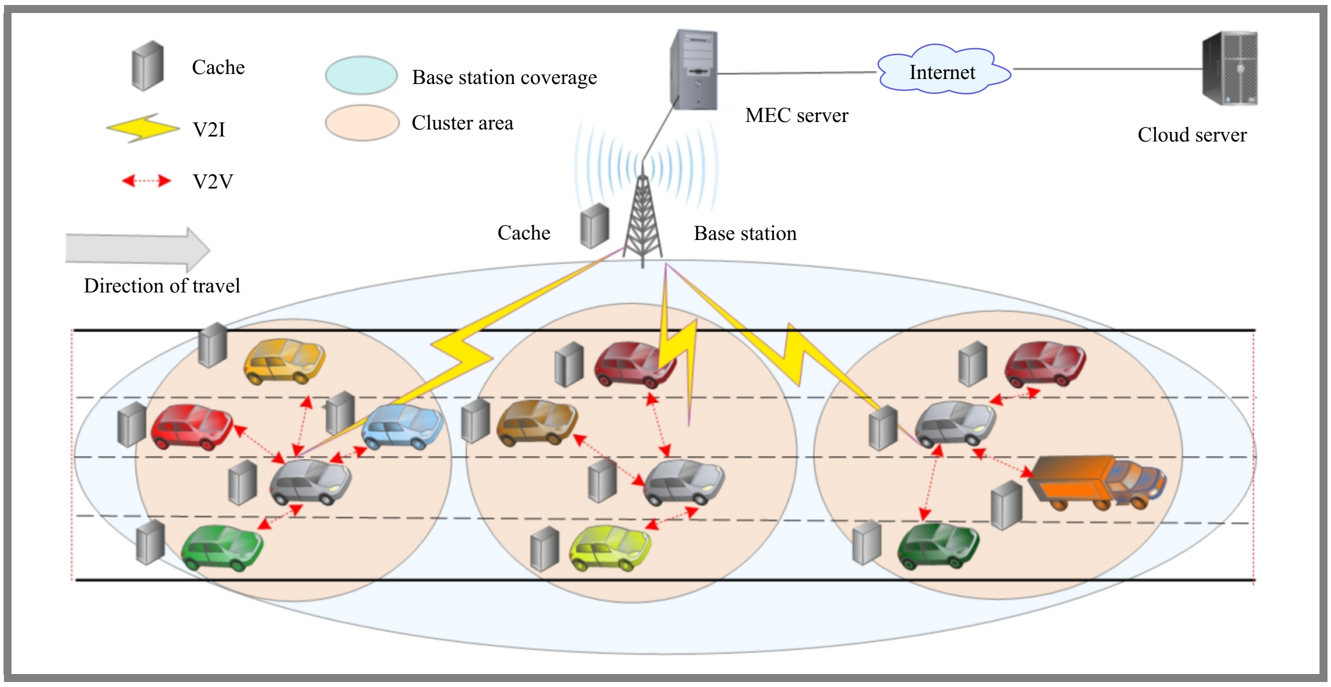

Fig. 1. Proposed architecture.

learned passenger features for high expected probabilistic values in their areas.

2.2. Caching in Edge

Recently, numerous research projects have been focusing on edge caching. In [14], a learning-based cooperative content caching policy was considered for the MEC architecture in situations where user preferences are unknown and only the demands for historical content are visible. This study proposes the MARL-based approach to address the cooperative content caching problem by modeling it as a multi-agent multi-armed bandit problem.

In [15], the authors proposed a novel edge-assisted intelligent caching framework that improved the cache hit rate. This framework can independently develop a caching technique in real time by analyzing the request sequence without requiring preliminary data processing or engineering features. Another study described in [16] investigated the prediction of spatial content preferences using distributed learning approaches and mobility predictions.

In contrast to the majority of other studies, paper [17] applied deep reinforcement learning to collaborative caching and set varying content sizes, while in [18], a collaborative caching approach for mobile edge computing servers incorporated multi-agent reinforcement learning.

2.3. Combining Vehicle and Edge Caching

Recent studies have integrated local vehicle and edge device caches, leveraging user and regional preferences to understand and optimize vehicle usage. The architecture proposed in [19] suggested a collaborative caching approach for the IoV by utilizing content request prediction (CCCRP) to minimize latency in obtaining content. The authors employed k-means clustering to group vehicles, LSTM networks to predict content requests, and reinforcement learning to improve caching decisions, hence enhancing the quality of service for vehicle requests.

The study described in [20] presented a cooperative caching approach for downloading content in reinforcement learning, applying the k-means algorithm for vehicle clustering and long short-term memory for content prediction, as well as the DRL algorithm and DQN to determine the most effective caching technique.

3. Proposed Architecture

In this section, we present an architecture for multilevel data caching, as depicted in Fig. 1, which includes cluster member users, cluster heads, base stations (BSs), and edge servers to cache various types of content. Each edge server described in this proposal can handle the BSs located within its coverage area. Several users were present in each base station's dominant region. This is because the emphasis is placed on the cache capacity of the vertical architecture.

Vehicles covered by the same BS insurance were grouped into clusters according to their mobility traits. Cluster heads are assigned to each cluster, and the cluster head selection remains constant in each time frame. Vehicles within the range of a single BS can form multiple non-overlapping clusters, with each cluster uniquely assigned to one BS. The cluster head for each vehicle corresponds to the cluster it is part of and is marked as $H_{i,1}$, where i represents the index of clusters within the coverage area of the BS.

Further, vehicles within the cluster may also be denoted by $H_{i,J}$, where j ($j \geqslant 2$) represents the index of the vehicle within cluster i. For example, the cluster head $H_{i,1}$ serves cluster members. All notations are summarized in Tab. 1. In this cache structure, when the vehicle user initiates a request, the local cache is first checked to locate the requested content. When the content is available in the local cache, it becomes directly accessible.

Otherwise, the user seeks content from $H_{i,1}$. If the content is cached in $H_{i,1}$, the user retrieves it directly. If it is not found in $H_{i,1}$, then the user sends a request to the BS covering their location. The user can instantly access the content from the covering BS if it is cached. If cache retrieval is unsuccessful, the request is forwarded to the edge server and, if not found, the request is forwarded to the cloud server.

The reliance on mobility traits to group vehicles into clusters enhances the reliability of communication lines between vehicles connected to the same base station. A cluster head is assigned to each cluster. It is important to remember that the cluster head is always chosen in the same manner for consecutive time slots. Normalized characteristics, such as the position and speed of each vehicle, have been derived. This process aims to group N_v–generated vehicles into M_c vehicle clusters.

Because every vehicle in the cluster, including the cluster head, has the ability to cache content, the cluster heads can establish one-hop communication with other vehicles in the same cluster. However, no communication can be established between any two vehicles that are part of different clusters.

3.1. Caching Model

Set $L = \{1, 2, \ldots, l\}$ represents the content repository, where l is the total amount of content. The cloud server stores the complete content repository L, whereas the edge server collaborates with the base stations, cluster heads, and vehicle users to cache the content. Each content has a distinct size, represented by $Z = \{z_1, z_2, \ldots, z_l\}$, where l is an element of L.

We grouped these content types into n categories, denoted by $cat = \{\text{category } 1, \text{category } 2, \ldots, \text{category } m\}$. The edge server has a local cache with capacity C_{edge}. The base station offers a cache capacity denoted as C_{BS} and the cluster head has a local cache with a capacity of $C_{Hi,1}$, where M_c is the number of clusters. $V = \{v_1, v_2, \ldots, V_u\}$ represents a group of users, assuming that certain content is stored in the user's local cache with a cache capacity of $C_{Hi,j}$. When $S(l, edge) = 1$, content l is stored at the edge, otherwise $S(l, edge) = 0$. For $S(l, BS) = 1$, content l is stored in the BS. $S(l, BS) = 0$ signifies that the content is not stored in the BS.

For $S(l, H_{i,1}) = 1$, the content l is stored in the cluster head, otherwise $S(l, H_{i,1}) = 0$ denotes the content l is not stored in $H_{i,1}$. Similarly, $S(l, H_{i,j}) = 1$ implies that content l is cached in the local cache of users in cluster i, whereas $S(l, H_{i,j}) = 0$ indicates otherwise. It is crucial to ensure that the cache size of any device does not exceed its total cache capacity at any given time t:

Tab. 1. Summary of notations used.

Symbol	Description
BS	Base station
$H_{i,1}$	Cluster head
$H_{i,j}$	Cluster member
Nv	Total number of vehicles in the system
Mc	Number of clusters grouping the vehicles
L	Content repository
Z	Size of each file repository
cat	Category of content
V	Set of vehicles
C_{edge}	Cache capacity of edge cache
C_{BS}	Cache capacity of base station cache
$C_{H_{i,1}}$	Cache capacity of cluster head cache
$C_{H_{i,j}}$	Cache capacity of local cache
$S(l, edge)$	Cache state of the file l in the edge
$S(l, BS)$	Cache state of the file l in the BS
$S(l, H_{i,1})$	Cache state of the file l in the cluster head
$S(l, H_{i,j})$	Cache state of the file l in the cluster member
$HitR_t^{edge}$	Cache hit rate of the edge
$HitR_t^{BS}$	Cache hit rate of the BS
$HitR_t^{H_{i,1}}$	Cache hit rate of the cluster head
$HitR_t^{H_{i,j}}$	Cache hit rate of the cluster member
Θ_{edge}	Caching edge policy
$\Theta_{H_{i,1}}$	Caching cluster head policy
$\Theta_{H_{i,j}}$	Caching cluster member policy
P_l	Probability of each file
$L_{i,l}(t)$	Location of vehicle
$S_{i,l}(t)$	Speed of vehicle
CL	Connectivity lifetime

$$\begin{cases} \sum_{e=1}^{l} z_e \cdot S(e, edge) \leqslant C_{edge}, & \forall\, e \in L \\ \sum_{r=1}^{l} z_r \cdot S(r, BS) \leqslant C_{BS}, & \forall\, r \in L \\ \sum_{b=1}^{l} z_b \cdot S(b, H_{i,1}) \leqslant C_{H_{i,1}}, & \forall\, b \in L \\ \sum_{d=1}^{l} z_d \cdot S(d, H_{i,j}) \leqslant C_{H_{i,j}}, & \forall\, d \in L \end{cases},$$

where z_e, z_r, z_b, and z_d represent the size of the cached content at each piece of equipment or location indexed by e, r, b, and d, respectively.

$S(e, edge), S(r, BS), S(b, H_{i,1})$, and $S(d, H_{i,j})$ denote the cache state of the content for the edge, BS, cluster head, and cluster members, respectively, with the respective indices. C_{edge}, C_{BS}, $C_{H_{i,1}}$, and $C_{H_{i,j}}$ represent the maximum cache capacities for the edge, BS, cluster head, and cluster members, respectively, owing to the constraint of the restricted caching capacity of the edge, BS, $H_{i,1}$, and $H_{i,j}$.

We propose a cache-placement technique that relies on the popularity of content. The chance of a given piece of content being requested is impacted by its popularity ranking, with more popular content having a higher level of probability of being requested. Zipf distribution is commonly used to represent the probability of a content request. The likelihood of content being requested can be ascertained by employing the Zipf model which considers the number of requests for that content. We define R_l to represent content l that is ranked at the R-th level of popularity. The Zipf model is described as:

$$P(R, s, N) = \frac{1/R^s}{\sum_{q=1}^{N} \frac{1}{q^s}}, \quad q \in \{1, 2, \ldots, l\} , \tag{1}$$

where N represents the total amount of content and s is the exponent parameter that characterizes the Zipf distribution. A greater value of s indicates that the content queries are more focused on the highest ranking. Evaluation of caching strategies involves examining the cache hit rate to understand its benefits and drawbacks. The cache hit rate of the edge server, cluster head, and cluster members is:

$$HitR_t^{edge} = \sum_{e=1}^{l} P_e \cdot S(e, edge), \quad \forall\, e \in L , \tag{2}$$

$$HitR_t^{BS} = \sum_{r=1}^{l} P_r \cdot S(r, BS), \quad \forall\, r \in L , \tag{3}$$

$$HitR_t^{H_{i,1}} = \sum_{b=1}^{l} P_b \cdot S(b, H_{i,1}), \quad \forall\, b \in L , \tag{4}$$

$$HitR_t^{H_{i,j}} = \sum_{d=1}^{l} P_d \cdot S(d, H_{i,j}), \quad \forall\, d \in L . \tag{5}$$

The probability vector $P_l = [p_1, p_2, \ldots, p_l]$ indicates the likelihood of each content being required by vehicle users in the upcoming time period.

3.2. Content Request Model

All the components of the proposed system, including vehicle users, cluster heads, base stations, and edge servers, are capable of caching content. If a vehicle within the cluster requires specific content, there are five potential locations where it can be obtained:

- Local cache. The vehicle user's local cache is first checked to locate the requested content. If the required content is present in the local cache, it is directly accessible.
- Cluster head. If the content is not available in the local cache, the user seeks it from the cluster head. The requested content is directly retrieved from the cluster head if it is cached using vehicle-to-vehicle (V2V) connectivity.
- Base station. If the content is not located in the cluster head, the user sends the request to the BS that covers it. The user will instantly access the content from the BS if it is cached using vehicle-to-infrastructure (V2I) connectivity.
- Edge server. For content not found in the BS, the user transmits the query to the edge server through the BS. If cache retrieval remains unsuccessful, the request is forwarded to the cloud server. However, this results in greater latency.

4. Problem Formulation

The cache hit rate is an important performance metric used in caching techniques, as it indicates the extent to which the content in the cache aligns with user requests. Our objective was to create an online caching technique that maximizes the cache hit ratio. Owing to the large volume of files, we categorized the individual pieces of content into various types and assigned varying probabilities to each of the selection types. Once the file type is identified based on probability, one file of that type is selected for the cache. The objective of the caching method for each vehicle user, cluster, and edge is to maximize the cache hit rate as follows:

$$\max_{\Theta_{H_{i,j}}} HitR_t^{H_{i,j}}(\Theta_{H_{i,j}}) , \tag{6}$$

$$\max_{\Theta_{H_{i,1}}} HitR_t^{H_{i,1}}(\Theta_{H_{i,1}}) , \tag{7}$$

$$\max_{\Theta_{edge}} HitR_t^{edge}(\Theta_{edge}) . \tag{8}$$

Our technique aims to improve the cache hit rates at both the edge server and the car user layer. Considering the proposed architecture model, limitations regarding the vehicle users, cluster heads, and edge servers' ability to cache data, as well as the mobility constraint imposed by the vehicle, the proposed issue formulation is as follows:

$$\begin{aligned} &\max\ \{\theta_{H_{i,j}}, \theta_{H_{i,1}}, \theta_{edge}\} \quad \text{for } HitR_t^{H_{i,j}} , \\ &\max\ \{\theta_{H_{i,j}}, \theta_{H_{i,1}}, \theta_{edge}\} \quad \text{for } HitR_t^{H_{i,1}} , \\ &\max\ \{\theta_{H_{i,j}}, \theta_{H_{i,1}}, \theta_{edge}\} \quad \text{for } HitR_t^{edge} , \\ &\max\ \{\theta_{H_{i,j}}, \theta_{H_{i,1}}, \theta_{edge}\} \quad \text{for } HitR_t^{total} . \end{aligned} \tag{9}$$

Given the constraints set C_1, we have:

$$\begin{cases} S(l, edge) \\ S(l, BS) \\ S(l, H_{i,1}) \\ S(l, H_{i,j}) \end{cases} \in \{0, 1\}, \quad \forall\, l \in L .$$

Given the constraints set C_2, we have:

$$\begin{cases} \sum_{e=1}^{l} z_e \cdot S(e, edge) \leqslant C_{edge}, & \forall\, e \in L \\ \sum_{r=1}^{l} z_r \cdot S(r, BS) \leqslant C_{BS}, & \forall\, r \in L \\ \sum_{b=1}^{l} z_b \cdot S(b, H_{i,1}) \leqslant C_{H_{i,1}}, & \forall\, b \in L \\ \sum_{d=1}^{l} z_d \cdot S(d, H_{i,j}) \leqslant C_{H_{i,j}}, & \forall\, d \in L \end{cases},$$

where the initial constraint indicates that each content, represented by 1 or 0, can either be cached or not cached. Hence, values from the set of $[0, 1]$ must be assigned to $S(l, edge), S(l, BS), S(l, H_{i,1})$, and $S(l, H_{i,j})$. Furthermore, the last constraint guarantees that the cumulative amount of the content cached in the edge, BS, $H_{i,1}$, and $H_{i,j}$ does not exceed their respective caching capacities. Reinforcement learning (RL) can be a prospective approach for solving the caching optimization problem described above. We propose to use an effective RL approach to address this multi-agent decision-making issue.

5. Proposed Solution

The proposed approach consists of three components: clustering vehicles using k-means [21], predicting content requests using LSTM, and implementing the TS algorithm for content caching decisions.

5.1. Vehicle Clustering

Our method involves clustering vehicles according to their mobility to reduce the signaling load from V2V broadcasting and improve vehicular communication connections. In each cluster, the cluster head can directly communicate with other members located within the same cluster using single-hop V2V communication. This setup ensures efficient and direct communication pathways within the cluster, optimizing the network's overall functionality and reducing the need for complex, multi-hop communication strategies. Vehicles are classified according to their mobility characteristics, into distinct clusters within a single BS coverage area.

We created M_c clusters from a set of N_v generated vehicles by using normalized attributes, such as position and speed,

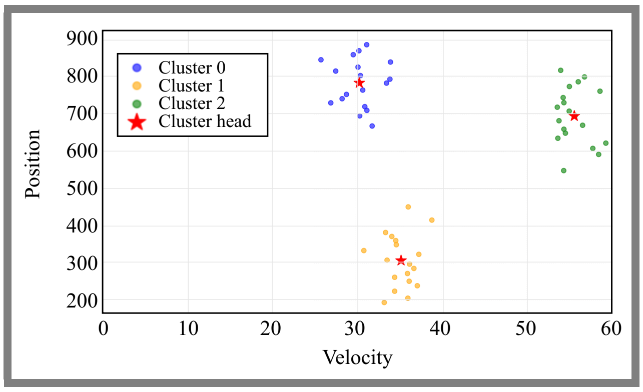

Fig. 2. Vehicle clustering with cluster heads.

obtained from each vehicle (Fig. 2). In vehicle clustering, the primary challenges include determining the clustering technique and choosing the cluster head. This study uses the k-means algorithm for clustering. As shown in Algorithm 1, four main steps are part of the process: choosing the M_c cluster centers, grouping the rest of the vehicles, updating the cluster centers, and repeating the process while the cluster centers remain constant.

To speed up clustering results, the algorithm first gives priority to choosing M_c vehicles that are widely separated. It is important to remember that the k-means algorithm depends on calculations of Euclidean distance. It is possible that the k-means cluster centers do not perfectly line up with the individual vehicles. Choosing cluster heads that match these clusters is a further refining step.

The cluster head provides the content cache source and manages access, requiring a reliable connection with other cars. Therefore, we utilized a distinct parameter called connectivity lifetime (CL) to choose cluster heads based on the durability of vehicle connections [22].

Let us represent the l-th car in cluster i as $N_{i,l}$. The vehicle's location and speed at time t are represented as $L_{i,l}(t)$ and $S_{i,l}(t)$, respectively. Vehicles occasionally send a "hello" beacon packet to other vehicles within the coverage area of the same BS. The hello beacon packet includes the vehicle's mobility information, such as its location $L_{i,l}(t)$ and speed $S_{i,l}(t)$. Therefore, every car can calculate its associations with other vehicles using this information. CL is the time it takes for a communication link to occur between two vehicles. Therefore, when the CL expires, the link between the two vehicles is expected to end, signifying that their relative distance is projected to reach the maximum broadcasting range, represented as δ, during this time frame.

The requirement for preserving a link between cars $N_{i,l}$ and $N'_{i,l}$ (where $l' \neq l$) follows this principle:

$$\left(L_{i,l}(t + \mathrm{CL}_{i,l}^{i,l'}) - L_{i,l'}(t + \mathrm{CL}_{i,l}^{i,l'})\right)^2 = \delta^2 . \tag{10}$$

If the vehicle maintains the same speed for the following time interval τ, the equation for the location of the car is:

$$L(t + \tau) = L(t) + \tau \cdot S(t) . \tag{11}$$

Therefore, Eq. (10) can be written as:

$$\left(L_{i,l}(t) - L_{i,l'}(t) + \mathrm{CL}_{i,l}^{i,l'}\,[S_{i,l}(t) - S_{i,l'}(t)]\right)^2 = \delta^2 , \tag{12}$$

where $CL_{i,l}^{i,l'}$ can be calculated by solving Eq. (12) in which the speed and location are 2D coordinates as follows:

$$\mathrm{CL}_{i,l}^{i,l'} = \frac{\delta^2 - |\mathbf{L}_{i,l}(t) - \mathbf{L}_{i,l'}(t)|^2}{|\mathbf{S}_{i,l}(t) - \mathbf{S}_{i,l'}(t)|} . \tag{13}$$

Once the clusters are determined using the k-means algorithm, the average CL for each vehicle $N_{i,l}$ in the corresponding cluster i can be computed as:

$$\overline{\mathrm{CL}}_{i,l} = \frac{1}{n_i - 1} \sum_{l'=1}^{n_i} \mathrm{CL}_{i,l}^{i,l'} , \quad l' \neq l \,. \tag{14}$$

Consequently, the vehicle with the highest average CL within this cluster may be designated as the cluster head. This choice ensures that the cluster head and every other vehicle within the same cluster enjoy optimal link stability.

Algorithm 1 Vehicle clustering based on the k-means approach.

```
1:  Input: Vehicle set $V = \{V_1, V_2, \dots, V_{nv}\}$, number of vehicle clusters $M_c$
2:  Output: Vehicle cluster set $\{c_1, c_2, \dots, c_{mc}\}$
3:  Select initial cluster centers $\{t_1, t_2, \dots, t_{mc}\}$
4:  repeat
5:      for $l = 1$ to $N_v$ do
6:          Determine the distance between $N_l$ and each center $t_i$, denoted as $d_{i,l} = \|N_l - t_i\|^2$
7:          Assign $N_l$ to the cluster whose cluster center is the closest
8:      end for
9:      for $i = 1$ to $M_c$ do
10:         Get the fresh cluster centre $t_i^* = \frac{1}{|C_i|} \sum_{N_l \in C_i} N_l$
11:         if $t_i^* \neq t_i$ then
12:             Refresh the existing cluster center $t_i = t_i^*$
13:         else
14:             Maintain the current cluster center constant
15:         end if
16:     end for
17: until cluster centers become fixed
```

5.2. Content Popularity Prediction

The proposed design architecture emphasizes the importance of predicting content popularity, wherein the caching technique relies on content popularity. Many previous studies assume that the popularity of the content is pre-established or use the frequency of requests as a criterion for assessing content popularity. In practice, content popularity requires an element of freshness. In fact, vehicle content requests already showed a steady trend over time [23]. By analyzing past request-related data, it is possible to uncover the fundamental trends governing the frequency of content requests, which can then be used to forecast the number of requests in upcoming time intervals.

Here, this forecast request volume is defined as popularity of content. The use of a machine learning technique like RNN has enabled us to forecast the popularity of newly generated content. RNNs is capable of uncovering underlying temporal patterns by analyzing the historical request times for each content. However, conventional RNNs may encounter issues such as vanishing or exploding gradients with prolonged propagation. As a remedy, LSTM is favored, representing an optimized version of RNN [24].

The structure of an LSTM unit is shown in Fig. 3. Let μ_t and h_t represent the input and output data at time slot t, respectively. At each time slot t, the cell receives a new input μ. Three gates, named input gate i, forget gate f, and output gate o, are positioned within the cell. The value of these gates can be computed as follows:

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, \mu_t] + b_f\big) \,, \tag{15}$$

$$i_t = \sigma\big(W_i \cdot [h_{t-1}, \mu_t] + b_i\big) \,, \tag{16}$$

$$o_t = \sigma\big(W_o \cdot [h_{t-1}, \mu_t] + b_o\big) \,, \tag{17}$$

where W_f, W_i, and W_o indicate the weight matrices, whereas b_f , b_i, and b_o represent the variable biases for the three gates. $\sigma(\cdot)$ is the non-linear activation sigmoid function.

The procedure for updating information in LSTM is as follows [25]. The forget gate f_t determines the portion of C_{t-1} to discard, such as:

$$f_t \cdot C_{t-1} \,. \tag{18}$$

The current state information is updated:

$$i_t \cdot \tilde{C}_t \,, \tag{19}$$

where

$$\tilde{C}_t = \mathrm{tgh}\,\big(W_c \cdot [h_{t-1}, \mu_t] + b_c\big) \,. \tag{20}$$

Here, b_c and w_c denote the variable bias and weight matrix of the memory cell, and $\mathrm{tgh}(\cdot)$ is the hyperbolic tangent function. The current unit is updated in the following manner:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \,. \tag{21}$$

Consequently, the output data is calculated as:

$$h_t = o_t \cdot \mathrm{tgh}(C_t) \,. \tag{22}$$

The LSTM approach is capable of predicting the volume of content requests in the future, allowing to optimize the coaching of content for car users. By using time series context, the LSTM model is provided with the count of content requests from the preceding time step $t-1$, as its input, denoted by:

$$\mu(t-1) = \big\{\mu_1(t-1), \mu_2(t-1), \dots, \mu_Q(t-1)\big\} \,. \tag{23}$$

The LSTM output predicts the count of content requests for the forthcoming time period, as a set of values represented by:

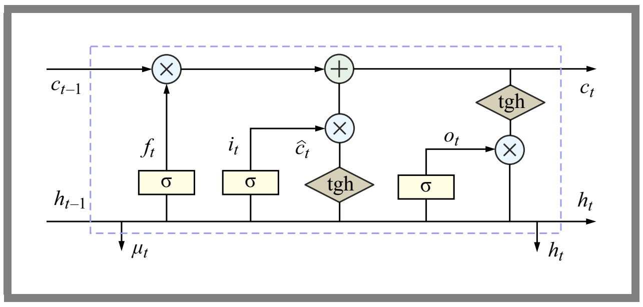

Fig. 3. LSTM cell.

$$\theta(t) = \left\{\theta_1(t), \theta_2(t), \ldots, \theta_Q(t)\right\} . \tag{24}$$

Items within function $\theta(t)$ are sorted to create a sorted index vector such as:

$$\lambda = \left\{\lambda_1, \lambda_2, \ldots, \lambda_Q\right\} . \tag{25}$$

This sorted index vector is then input into the Zipf model to determine the popularity of content in the repository, as described in the caching model, producing a popularity distribution:

$$\pi = \left\{\pi_1, \pi_2, \ldots, \pi_Q\right\} . \tag{26}$$

Our LSTM-based prediction model for predicting content request volumes is structured as follows:

- Input layer provides historical data of content request numbers, processed and normalized.
- First LSTM layer:
  - consists of 128 units (number of memory cells in the layer) with `return_sequences=true` to maintain temporal sequence processing for deeper layers.
  - uses the rectified linear unit (ReLU) activation function for intermediate layers to introduce non-linearity.
  - includes a dropout rate of 0.2 to mitigate overfitting by randomly omitting a subset of features during training.
- Second LSTM layer:
  - comprises 64 units configured with `return_sequences=true` and takes the sequence output from the previous LSTM layer and processes it further, outputting a sequence of 64-dimensional hidden states.
  - ReLU activation function.
  - this layer also includes a dropout of 0.2.
- Third LSTM layer:
  - 32 units.
  - ReLU activation function.
  - false return sequences (to return the final hidden state for the next layer).
  - 0.2 dropout.

  This layer takes the sequence output from the previous LSTM layer and processes it to output a 32-dimensional hidden state representing the entire sequence.
- Dense layer (output layer) features a single neuron with a linear activation function to predict the count of future content requests, reflecting the anticipated popularity.

The model compilation includes:

- Optimizer – the Adam optimizer is utilized for its efficient computation and adaptive learning rate of 0.001, enhancing the convergence of training.
- Loss function – mean squared error (MSE) is employed to quantify the accuracy of predictions, providing a clear measure of prediction error in the context of regression.

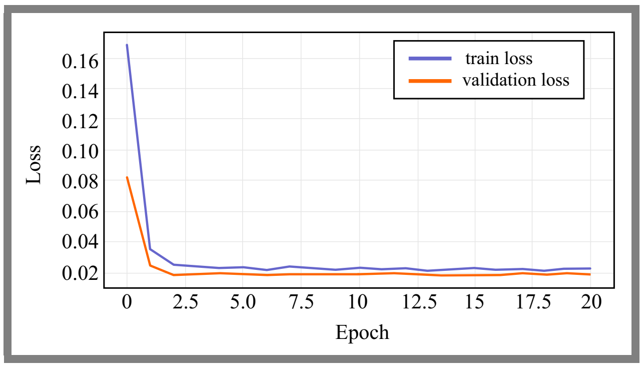

Fig. 4. Model loss over an epoch.

- Metrics – mean absolute error (MAE) provides a measure of the mean absolute difference between observed values and predictions, offering a more direct interpretation than MSE.

The training configuration is as follows:

- Epochs and batch size – the model is trained over 100 epochs with a batch size of 32, balancing the model's exposure to the training data with computational efficiency.
- Early stopping is implemented with a patience of 10 epochs; training is stopped early if the validation loss does not improve for 10 consecutive epochs, preventing overfitting.

For model training, 20% of the data is used for validating performance of the model during training, which facilitates tuning and prevents overfitting. Early stopping is used to halt the training process when the model's performance on the validation set stops improving.

5.3. Evaluation of Proposed Model

The dataset used for predicting content requests originates from the Big Data Challenge [26]. It includes data for ten types of content, collected between November 1, 2013 and January 1, 2014, with each data point representing a 10-minute interval. This analysis is based on a dataset of 1,000 such data points.

The first 800 data points are utilized as the training set, and the model's performance is subsequently tested on the last 200 data points. Once trained, the model's performance is evaluated on the validation set. Figure 4 illustrates the training and validation losses over epochs and reveals the following insights:

- Sharp decline in initial epochs. There is a significant drop in both training and validation losses within the first few epochs, indicating that the model quickly captures dominant patterns in the data.
- Convergence. After the initial sharp decline, both losses level off, showing only slight further reductions. This trend suggests that continuing training beyond these epochs might not yield substantial improvements, pointing towards potential diminishing returns.
- Closeness of losses. The training and validation losses remain close throughout the process, suggesting that the model generalizes well and is not overfitting. This closeness

is indicative of a model that performs well on both seen and unseen data.

This comprehensive evaluation highlights the LSTM-based content prediction model's ability to learn and predict content request volumes effectively, showcasing its utility in practical scenarios based on the dataset from the Big Data Challenge. The analysis of the loss plot further confirms the model's capability to generalize, making it a reliable tool for forecasting content requests in similar settings.

By integrating LSTM predictions into our caching strategy, we can dynamically adjust the cached content based on real-time popularity forecasts. This approach not only enhances the efficiency of the caching system but also ensures that the most relevant and demanded content is available to users promptly, thereby reducing latency and improving user experience.

This sophisticated LSTM framework forms the backbone of the proposed predictive model, enabling robust and accurate forecasting of content popularity, which is crucial for the effective management of cache resources in vehicular networks.

5.4. Cooperative Caching-based TS Algorithm

Reinforcement learning is an algorithmic approach centered on mapping behaviors based on the environmental state [27]. In this framework, the agent within the reinforcement learning system chooses and performs actions from a set based on the system's state, subsequently receiving a reward corresponding to the action taken. This reward serves as a metric for assessing the effectiveness of the chosen action. After the training phase, the agent can swiftly identify and perform actions that are linked to higher rewards, depending on the system's state.

In the context of MAB problems, each arm corresponds to a distinct action, yielding a reward upon selection. However, the likelihood of receiving a reward varies across arms, presenting the challenge of efficiently choosing arms within a limited number of trials to optimize rewards. Balancing exploration and exploitation is a widely recognized challenge in this scenario. To maximize rewards, decision-makers must strike a balance between exploring to leverage arms more effectively and exploiting the knowledge gained from prior exploration. The MAB problem, utilizing strategies like TS, aims to maximize the expected reward, ensuring the best possible outcome [28].

TS is more effective for caching issues with high uncertainty, as its exploration of varied strategies can enhance the overall performance. TS is a stochastic algorithm used for decision making in situations of uncertainty, commonly applied to MAB problems. It assigns scores to each arm by assuming the reward probability for each arm conforms to a beta distribution [29].

The latter is a continuous distribution of probabilities inside the interval $[0, 1]$, distinguished by two positive shape parameters denoted by a and b. The average value of the beta distribution is determined by $\frac{a}{a+b}$. A larger a results in a higher average value of $\mathrm{beta}(a, b)$, whereas a higher b decreases this average. The TS algorithm estimates the return, represented by ϕ, by sampling $\mathrm{beta}(a, b)$ [29].

Following the outcome of arm selections, updates are made to the selected arm. Receiving a reward of 1 increases the corresponding a value by 1. When the reward is 0, the b value is incremented by 1. The sampling's inherent randomness enables TS to naturally achieve a balance between exploration and exploitation (EE).

Additionally, adjustments to the EE balance can be made by altering the update mechanism for the TS parameters a and b. We designated the edge server, base station, cluster head, and each user with a local cache as agents. Each category is considered an arm. Each arm has a unique likelihood of being chosen. Once the file category is identified based on probability, the file is selected from that category to cache.

In this study, the TS technique is used to iteratively select actions (arms) based on their predicted reward probability. The algorithm tracks reward distribution estimations for each arm using beta distributions. The algorithm samples beta distributions at each iteration to evaluate the likelihood of each arm being the best selection. Next, it chooses the arm with the highest calculated likelihood and adjusts its settings according to the incentives it has received. In this context, rewards refer to cache hit rates.

The reward is an indicator of success linked to choosing a specific arm. The reward is calculated according to the hit rate attained by completing the required tasks. The hit rate indicates the ratio of successful task completions to the total number of requested tasks. To update the parameters of the beta distribution at time-slot $t + 1$, the following formula was used:

$$\begin{aligned} a^{t+1}_{d,cat_i} &= a^{t}_{d,cat_i} + R^{t}_{d,cat_i} \\ b^{t+1}_{d,cat_i} &= b^{t}_{d,cat_i} + (1 - R^{t}_{d,cat_i}) \end{aligned}, \tag{27}$$

where R^{t}_{d,cat_i} is the hit rate (reward) of the selected arm of the file in the cache of each device d at time t.

Algorithm 2 illustrates the proposed TS-MMCM approach, while the TS algorithm process is outlined in Algorithm 3. The TS-based algorithm sorts content into numerous categories. Without categorizing content, each agent has an action space of around $(2^{|L|})$ when selecting from $|L|$ files. Therefore, the algorithm becomes ineffective. The TS-based approach is suggested to decrease the action space of Q-learning [30]. Initially, the algorithm initializes the beta distribution of each arm as follows:

$$\phi^{0}_{cat_i} \sim \mathrm{beta}(a^{0}_{cat_i}, b^{0}_{cat_i}) \leftarrow (1, 1), \text{ where } 1 \leqslant i \leqslant N \ , \tag{28}$$

where N is the number of arms.

Therefore, the arm with the highest sampled value is selected as the optimal arm. According to the stored probabilities of each arm, content will be cached based on its probability of being requested in the future. Once the items are stored in the caches of each device, the rewards are acquired. The posterior distribution of each selected arm is updated as follows:

$$\phi^{t+1}_{cat_i} \sim \mathrm{beta}(a^{t+1}_{cat_i}, b^{t+1}_{cat_i}), \quad \text{where } 1 \leqslant i \leqslant N \ . \tag{29}$$

Algorithm 2 TS-based mobility-aware multi-hierarchical caching model with vehicle clustering and content popularity prediction methods (TS-MMCM).

```
1:  Phase 1. Initialization
2:  Initialize parameters: number of vehicles, vehicle clusters, period $T$, content categories $|cat|$, repository, content sizes, parameter $s$, cache capacity
3:  Initialize beta distribution parameters:
4:  $\phi^0_{cat_i} \sim \text{beta}(a^0_{cat_i}, b^0_{cat_i}) \leftarrow (1,1), 1 \leqslant i \leqslant N$
5:  Phase 2. Vehicle clustering
6:  Perform Algorithm 1 to obtain vehicle clusters
7:  Phase 3. Obtain cluster heads
8:  Choose cluster heads based on CL selection criterion:
    $\overline{\text{CL}}_{i,l} = \frac{1}{n_i-1} \sum_{l'=1}^{n_i} \text{CL}^{i,l'}_{i,l}, \quad l' \neq l$
9:  Phase 4. Find the best caching decision
10: repeat for each round
11:     Obtain the predicted number of content requests $\theta(t)$ using LSTM
12:     Sort $\theta(t)$ to obtain sorted index vector $\lambda_Q$
13:     Obtain content popularity vector $\pi_Q$ using Zipf model: $P(R,s,N) = \frac{1/R^s}{\sum_{q=1}^{N} \frac{1}{q^s}}, q \in \{1,2,\ldots,l\}$
14:     for each cluster $H_i$ do
15:         for each vehicle user $H_{i,j} \in H_i$ do
16:             Determine caching method in each local cache using a TS-based algorithm
17:         end for
18:         Determine the caching policy in each cluster head $H_{i,1}$ using a TS-based algorithm
19:     end for
20:     Determine caching policy in the edge server using a TS-based algorithm
21:     Update content in each cache
22: until convergence criteria are met or fixed number of rounds $T$
```

6. Simulation Results

In this section, we validate the effectiveness and efficiency of proposed approach by conducting simulations using Python. The performance metrics we considered are the hit rate and latency. The hit rate refers to the proportion of data or resource requests that are successfully retrieved from the cache, while the latency is the time duration from when a request is sent by the vehicle user to when the last data packet is received. Table 2 lists the simulation's parameters and their values.

In the simulation scenario, a library containing 1000 items was used, each having a size ranging from 5 to 100 MB. The total size of all pieces of content was 51 GB. The cache capacity of the edge server was configured to be between 10% and 25% of the entire volume of content. The cache capacity allocated to each user and cluster head was adjusted to be between 10% and 25% of the cache size of the edge server, ensuring a fair and efficient distribution of caching resources across the network [30]. We modelled the arrival of requests from cars as a Poisson process in each time slot. To simulate

Algorithm 3 Thompson sampling.

```
1:  Input: Initialize parameters
2:  Output: Report the selected arms and their corresponding probabilities
3:  Number of categories $N$
4:  Initialize the posterior distributions of each arm as a function given by Eq. (28)
5:  Initialize variable for reward
6:  repeat
7:      for each round do
8:      Pull arms:
9:        Sample from the posterior distributions of each arm
10:       Select the arm with the highest sampled value
11:     Obtain reward:
12:       Execute the chosen action/arm in the environment
13:       Update the cache
14:       Observe the reward obtained
15:     Update parameters for the chosen arm:
16:       Update the posterior distribution of the selected arm based on observed reward according to Eq. (29)
17:       Update the reward for the chosen arm
18:     end for
19: Next round
20: until convergence or fixed number of rounds
```

user requests, we varied the shape parameter values of the Zipf distribution.

6.1. Exploring Caching Strategy Performance

The proposed strategy is compared with the following approaches: LRU, LFU, fuzzy logic, and ICSAD [30].

Figure 5 depicts the performance of the preceding algorithms in relation to Zipf parameters. Figure 5a shows the average hit rate as a function of the Zipf parameter for many caching or selection algorithms. The proposed algorithm (TS-MMCM) showed superior performance compared to all other algorithms, demonstrating continuous and significant improvement as the Zipf value grows from 1.0 to 1.6. This demonstrates that our algorithm efficiently gives priority to caching the most widely accessed content.

Its average hit rate grows from 0.3 to above 0.6, indicating a strong ability to adjust to the Zipf parameter-defined popularity distribution. This demonstrates that the proposed approach is more efficient in obtaining a greater hit rate when

Tab. 2. Simulation parameters.

Parameter	Value
Distance	2 [km]
Vehicle speed	20, 60 [km/h]
Request rate	20, 30 [requests/m]
Number of vehicle users	55
Number of file categories	5
Cache capacity rate	10, 15, 20, 25 [%]

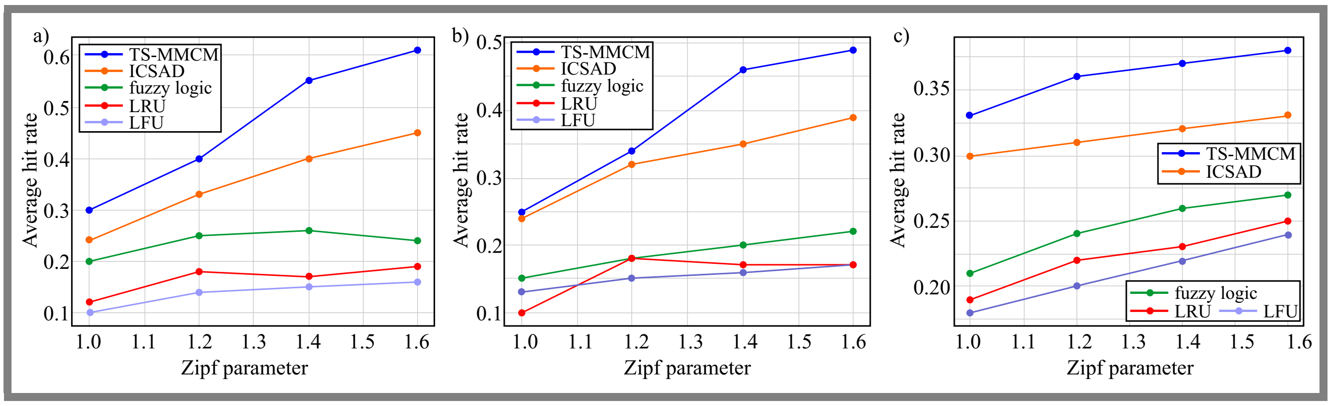

Fig. 5. Comparative analysis of caching strategies across different cache locations: a) for locale cache, b) for cluster head cache, and c) for edge cache.

the distribution of requests becomes more imbalanced (which is indicated by an increase in the Zipf parameter).

ICSAD is the algorithm that offers second best performance, showing a moderate rising trend. The average hit rate started at approximately 0.25 when the Zipf parameter was set to 1.0 and gradually increased to just below 0.5 when the Zipf parameter was set to 1.6. This hit rate is consistently lower than that of the TS-MMCM algorithm in all cases. Fuzzy logic showed a rather flat performance with a slight upward trend. It started at an average hit rate of approx. 0.2 and ended at approx. 0.25. Although performance improved as the Zipf parameter increased, the gains are not as great as those observed when using our algorithm or ICSAD.

LRU, the traditional caching strategy, followed a slight upward trend, starting just above a hit rate of 0.1 and approaching 0.2 as the Zipf parameter increased to 1.6. This suggests its modest sensitivity to popularity distribution.

The LFU algorithm exhibited the least amount of improvement, indicating it may be least suitable for adapting to changes in content popularity distributions. Figure 5b illustrates the effectiveness of different caching algorithms in relation to the Zipf parameter within the cluster head cache. Our approach routinely surpasses ICSAD, fuzzy logic, LRU, and LFU approaches in terms of performance, with a noticeable positive relationship between the average hit rate and the Zipf parameter. The initial hit rate of the proposed algorithm equaled approx. 0.25 and it gradually increased to nearly 0.5. In contrast, the remaining algorithms demonstrated different levels of improvement as the Zipf parameter increased, but none of them surpassed a hit rate of 0.4. The figure indicates that TS-MMCM easily adjusts to various data request distributions, as evidenced by the Zipf parameter. Figure 5c shows a comparison of the performance of hit rate-based algorithms in the edge cache as the Zipf parameter increased from 1.0 to 1.6.

Performance of the proposed algorithm surpassed that of the others, demonstrating a consistent rise in the hit rate as the Zipf parameter increased. The ICSAD algorithm is ranked second, with the fuzzy logic, LRU, and LFU algorithms following in that order. The disparity in performance among algorithms indicates different degrees of efficiency, with the conventional LRU and LFU caching techniques falling behind more recent approaches. The figure likely illustrates the superiority of a new algorithm in the context of data caching or retrieval, driven by the distribution of data, as indicated by Zipf.

Figure 6 shows the relationships between the cache hit ratio of the aforementioned algorithms and the overall caching capacity rate. The algorithms are compared at capacity rates of 0.1, 0.15, 0.2, and 0.25. These capacity rates indicate the combined cache capacity rate of the local cache, cluster head, and edge server. The cache hit ratio is the ratio of requests successfully retrieved from the cache to the total number of requests made.

The proposed algorithm offers exceptional performance, since it consistently achieves the greatest hit rates across all capacity rates. The frequency of successful cache retrievals improved as the capacity rate increased, indicating that it efficiently utilized the available cache space to store frequently requested content.

The ICSAD algorithm demonstrates a correlation between the hit rate and the capacity rate, but it fell short of achieving the same degree of performance as TS-MMCM, albeit it ranked as the second most effective method. Performance of the fuzzy logic algorithm was inferior to that of the proposed algorithm and ICSAD at the 0.1 capacity rate, but increased at the 0.15 capacity rate. However, the improvement is not as great at

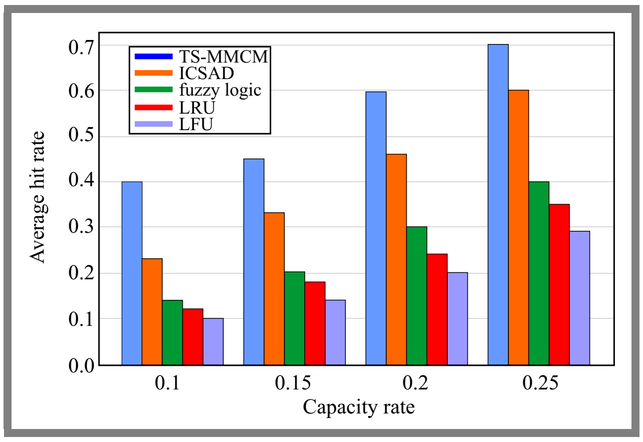

Fig. 6. Cache hit ratio versus total caching capacity rate.

higher capacity rates when compared with TS-MMCM and ICSAD.

The LRU method constantly exhibited worse performance when compared to TS-MMCM, ICSAD, and fuzzy logic. However, it surpassed the LFU algorithm. The hit rate increased when the capacity rate escalated, although it stayed below 0.4 even at the greatest capacity.

The LFU algorithm initially had the lowest capacity rate of 0.1, but it showed a progressive rise in performance as the capacity rate increased. However, it consistently remained the least-performing algorithm compared to other approaches at all capacity rates.

As the capacity rate increased, the average hit rate for each algorithm also increased. This is an expected behavior as greater capacity generally means that more data can be stored, which is likely to translate into higher success rates.

This is because more content may be cached with a bigger caching capacity rate, allowing the local cache, cluster head, and edge server to respond to more requests from the vehicle users. The TS-MMCM algorithm and ICSAD had the highest average hit rates at a capacity rate of 0.25, suggesting that they are more effective at making use of bigger capacity rates than fuzzy logic, LRU, and LFU.

In general, Fig. 6 shows that the TS-MMCM algorithm and ICSAD are the most effective solutions in terms of hit rate at the capacity rates tested, with the proposed algorithm outperforming ICSAD. Meanwhile, LFU seems to be the least effective across all capacity rates.

6.2. Cache Hit Rate and Latency Optimization via Advanced Clustering Technique

To improve the accuracy of content popularity determination and to increase the cache hit rate even further, this paper described a procedure in which the LSTM method is used to predict content requests. We evaluated the effectiveness of vehicle grouping according to speed and position. Vehicles were grouped into clusters of three, six, and nine, and their performance was analyzed based on the request rate, using a Zipf parameter of 1.0 and setting the total cache capacity rate at 0.15.

Figure 7 illustrates the correlation between the cache hit rate in a cooperative caching system and the rate at which requests or data requests are made. The figure consists of three lines, each indicating a distinct number of clusters. The y-axis displays the hit rate, which is the fraction of requests that the cache serves without having to retrieve data from the origin server. A higher hit rate signifies superior performance, as it implies a larger number of requests being promptly served from the cache. The x-axis represents the request rate, which is assumed to be a measure of how often requests are submitted to the cache. The rate is often quantified as the number of requests per unit of time.

Cluster 3 exhibited the lowest hit rate among all request rates, initially equaling 0.82 and gradually increasing to 0.92 as the request rate rose from 20 to 30. These findings indicate that cluster 3 is the least effective among the three in handling

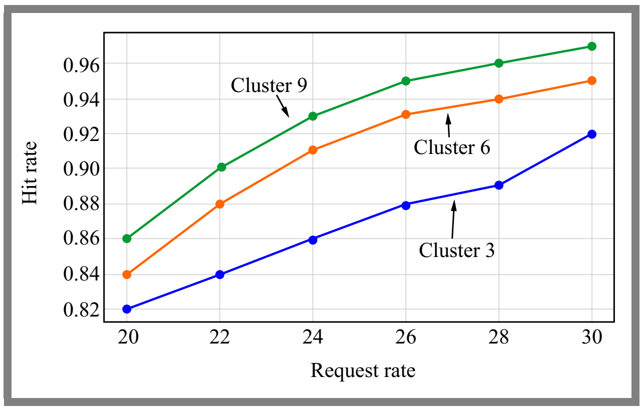

Fig. 7. Cache hit ratio vs. request rate.

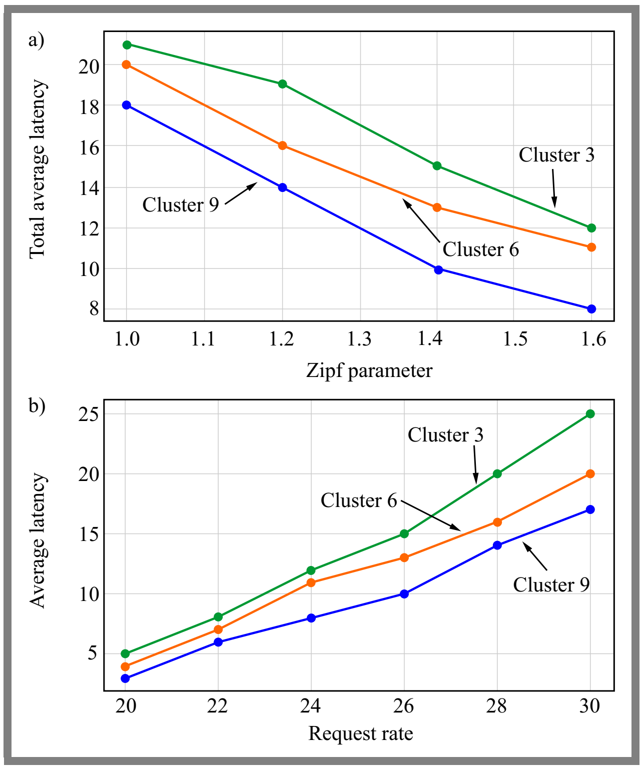

Fig. 8. Vehicle clustering performance: a) total average latency vs. Zipf parameter and b) average latency vs. request rate.

cache requests, although it demonstrated progress as the rate of requests increased. The hit rate of cluster 6 initially stood at 0.84 and gradually rose to just above 0.94 when the request rate escalated from 20 to 30. It surpassed cluster 3 in terms of performance at all request rate levels.

Cluster 9 exhibited the highest hit rate, starting at 0.86 and peaking at nearly 0.97 during the highest request rate shown. It offered superior performance compared to the two other clusters.

All clusters experienced an increase in their hit rates as the request rate rose. This indicates that the cooperative caching mechanism is efficient in using the higher volume of requests to accurately anticipate and cache popular content. Increased request rates can enhance the precision of predicting popular items, allowing for prefetching or caching and thus resulting in an improved hit rate.

Overall, Fig. 7 demonstrates that in a cooperative caching framework, an increase in the request rate leads to a corresponding rise in the hit rate across all clusters. Cluster 9 exhibited the highest performance, while cluster 3 showed the lowest performance.

Figure 8 shows the total average latency based on the Zipf parameter, as well as the average latency based on the request rate for three cluster sizes.

It follows from Figure 8a that the total average latency decreases as the number of clusters increases, indicating that vehicle clustering may efficiently reduce this time, while Fig. 8b demonstrates how the proposed clustering architecture might lead to a significant decrease in latency. Increasing the number of clusters can further reduce average latency.

Simulation results show that TS-MMCM algorithm significantly improves IoV caching system's performance compared to that achieved with the use of previous algorithms, enabling efficient adjustment to rapidly evolving network structures in time-sensitive IoV environments.

7. Conclusions

This study shows that intentional vehicle clustering combined with machine learning to forecast content popularity enhances both system performance and user experience by enabling better caching decisions and fostering more dependable communication links within vehicular networks. The goal of the proposed future study is to apply sophisticated machine learning models, energy efficiency considerations, and multidisciplinary research to ensure the model's practical feasibility in real-world scenarios while simultaneously enhancing the caching model's scalability, efficiency, and security.

References

[1] B. Radouane, G. Lyamine, K. Ahmed, and B. Kamel, "Scalable Mobile Computing: From Cloud Computing to Mobile Edge Computing", *2022 5th International Conference on Networking, Information Systems and Security: Envisage Intelligent Systems in 5G/6G-based Interconnected Digital Worlds (NISS)*, Bandung, Indonesia, 2022 (https://doi.org/10.1109/NISS55057.2022.10085600).

[2] W. Jiang *et al.*, "Multi-agent Reinforcement Learning for Efficient Content Caching in Mobile D2D Networks", *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1610–1622, 2019 (https://doi.org/10.1109/TWC.2019.2894403).

[3] Z. Yang, Y. Liu, Y. Chen, and L. Jiao, "Learning Automata Based Q-learning for Content Placement in Cooperative Caching", *IEEE Transactions on Communications*, vol. 68, no. 6, pp. 3667–3680, 2020 (https://doi.org/10.1109/TCOMM.2020.2982136).

[4] X. Fang, T. Zhang, Y. Liu, and Z. Zeng, "Multi-agent Cooperative Alternating Q-learning Caching in D2D-enabled Cellular Networks", *IEEE Global Communication Conference (GLOBECOM)*, Waikoloa, USA, 2019 (https://doi.org/10.1109/GLOBECOM38437.2019.9014053).

[5] Y. Qian *et al.*, "Reinforcement Learning-based Optimal Computing and Caching in Mobile Edge Network", *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2343–2355, 2020 (https://doi.org/10.1109/JSAC.2020.3000396).

[6] P. Liu, Y. Zhang, T. Fu, and J. Hu, "Intelligent Mobile Edge Caching for Popular Contents in Vehicular Cloud Toward 6G", *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 5265–5274, 2021 (https://doi.org/10.1109/TVT.2021.3076304).

[7] W. Qi *et al.*, "Extensive Edge Intelligence for Future Vehicular Networks in 6G", *IEEE Wireless Communications*, vol. 28, no. 4, pp. 128–135, 2021 (https://doi.org/10.1109/MWC.001.2000393).

[8] J. Shi *et al.*, "A Novel Deep Q-learning-based Air-assisted Vehicular Caching Scheme for Safe Autonomous Driving", *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4348–4358, 2021 (https://doi.org/10.1109/TITS.2020.3018720).

[9] Z. Zhang, C.-H. Lung, M. St-Hilaire, and I. Lambadaris, "Smart Proactive Caching: Empower the Video Delivery for Autonomous Vehicles in ICN-based Networks", *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7955–7965, 2020 (https://doi.org/10.1109/TVT.2020.2994181).

[10] Y. Liu and B. Mao, "On a Novel Content Edge Caching Approach Based on Multi-Agent Federated Reinforcement Learning in Internet of Vehicles", *2023 32nd Wireless and Optical Communications Conference (WOCC)*, Newark, USA, 2023 (https://doi.org/10.1109/WOCC58016.2023.10139417).

[11] A. Ndikumana *et al.*, "Deep Learning Based Caching for Self-driving Cars in Multi-access Edge Computing", *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 5, pp. 2862–2877, 2021 (https://doi.org/10.1109/TITS.2020.2976572).

[12] Z. Zhu *et al.*, "Proactive Caching in Auto Driving Scene via Deep Reinforcement Learning", *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, Xi'an, China, 2019 (https://doi.org/10.1109/WCSP.2019.8928131).

[13] A. Ndikumana and C.S. Hong, "Self-driving Car Meets Multi-access Edge Computing for Deep Learning-based Caching", *2019 International Conference on Information Networking (ICOIN)*, Kuala Lumpur, Malaysia, 2019 (https://doi.org/10.1109/ICOIN.2019.8718113).

[14] W. Jiang, G. Feng, S. Qin, and Y.-C. Liang, "Learning-based Cooperative Content Caching Policy for Mobile Edge Computing", *ICC 2019 – 2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, 2019 (https://doi.org/10.1109/ICC.2019.8761121).

[15] C. Zhang *et al.*, "Toward Edge-assisted Video Content Intelligent Caching with Long Short-term Memory Learning", *IEEE Access*, vol. 7, pp. 152832–152846, 2019 (https://doi.org/10.1109/ACCESS.2019.2947067).

[16] Y. Ye, M. Xiao, and M. Skoglund, "Mobility-aware Content Preference Learning in Decentralized Caching Networks", *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 1, pp. 62–73, 2020 (https://doi.org/10.1109/TCCN.2019.2937519).

[17] Y. Zhang *et al.*, "Cooperative Edge Caching: A Multi-agent Deep Learning Based Approach", *IEEE Access*, vol. 8, pp. 133212–133224, 2020 (https://doi.org/10.1109/ACCESS.2020.3010329).

[18] X. Xu, M. Tao, and C. Shen, "Collaborative Multi-agent Multi-armed Bandit Learning for Small-cell Caching", *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2570–2585, 2020 (https://doi.org/10.1109/TWC.2020.2966599).

[19] R. Wang *et al.*, "Cooperative Caching Strategy with Content Request Prediction in Internet of Vehicles", *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 8964–8975, 2021 (https://doi.org/10.1109/JIOT.2021.3056084).

[20] X. Bi and L. Zhao, "Collaborative Caching Strategy for RL-based Content Downloading Algorithm in Clustered Vehicular Networks", *IEEE Internet of Things Journal*, vol. 10, no. 11, pp. 9585–9596, 2023 (https://doi.org/10.1109/JIOT.2023.3235661).

[21] T. Kanungo *et al.*, "An Efficient K-means Clustering Algorithm: Analysis and Implementation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002 (https://doi.org/10.1109/TPAMI.2002.1017616).

[22] D. Zhang *et al.*, "New Multi-hop Clustering Algorithm for Vehicular Ad Hoc Networks", *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 4, pp. 1517–1530, 2019 (https://doi.org/10.1109/TITS.2018.2853165).

[23] C. Zhang *et al.*, "Deep Transfer Learning for Intelligent Cellular Traffic Prediction Based on Cross-domain Big Data", *IEEE Journal*

on Selected Areas in Communications, vol. 37, no. 6, pp. 1389–1401, 2019 (https://doi.org/10.1109/JSAC.2019.2904363).

[24] C. Olah, *Understanding LSTM Networks*, Github, 2015 [Online] Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[25] C. Wang, Z. Zhao, Q. Sun, and H. Zhang, "Deep Learning-based Intelligent Dual Connectivity for Mobility Management in Dense Network", *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, Chicago, USA, 2018 (https://doi.org/10.1109/VTC-Fall.2018.8690554).

[26] G. Barlacchi *et al.*, "Multi-source Dataset of Urban Life in the City of Milan and the Province of Trentino", *Scientific Data*, no. 2, art. no. 150055, 2015 (https://doi.org/10.1038/sdata.2015.55).

[27] Y. Cui, Y. Liang, and R. Wang, "Resource Allocation Algorithm with Multi-platform Intelligent Offloading in D2D-enabled Vehicular Networks", *IEEE Access*, vol. 7, pp. 21246–21253, 2019 (https://doi.org/10.1109/ACCESS.2018.2882000).

[28] S. Agrawal and N. Goyal, "Thompson Sampling for Contextual Bandits with Linear Payoffs", *ArXiv*, 2012 (https://doi.org/10.48550/arXiv.1209.3352).

[29] Y. Chen, J. Liu, Y. Shi, and M. Sheng, "Prefetch and Cache Replacement Based on Thompson Sampling for Satellite IoT Network", *ICC 2021 – IEEE International Conference on Communications*, Montreal, Canada, 2021 (https://doi.org/10.1109/ICC42927.2021.9500508).

[30] L. Zhao *et al.*, "Intelligent Content Caching Strategy in Autonomous Driving Toward 6G", *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 9786–9796, 2022 (https://doi.org/10.1109/TITS.2021.3114199).

Radouane Baghiani, Ph.D. student
Computer Sciences and Information Technologies Department
https://orcid.org/0009-0001-7694-7204
E-mail: baghiani.radouane@univ-ouargla.dz
LINATI Laboratory, Kasdi Merbah University, Ouargla, Algeria
https://www.univ-ouargla.dz

Lyamine Guezouli, Ph.D.
Renewable Energies and New Technologies Department
https://orcid.org/0000-0002-7406-7633
E-mail: lyamine.guezouli@hns-re2sd.dz
LEREESI Laboratory, HNS-RE2SD, Batna, Algeria
http://www.hns-re2sd.dz

Ahmed Korichi, Professor
Computer Sciences and Information Technologies Department
https://orcid.org/0000-0002-5741-4963
E-mail: ahmed.korichi@univ-ouargla.dz
LINATI Laboratory, Kasdi Merbah University, Ouargla, Algeria
https://www.univ-ouargla.dz