electronics

Article

Enhanced In-Network Caching for Deep Learning in Edge Networks

Jiaqi Zhang [1,2], Wenjing Liu [3,4,*], Li Zhang [1,5] and Jie Tian [6]

[1] Department of Computer Education, Baicheng Medical College, Baicheng 137701, China; 20191100104@imut.edu.cn (J.Z.); zhangli@bcmc.edu.cn (L.Z.)
[2] College of Information Engineering, Inner Mongolia University of Technology, Hohhot 010051, China
[3] College of Data Science and Application, Inner Mongolia University of Technology, Hohhot 010051, China
[4] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[5] College of Computer and Technology, Jilin University, Changchun 130015, China
[6] Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA; jt66@njit.edu
* Correspondence: liuwenjing2015@bupt.edu.cn

Abstract: With the deep integration of communication technology and Internet of Things technology, the edge network structure is becoming increasingly dense and heterogeneous. At the same time, in the edge network environment, characteristics such as wide-area differentiated services, decentralized deployment of computing and network resources, and highly dynamic network environment lead to the deployment of redundant or insufficient edge cache nodes, which restricts the efficiency of network service caching and resource allocation. In response to the above problems, research on the joint optimization of service caching and resources in the decentralized edge network scenario is carried out. Therefore, we have conducted research on the collaborative caching of training data among multiple edge nodes and optimized the number of collaborative caching nodes. Firstly, we use a multi-queue model to model the collaborative caching process. This model can be used to simulate the in-network cache replacement process on collaborative caching nodes. In this way, we can describe the data flow and storage changes during the caching process more clearly. Secondly, considering the limitation of storage space of edge nodes and the demand for training data within a training epoch, we propose a stochastic gradient descent algorithm to obtain the optimal number of caching nodes. This algorithm entirely takes into account the resource constraints in practical applications and provides an effective way to optimize the number of caching nodes. Finally, the simulation results clearly show that the optimized number of caching nodes can significantly improve the adequacy rate and hit rate of the training data, with the adequacy rate reaching 84% and the hit rate reaching 100%.

Keywords: deep learning in edge networks; in-network caching; optimization configuration; queuing theory

Citation: Zhang, J.; Liu, W.; Zhang, L.; Tian, J. Enhanced In-Network Caching for Deep Learning in Edge Networks. *Electronics* **2024**, *13*, 4632. https://doi.org/10.3390/electronics13234632

Academic Editor: Carlo Mastroianni

Received: 16 October 2024
Revised: 22 November 2024
Accepted: 22 November 2024
Published: 24 November 2024

1. Introduction

Industry 5.0 has favored edge computing and AI technologies more frequently in recent years. The driving forces behind this decision are the expansion of Industrial IoT connected manufacturing, Industrial IoT resources, data analysis techniques that require less power-consuming computing equipment, and the demand for real-time decision-making in Industry 5.0 [1]. With the extension and expansion of the network center, edge computing technology is another major network construction change. The traditional cloud computing method is to pass all the data to the remote data center for processing through the core network, which will not only cause the network center to face enormous communication pressure, but also cause a significant delay due to long-distance data transmission. Compared with cloud computing, edge computing can not only relieve bandwidth pressure but

also ensure real-time data transmission by deploying the network, storage, and computing resources in a distributed manner at the edge of the network and in processing the request tasks submitted by terminal devices [2,3]. Edge computing technology has the following characteristics: First, it can reduce the load pressure of network bandwidth and sink request tasks to edge cache nodes. Second, it can process request tasks at edge cache nodes, thereby reducing transmission delay and satisfying delay-sensitive applications. Third, it can effectively protect private data by taking the authorized entity as the trust center and integrating multi-trust domain sharing. In general, edge computing sinks some essential services to edge cache nodes for caching. It provides services to users through distributed deep learning on edge servers, which is an intermediate stage between cloud computing centers and personal computing centers [4]. The distributed deep learning framework in the network is shown in Figure 1.

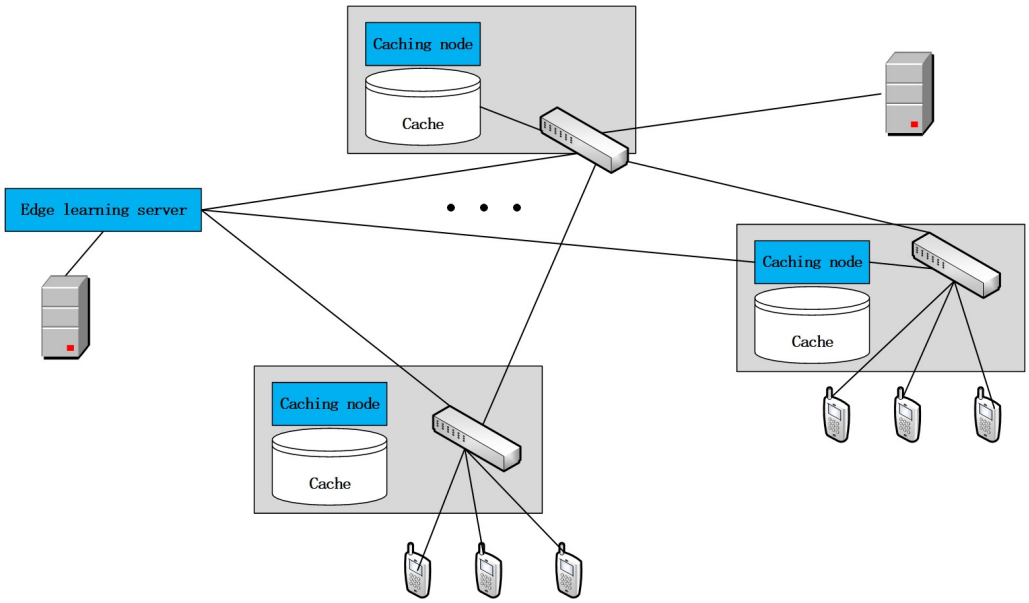

Figure 1. In-network caching for distributed deep learning.

At present, edge computing combined with neural network models is mainly used in the following five aspects, and these applications have significant implications for cache adequacy rate and hit rate within the system: (1) The application of edge deep learning [5,6]: In this context, edge deep learning is often employed in scenarios where real-time data processing is crucial. For instance, in Industrial Internet of Things (IIoT) applications, sensors constantly generate massive amounts of data. When using edge deep learning for tasks like predictive maintenance, a high cache adequacy rate is essential. This ensures that the necessary historical data and model parameters related to equipment operation are readily available in the cache. The hit rate also plays a vital role as it determines the efficiency of accessing the cached data during the decision-making process of the deep learning model. If the cache hit rate is low, it may lead to delays in analyzing sensor data and identifying potential equipment failures. (2) The reasoning of edge deep learning [7–10]: Edge deep-learning reasoning requires quick access to relevant data and models stored in the cache. Consider an intelligent surveillance system in an urban environment. The deep learning model needs to rapidly retrieve cached data related to object recognition patterns and previous surveillance events. A sufficient cache adequacy rate guarantees that the model has all the necessary information for accurate reasoning. Meanwhile, a high hit rate ensures that the data retrieval process is seamless, enabling the system to quickly make inferences about potential security threats or abnormal situations based on the real-time video feed. (3) The training of edge deep learning [11,12]: During edge deep learning training, a large amount of data is required to optimize the model. In applications such as autonomous vehicles, where edge computing is used to train the vehicle's decision-making

system, the cache adequacy rate directly affects the availability of training data. If the cache can store sufficient data related to different driving scenarios, road conditions, and traffic patterns, the training process will be more effective. Additionally, a high cache hit rate reduces the time spent on data fetching from other sources, accelerating the training process and improving the overall performance of the deep learning model for better decision-making in complex driving situations. (4) The application of edge deep learning deployment [13–15]: When deploying edge deep learning applications, ensuring a proper cache environment is crucial. For example, in a distributed healthcare system where edge devices are used for patient monitoring and diagnosis, the cache adequacy rate determines whether the deep learning models on these devices have access to comprehensive patient data, including a history of vital signs and previous diagnosis records. A high hit rate allows for quick access to these data during the deployment of diagnostic models, enabling timely and accurate medical decisions. This is especially important in emergency situations where every second counts. (5) Deep learning for edge computing optimization [16–18]: Edge computing optimization using deep learning aims to improve the overall efficiency of the system. In scenarios like intelligent grid management, where numerous edge nodes handle power consumption data and grid status information, a high cache adequacy rate ensures that the deep learning models have sufficient data for optimizing power distribution and predicting demand. The cache hit rate affects the speed at which these models can access and process the data, enabling a more efficient control of the power grid and reducing energy waste. Overall, optimizing cache adequacy rate and hit rate in these applications can significantly enhance the performance of edge computing combined with neural network models. Usually, after the neural network model is deployed on the edge server, to ensure the hit rate and adequate rate of the cached training data and improve the efficiency of model training and update, it is necessary to analyze the distribution of training data in the edge cache nodes. Based on this, edge computing needs to use multiple edge cache nodes to perform the distributed processing of computing tasks through operations such as caching [19,20]. Due to the unbalanced resource allocation of edge nodes and the small number of resources, edge caching still faces challenges [21]. It is necessary to analyze the impact of the number of nodes caching data on model training and configure a reasonable number of edge cache nodes for the cached data. When the configured number of edge cache nodes is too large, there will be idle edge cache nodes, resulting in a waste of resource allocation; on the contrary, when the configured number of edge cache nodes is too small, the pre-cached data will be inadequate, and it is difficult to support the neural network in the edge server. The model training of the network will thus be inadequate, thereby reducing the performance of the network cache. Therefore, configuring the optimal number of edge cache nodes for different Internet tasks is a critical issue in improving the performance of in-network caches.

To improve the performance of in-network caching, this paper establishes a multi-edge node training data collaborative caching model to simulate the in-network cache replacement process of edge caching nodes. In the period of model training, based on the constraints of maximizing cache utilization, a stochastic gradient descent algorithm is designed to solve the optimal number of edge cache nodes. The main contributions of this paper are as follows:

1. Based on the multi-queue model, an in-network collaborative caching model of multi-edge cache nodes is established to simulate the in-network cache replacement process of edge cache nodes.
2. In the period of model training, based on the constraints of maximizing cache utilization, a stochastic gradient descent algorithm is designed to find the optimal number of edge cache nodes.
3. The sim4DistrDL-based simulation results demonstrate that the hit rate of cached training data can be improved and the proposed scheme can efficiently support the training process of deep learning models at the edge.

Section 2 introduces related work on caching; Section 3 describes the in-network caching model for multiple-edge-node collaborative deep learning; Section 4 evaluates the effectiveness of the enhanced in-network caching scheme for multiple-edge-node collaborative deep learning through simulation; and Section 5 concludes the whole article.

2. Related Work

To improve the caching performance and the efficiency of neural network training in edge networks, researchers at home and abroad have conducted extensive and in-depth studies. They start from different angles to explore new caching strategies and optimization algorithms, with the aim of reducing data transmission latency and improving cache hit rate. At the same time, they improve the training methods of neural networks to adapt to the resource constraints and real-time requirements of edge computing environments. Among the various research efforts, Zahra Aghapour et al. proposed a method based on deep reinforcement learning. This method divides offloading and resource allocation into two sub-problems: the algorithm updates the offloading policy according to environmental information, and it optimizes resource allocation with the Salp SwarmAlgorithm [22]. Qianlin Liang et al. designed analytic models to capture the performance of DNN inference workloads on shared edge accelerators, such as GPU and edgeTPU, under different multiplexing and concurrency behaviors [23]. Zhiyuan Wang et al. designed and implemented an FL system called CoopFL, which trains DNNs through the cooperation between devices and edge servers [24]. Zhenyang Shu et al. modeled a relay-assisted multi-access edge computing (MEC) framework, employing multi-hop transmission to enable cross-domain service coverage [25]. Avilia Kusumaputeri Nugroho et al. presented NAFEOS, a proposed solution formulated as a two-stage algorithm that can effectively integrate association optimization with vertical and horizontal scaling. The Stage-1 problem optimizes the user–base station association and federation assignment so that the edge servers can be utilized in a balanced manner. The following Stage-2 dynamically schedules both vertical and horizontal scaling so that the fluctuating task offloading demands from users are fulfilled [26]. Zheng Lin et al. advocated for the integration of the edge computing paradigm and parallel split learning (PSL), allowing multiple edge devices to offload substantial training workloads to an edge server via layer-wise model splitting [27]. Yonglin Pu et al. drew on the domino effect and the combined impact of task migration. A dynamic node-growing algorithm (DNGA) and a dynamic node-shrinking algorithm (DNSA), both of which are grounded in the task migration strategy, are put forward [28]. Yishan Chen et al. built a flexible and generic DAG task model to support the associative task offloading process with complex data dependencies in IoT edge computing environments. Additionally, a priority-based DAG task offloading algorithm and a secondary resource allocation algorithm are proposed to minimize the response delay and improve the resource utilization of edge servers [29]. Linbo Liao et al. proposed an online algorithm, namely, the double reinforcement learning computation offloading (DRLCO) algorithm, which jointly decides the offloading decision; the CPU frequency; and transmit power for computation offloading [30]. Binbin Huang et al. adopted the graph convolution neural network to transform multiple workflow instances with different shapes and sizes into embeddings and formulate the online multiple workflow scheduling problem as a finite Markov decision process. Furthermore, they proposed a policy gradient learning-based online multiple-workflow scheduling scheme (PG-OMWS) to optimize the average completion time of all workflow instances [31]. Xingwang Wang et al. presented EHCI, an on-edge high-throughput collaborative inference framework for real-time video analytics. On the mobile device, EHCI crops the critical regions from the current video frame based on the local detection cache and offloads these regions to the edge server, which can significantly reduce bandwidth consumption and computation costs [32]. The above literature studies present issues such as the joint optimization of task offloading and resource allocation for different MEC scenarios. However, the above research ignores the impact of the relationship between the deployment number of edge nodes and service caching on performance. At the same time, centralized control manage-

ment schemes make it difficult to meet the application requirements of ultra-large capacity and ultra-low latency in edge networks, which also brings challenges to the integration and scheduling of multi-dimensional network resources. Therefore, a more flexibly distributed resource management scheme is needed.

Deep learning methods learn low-dimensional codes from high-dimensional input vectors and are widely used in many fields, such as image recognition, and achieve remarkable results. In addition, the queuing theory is a well-known method that can be applied to analyze and optimize systems. Although it is not a traditional supervised feature learning method, in the context of deep learning, it has been explored to find concise and high-level representations of complex data and has been widely studied to realize deep learning. Based on previous studies, this paper proposes an enhanced in-network caching scheme for deep learning in edge networks. Its purpose is to analyze and evaluate the optimal configuration number of edge cache nodes. This is achieved by establishing a multi-queue model that simulates the cache in the network, aiming to effectively improve the cache performance in the edge network.

3. In-Network Cache Modeling for Deep Learning in Edge Networks

In order to study the multi-edge node training data collaborative caching modeling, this section first presents the notation and problem definition. Table 1 describes the symbols used in this article.

Table 1. Description of symbols.

Symbol	Description
D	the maximum capacity of a single edge node
λ	average arrival speed
ω	average response time
π	steady-state probability
π_i	the steady-state probability of the i-th node
c	the number of deployed edge nodes
ρ	replacement strength of an edge cache node
W_s	average sojourn time
L_s	average queue length
h	awaiting the limiting value of the average queue length

Currently, the Least Recently Used (LRU) [33,34] algorithm is the most widely used in caching. The idea of the algorithm is that when there is a referenced page in memory, the content of the list needs to be separated, and the page is moved to the head of the cache queue; on the contrary, it is directly cached. Therefore, the execution process of the algorithm in the multi-edge cache node is as follows: When there is a new data request, first, it is judged whether the data exist in the cache queue of the multi-edge cache node. Second, if data exist, move them to the head of the multi-edge cache node's cache queue; otherwise, insert them directly into the head of the queue. Since the request probability of each datum is the same and has no aftereffect, the LRU cache queue of multiple edge cache nodes can be abstracted into a Markov chain model.

3.1. Cached Data Replacement Model for Edge Nodes

In order to identify and analyze the relationship between the sojourn time of data caching and the queue length of data caching in multi-edge nodes, a multi-queue hierarchical collaboration model that simulates in-network caching is established. Since the data caching process is memoryless and follows a Markov chain, queuing theory algorithms can be adopted. To facilitate the calculation, it is first assumed that the cached data of the edge cache nodes obey a discrete probability distribution. Secondly, due to the limited

cache capacity of a single edge node, according to the limited algorithm of queuing theory, the multi-queue hierarchical collaboration model should satisfy M1/M2/c/D, where M1 represents the distribution of the data request interval; M2 represents the distribution of the response time; c represents the number of deployed edge nodes; and D represents the maximum capacity of a single edge node. Finally, a model for simulating the replacement of cached data in the edge network is established to discover and analyze the relationship between the sojourn time of the cached data in the node and the queue length. The difference equation for this model is expressed as follows:

$$\begin{cases} \lambda\pi_{n-1} - [\lambda + n(\varpi)]\pi_n + (n+1)(\varpi)\pi_{n+1} = 0 \\ \lambda\pi_{n-1} - [\lambda + c(\varpi)]\pi_n + c(\varpi)\pi_{n+1} = 0 \\ \lambda\pi_{D-1} - c(\varpi)\pi_D = 0 \\ \lambda\pi_0 - (\varpi)\pi_1 = 0 \end{cases} \tag{1}$$

Among them, n represents the number of data that need to be cached, and π_D represents the steady state of the node cache capacity (that is, the speed of data caching and releasing is balanced and does not exceed the node capacity limit). According to the queuing theory model, the replacement strength ρ of an edge cache node is

$$\rho = \frac{\lambda}{\varpi} \tag{2}$$

The steady-state probabilities of the node caching data and releasing data are

$$\pi_0 = \left[\sum_{i=0}^{c-1} \frac{\rho^n}{n!} + \frac{\rho^c\left(1-\rho_c^{D-c+1}\right)}{c!(1-\rho_c)}\right]^{-1} \quad 0 < \rho_c < 1 \tag{3}$$

The replacement strength ρ_c of c edge cache nodes is

$$\rho_c = \frac{c\lambda}{\varpi} \tag{4}$$

The steady-state probabilities of caching and releasing pieces of data in the node are

$$\pi_n = \frac{\rho^n}{c!c^{n-c}}\pi_0 \tag{5}$$

The average queue length L_s of c edge cache nodes is

$$L_s = \sum_{n=1}^{c}(n-1)\pi_n \tag{6}$$

The average sojourn time W_s of c edge cache nodes is

$$W_s = \frac{L_S}{\lambda(1-\pi_D)} \tag{7}$$

In the multi-edge node data cache replacement model based on the queuing theory, fewer edge cache nodes need to be deployed when the cache queue length of each node is constant, the training data remain for a long time, the required cache space is large, and more edge cache nodes need to be deployed. Both cases will affect the performance of training data caching within the edge network, thereby affecting the training results of the model. Therefore, when the cache space (queue) of each node is determined based on the multi-queue cooperative cache model, the configuration number of edge cache nodes can be effectively optimized.

3.2. Cache Deployment Optimization for In-Network Caching

In order to optimize the configuration number of edge cache nodes, this paper proposes a configuration algorithm based on stochastic gradient descent (enhanced stochastic gradient descent algorithm, ESGD) [35–37]. The idea of the algorithm is that when the maximum capacity of a single edge node is limited, the optimal relationship between the sojourn time and queue length in the multi-queue cooperative cache model is taken as the solution goal, and the idea of stochastic gradient descent is used to solve for the optimal node quantity.

For the convenience of calculation, the algorithm first assumes that the training period is T. Second, input parameters such as the stochastic gradient descent model, dataset, etc., are considered to find edge cache nodes where the data sojourn time and data queue length meet the requirements of the minimum time and adequacy for training data. Finally, within the period of model training, the optimal number of node configurations c is returned based on the constraint of maximizing the cache utilization. Among the parameters, when looking for an edge cache node that can meet the minimum time and adequate training data for the sojourn time and data queue length in the node, according to the limit theorem, there is ϵ, such that

$$\log \sum \mathrm{L}_s \otimes \mathrm{W}_s - \mathrm{h} < \varepsilon \tag{8}$$

The queue length limit h is expressed as

$$\mathrm{h} = \frac{d \log \sum \mathrm{L_S} \otimes \mathrm{W_S}}{dc_i} \tag{9}$$

This Algorithm 1 takes the dataset, learning rate, average arrival speed, average response speed, training period, replacement strength of an edge cache node, edge cache node-set, and maximum capacity of a single edge node as inputs, and the output is the optimal number of nodes. First, initialize the variables (Line 1), and calculate the steady-state probability that each edge cache node maintains a balanced storage space in the data caching process (Line 2). Then, calculate the average queue length, steady-state probability, average sojourn time, and length limit of the collaborative cache of n edge cache nodes (Lines 3–11). Next, traverse the n nodes in the edge cache node-set to determine the sojourn time and amount of data (queue length) in the edge cache node, and check whether the minimum time and adequacy of training data for the neural network are met (Line 12). If the conditions are met, update the optimal number of nodes; otherwise, return and break the loop. When all the nodes in the edge cache node-set have been traversed, jump out of the loop (Lines 13–17). Finally, return the value of c, which is the optimal number of nodes (Line 18). The convergence rate of the algorithm is related to the learning rate α. A higher learning rate α enables the model to converge faster, but it may also lead to under-fitting problems.

Algorithm 1 ESGD algorithm

Input:

Stochastic gradient descent model, dataset, learning rate α, average arrival speed λ, average response speed ϖ, training period T, replacement strength ρ of an edge cache node, edge cache node-set (n_0, n_1, ... , n_c), D;

Output:

c;

Initialization parameters c, ρ, L_s, W_s, c_i, b_i, n, D;

$\pi_0 = \left[\sum_{i=0}^{c-1} \frac{\rho^n}{n!} + \frac{\rho^c\left(1-\rho_c^{D-c+1}\right)}{c!(1-\rho_c)}\right]^{-1}$;

t = 0;

while (t $\leq$ T) **do**

    t++;

    for i in range(n) **do**

        $L_s = \sum_{n=1}^{c}(n-1)\pi_n$;

        $\pi_n = \frac{\rho^n}{c!c^{n-c}}$;

        $W_s = \frac{L_S}{\lambda(1-\pi_D)}$;

        $h = \frac{d\log\sum L_S \otimes W_S}{dc_i}$;

    end for

    if $\log\sum W_s \otimes L_s - h < \varepsilon$ **then**

        $c = c_i$;

        break;

    end if

    $c_{i+1} = c_i - \alpha * h$;

end while

return c

End

4. Simulation and Results Analysis

4.1. Simulation Environment

Hardware environment: Leno i7, 3.4G CPU, and 16G RAM. Software environment: Ubuntu 16.04, sim4DistrDL with kernel version 3.29, and C++ language. On the sim4DistrDL [38–40] simulation platform, the multi-edge node training data collaborative cache modeling was simulated and evaluated, and five edge network typologies were built. Five types of edge networks were constructed. As shown in Figure 2, Figure 2a illustrates that one edge server obtains training data from two edge cache nodes via one router. Figure 2b shows that one edge server acquires training data from three edge cache nodes via one router. Figure 2c indicates that one edge server obtains training data from four edge cache nodes via one router. Figure 2d demonstrates that one edge server fetches training data from five edge cache nodes via two routers. Figure 2e presents that one edge server retrieves training data from six edge cache nodes via three routers. In the real network, we observed the training process of popular deep learning models (MLP, AlexNet, VGG) on OpenNN and correspondingly set the simulation parameters according to the observation. Let the maximum cache capacity of each edge cache node be $D = 1024$ MB, the average speed of cached data be 1.36 KB/s, and the average speed of responding to server requests be 0.94 KB/s. In the entire edge network environment, the sending frequency of background traffic packets is set to 0.4 KB/s.

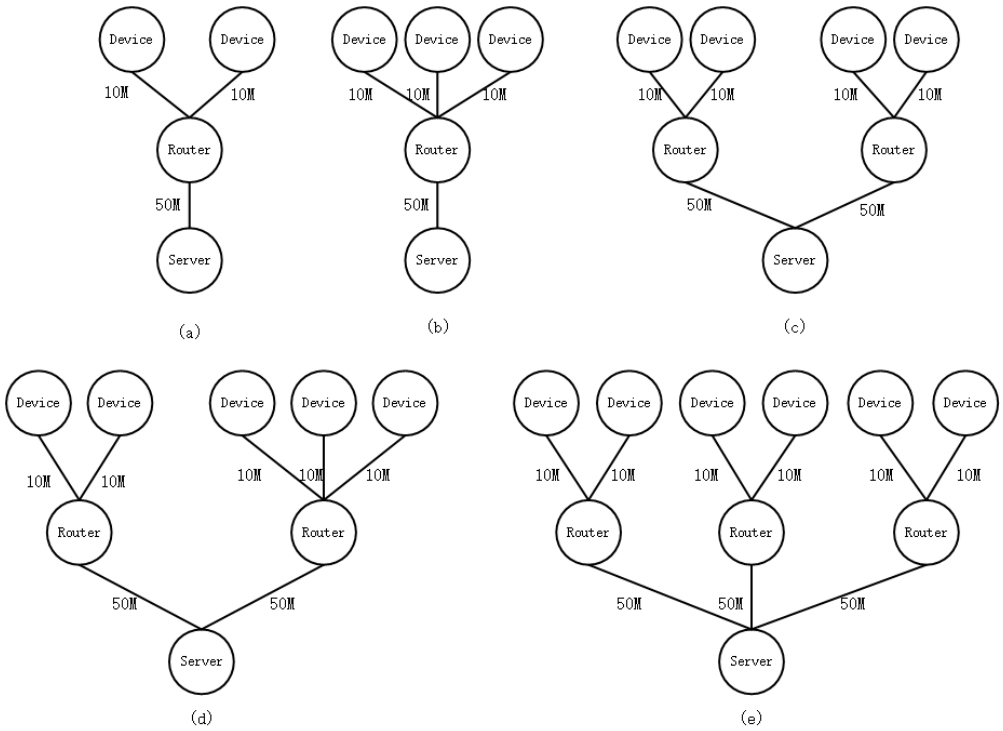

Figure 2. Topology diagrams.

4.2. Simulation Dataset

A dataset of twenty newsgroups [41] was used, which was published by Ken Lang in the Proceedings of the 12th International Conference on Machine Learning in 1995 and is one of the international standard datasets currently used for text classification, text mining, and information retrieval research. The dataset consists of 20,000 documents, approximately equally divided into 20 groups. In order to effectively evaluate the impact of the optimal configuration number of edge cache nodes on the cache performance in the edge network (data cache hit rate and adequate rate), 10 groups were used; each group contained 400 documents, corresponding to a topic number (documents with the same category of feature words were divided into the same group of topics). For example, the group number for alt. stheism is topic No. 0; the group number for comp. graphics is topic No.1; the comp.windows.x group is numbered as topic 2, etc., until the 10 groups are numbered sequentially. Since each file consists of text data and labels, it has typical characteristics. Therefore, the dataset can effectively simulate the hit rate and adequate rate of the neural network training model cache in the edge network under different hierarchical network typologies. A total of 4000 files were then cached in different hierarchical network topology nodes. The specific information of the dataset is shown in Table 2.

Table 2. The information of the dataset.

Topic	Topic Number	Number of Files
comp. graphics	1	400
comp. windows.x	2	400
misc. forsale	3	400
rec. autos	4	400
rec. sport. baseball	5	400
sci. electronic	6	400
sci. space	7	400
talk. politics. guns	8	400
talk. religion. misc	9	400

4.3. Evaluation Index

The effectiveness of the multi-edge node training data collaborative caching model is evaluated using the hit rate and the adequate rate. The hit rate is expressed as the number of hit packets divided by the number of cached data. The adequate rate is expressed as the amount of data requested for model training divided by the amount of cached data.

The hit rate formula is

$$\text{Hit rate} = \frac{N_r}{N} \tag{10}$$

The adequate rate formula is

$$\text{Adequate rate} = \frac{N_r^{\text{training}}}{N} \tag{11}$$

where N_r is the amount of data needed to satisfy the received request (number of hit packets), $N_r^{training}$ is the amount of data requested for model training, and N is the amount of cached data.

4.4. Simulation Competition Scheme

In different hierarchical network topologies, the comparison schemes include the enhanced stochastic gradient descent algorithm (ESGD), maximum popularity cache (MPC), and uniform cache (UC). The performance of these schemes in terms of hit rate and adequate rate is compared. Among them, MPC first ranks the popularity of the content, and then the ranking node caches the most popular content. UC is to cache content evenly across nodes. The scheme comparison is shown in Table 3.

Table 3. The steps of the scheme.

Scheme	Step
MPC	1. Rank the popularity of content
	2. Sorter caches the most popular content
	3. Cache content in nodes
UC	1. Ignore differences in content popularity
	2.The caching probability of content is uniformly distributed
	3. Cache content in nodes

4.5. Simulation Results

Ten groups of news data were adopted to analyze and compare the impact of the number of deployed nodes on data caching and model training in the edge network environment with different numbers of deployed nodes. As shown in Figure 3, when only two or three cache nodes are deployed in the edge network environment, the adequacy rates of ESGD, MPC, and UC are all relatively low. The reason for this phenomenon is that with a small number of deployed nodes, the nodes cannot cache sufficient data, resulting in the cached data being insufficient to support model training. When four cache nodes are deployed in the edge network environment, the adequacy rates of ESGD, MPC, and UC can all meet the minimum adequacy rate requirement of 70%. The minimum adequacy rate is determined based on the empirical rule, that is, it is based on the number of observed training samples and the performance of the training model, and it represents the minimum amount of data used for model training. When five or six cache nodes are deployed and when using the ESGD and MPC models, the data cached in the nodes are relatively sufficient to support model training. However, when using the UC model to cache data, it fails to meet the minimum adequacy rate requirement for model training. The reason for this phenomenon is that the ESGD model has restrictions on the cache queue length and the data residence time in the nodes. The MPC model caches and replaces stale data based on the popularity of the data, while the UC model adopts uniform caching and

cannot cache data according to user preferences. Moreover, the data cached in the nodes have no characteristics, and there is a large amount of redundant data.

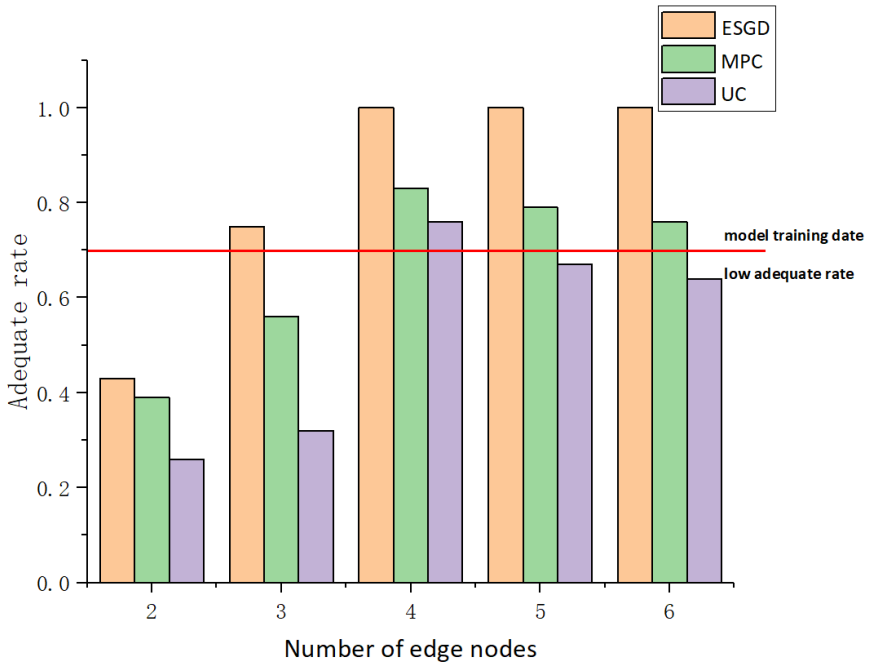

Figure 3. The result of adequate rate.

We conducted a more in-depth analysis and comparison of how the number of deployed nodes affects the cache hit rate via simulation. Under ideal circumstances, the cache hit rate is usually positively correlated with the number of deployed nodes, that is, the more nodes are deployed, the higher the hit rate. However, in practical scenarios, the redundant deployment of nodes can bring many problems. On the one hand, it will increase costs and cause a waste of resources; on the other hand, it will lead to performance degradation, and a large amount of data of no value for model training will be stored in the nodes. As shown in Figure 4, when only two or three nodes are deployed in the edge network environment for data caching and model training, the hit rates of the ESGD, MPC, and UC are all relatively low. The reason for this is that too few nodes result in limited cache capacity, and the data required by the neural network training model may not be fully stored. When data requests are made, a large number of requests cannot be satisfied in the cache, which requires data to be retrieved from other storage locations or may even fail to obtain accurate data, thereby negatively affecting the accuracy of model training and ultimately leading to a reduction in the cache hit rate. When four nodes are deployed in the edge network environment for data caching and model training, the cache hit rates of ESGD, MPC, and UC all increase. However, when five or six edge cache nodes are deployed, the cache hit rates decrease again. The reason for this phenomenon is that there are redundant edge cache nodes in the edge network, and at the same time, there are a large number of background traffic packets in the nodes, such as system updates, network monitoring data transmissions, etc. These traffic flows will occupy cache resources. In the edge network environment, such background traffic may frequently replace the valid data in the cache. Even if there are a large number of edge cache nodes, when background traffic packets keep pouring in, the cache nodes will be busy processing these non-critical data, thus squeezing out the data truly used for user requests from the cache. This will cause the neural network training model to deviate, thereby leading to a further decrease in the cache hit rate.

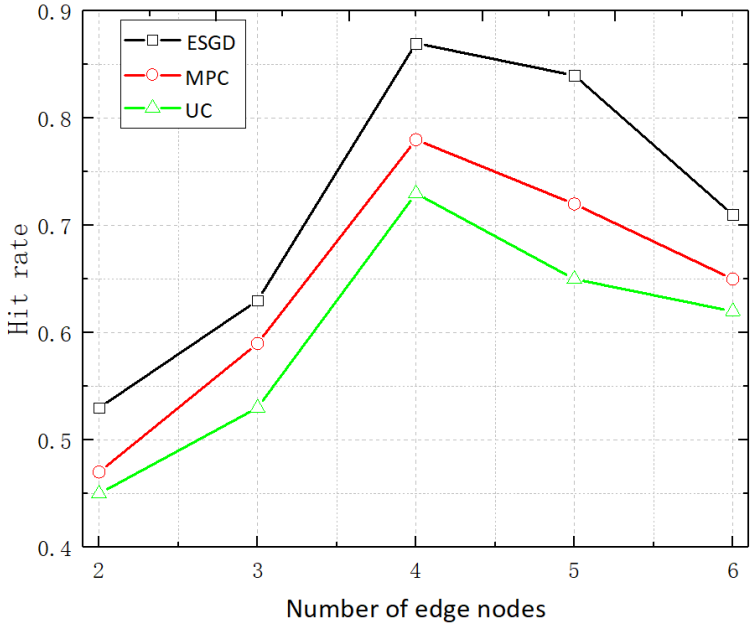

Figure 4. The result of hit rate.

As shown in Figures 3 and 4, both the adequacy rate and the hit rate of the ESGD model are higher than those of the MPC and UC models. The reason for this is that the ESGD scheme takes the optimization relationship between the data cache residence time and the cache space in the multi-queue collaborative caching model as the solution goal, and uses the stochastic gradient descent idea to search for the optimal number of nodes for data caching. Specifically, this model ensures that the data in the cache are those that are likely to be accessed in the near future by reasonably setting the maximum residence time of data in the node. As shown in Table 4, after the edge nodes are optimally deployed, the adequacy rate of the training data required by the ESGD model can reach 87%, and the cache hit rate can reach 100%. This is mainly attributed to the high emphasis of the ESGD model on the balance between cache space and data residence time. The ESGD model adjusts the residence time of data in the cache according to the access frequency and timeliness of the data. For example, for real-time data frequently used in model training, their residence time will be correspondingly extended. Through the optimization of edge node deployment, the cache space of each node can be more reasonably allocated so that this balance reaches an optimal state, thereby ensuring that the key training data will not be replaced prematurely, and thus ensuring the adequacy rate of the training data. In addition, the ESGD uses the stochastic gradient descent algorithm to determine the optimal number of edge nodes. In this process, it comprehensively considers various factors such as network bandwidth, data flow volume, and data update frequency. During the optimization process, it can accurately calculate the required number of nodes based on the actual conditions of the edge network, including network coverage range, user density, etc. In this way, each node can effectively store training data, thereby increasing the cache hit rate.

Table 4. The impact of node quantity on adequate rate and hit rate.

Number of Nodes	Adequate Rate	Hit Rate
2	53%	43%
3	63%	75%
4	87%	100%
5	84%	100%
6	71%	100%

5. Conclusions

This chapter presents an enhanced in-network cache for edge deep learning. A multi-edge-node data cache replacement model based on the queuing theory is established, and the optimal number of data collaborative cache nodes is determined based on this model. Through a comparison with other cache mechanisms, it can be proven that the ESGD strategy proposed in this study can effectively guarantee the cache performance in the edge network. Moreover, in scenarios such as the industrial Internet, intelligent manufacturing, and intelligent vehicle management, the number of node services to be deployed in the relevant edge network environments is often huge. Therefore, when simulating the scenarios, a large amount of actual data is required to simulate the large-scale service deployment scenarios. If experiments and simulations can be conducted using the actual big data and relevant platforms, it will be more conducive to the adjustment and improvement of the existing scheduling schemes and will also be more instructive. However, due to the limitations of simulation, in the case of a maneuvering network, since there are different input variables that will affect the optimal number of edge nodes, deep learning technology will exhibit better performance and achieve better edge node cache performance.

Author Contributions: Conceptualization, J.Z. and W.L.; methodology, L.Z.; validation, J.Z.; resources, W.L.; writing—original draft preparation, J.Z.; writing—review and editing, W.L.; supervision, J.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Science Foundation of China (61962045), Program for Young Talents of Science and Technology in Universities of Inner Mongolia Autonomous Region (NJYT23104), Basic Scientific Research Expenses Program of Universities directly under Inner Mongolia Autonomous Region (JY20220273, JY20240002), and the Natural Science Foundation of Inner Mongolia Autonomous Region (2023LHMS06023, JY20240061).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Hazra, A.; Rana, P.; Adhikari, M.; Amgoth, T. Fog computing for next-generation internet of things: Fundamental, state-of-the-art and research challenges. *Comput. Sci. Rev.* **2023**, *48*, 100549. [CrossRef]
2. Chi, H.R.; Wu, C.K.; Huang, N.-F.; Tsang, K.-F.; Radwan, A. A survey of network automation for industrial internet-of-things towards industry 5.0. *IEEE Trans. Ind. Inform.* **2023**, *19*, 2065–2077. [CrossRef]
3. Cherukuri, B.R. Edge computing vs. cloud computing: A comparative analysis for real-time ai applications. *Int. J. Multidiscip. Res.* **2024**, *6*, 1–17. [CrossRef]
4. Mahbub, M.; Shubair, R.M. Contemporary advances in multi-access edge computing: A survey of fundamentals, architecture, technologies, deployment cases, security, challenges, and directions. *J. Netw. Comput. Appl.* **2023**, *219*, 103726. [CrossRef]
5. Men, J.; Feng, L.; Chen, X.; Tian, L. Atmospheric correction under cloud edge effects for geostationary ocean color imager through deep learning. *ISPRS J. Photogramm. Remote Sens.* **2023**, *201*, 38–53. [CrossRef]
6. Lu, Y.; Lin, Q.; Chi, H.; Chen, J.-Y. Automatic incident detection using edge-cloud collaboration based deep learning scheme for intelligent transportation systems. *Appl. Intell.* **2023**, *53*, 24864–24875. [CrossRef]
7. Liu, G.; Dai, F.; Xu, X.; Fu, X.; Dou, W.; Kumar, N.; Bilal, M. An adaptive dnn inference acceleration framework with end–edge–cloud collaborative computing. *Future Gener. Comput. Syst.* **2023**, *140*, 422–435. [CrossRef]
8. Li, R.; Ouyang, T.; Zeng, L.; Liao, G.; Zhou, Z.; Chen, X. Online optimization of dnn inference network utility in collaborative edge computing. *IEEE/ACM Trans. Netw.* **2024**, *32*, 4414–4426. [CrossRef]
9. Geng, H.; Zeng, D.; Li, Y.; Gu, L.; Chen, Q.; Li, P. Plays: Minimizing dnn inference latency in serverless edge cloud for artificial intelligence-of-things. *IEEE Internet Things J.* **2024**, *11*, 37731–37740. [CrossRef]
10. Sada, A.B.; Khelloufi, A.; Naouri, A.; Ning, H.; Aung, N.; Dhelim, S. Multi-agent deep reinforcement learning-based inference task scheduling and offloading for maximum inference accuracy under time and energy constraints. *Electronics* **2024**, *13*, 2580. [CrossRef]
11. Yang, L.; Gan, Y.; Chen, J.; Cao, J. Autosf: Adaptive distributed model training in dynamic edge computing. *IEEE Trans. Mob. Comput.* **2024**, *23*, 6549–6562. [CrossRef]
12. Hua, H.; Li, Y.; Wang, T.; Dong, N.; Li, W.; Cao, J. Edge computing with artificial intelligence: A machine learning perspective. *ACM Comput. Surv.* **2023**, *55*, 1–35. [CrossRef]

13. Wu, C.; Peng, Q.; Xia, Y.; Jin, Y.; Hu, Z. Towards cost-effective and robust ai microservice deployment in edge computing environments. *Future Gener. Comput. Syst.* **2023**, *141*, 129–142. [CrossRef]
14. Pandey, R.; Uziel, S.; Hutschenreuther, T.; Krug, S. Towards deploying dnn models on edge for predictive maintenance applications. *Electronics* **2023**, *12*, 639. [CrossRef]
15. Heidari, A.; Navimipour, N.J.; Unal, M.; Zhang, G. Machine learning applications in internet-of-drones: Systematic review, recent deployments, and open issues. *ACM Comput. Surv.* **2023**, *55*, 1–45. [CrossRef]
16. Jiang, X.; Hou, P.; Zhu, H.; Li, B.; Wang, Z.; Ding, H. Dynamic and intelligent edge server placement based on deep reinforcement learning in mobile edge computing. *Ad Hoc Netw.* **2023**, *145*, 103172. [CrossRef]
17. Abdulazeez, D.H.; Askar, S.K. Offloading mechanisms based on reinforcement learning and deep learning algorithms in the fog computing environment. *IEEE Access* **2023**, *11*, 12555–12586. [CrossRef]
18. Yu, H.; Yu, D.; Wang, C.; Hu, Y.; Li, Y. Edge intelligence-driven digital twin of cnc system: Architecture and deployment. *Robot. Comput.-Integr. Manuf.* **2023**, *79*, 102418. [CrossRef]
19. Nie, Q.; Tang, D.; Liu, C.; Wang, L.; Song, J. A multi-agent and cloud-edge orchestration framework of digital twin for distributed production control. *Robot. Comput.-Integr. Manuf.* **2023**, *82*, 102543. [CrossRef]
20. Liu, C.; Liu, K.; Ren, H.; Xu, X.; Xie, R.; Cao, J. Rtds: Real-time distributed strategy for multi-period task offloading in vehicular edge computing environment. *Neural Comput. Appl.* **2023**, *35*, 12373–12387. [CrossRef]
21. Raeisi-Varzaneh, M.; Dakkak, O.; Habbal, A.; Kim, B.-S. Resource scheduling in edge computing: Architecture, taxonomy, open issues and future research directions. *IEEE Access* **2023**, *11*, 25329–25350. [CrossRef]
22. Aghapour, Z.; Sharifian, S.; Taheri, H. Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed ai execution tasks in iot edge computing environments. *Comput. Netw.* **2023**, *223*, 109577. [CrossRef]
23. Liang, Q.; Hanafy, W.A.; Ali-Eldin, A.; Shenoy, P. Model-driven cluster resource management for ai workloads in edge clouds. *ACM Trans. Auton. Adapt. Syst.* **2023**, *18*, 1–26. [CrossRef]
24. Wang, Z.; Xu, H.; Xu, Y.; Jiang, Z.; Liu, J. Coopfl: Accelerating federated learning with dnn partitioning and offloading in heterogeneous edge computing. *Comput. Netw.* **2023**, *220*, 109490. [CrossRef]
25. Shu, Z.; Deng, X.; Wang, L.; Gui, J.; Wan, S.; Zhang, H.; Min, G. Relay-assisted edge computing framework for dynamic resource allocation and multiple-access tasks processing in digital divide regions. *IEEE Internet Things J.* **2024**, *11*, 35724–35741. [CrossRef]
26. Nugroho, A.K.; Shioda, S.; Kim, T. Optimal resource provisioning and task offloading for network-aware and federated edge computing. *Sensors* **2023**, *23*, 9200. [CrossRef]
27. Lin, Z.; Zhu, G.; Deng, Y.; Chen, X.; Gao, Y.; Huang, K.; Fang, Y. Efficient parallel split learning over resource-constrained wireless edge networks. *IEEE Trans. Mob. Comput.* **2024**, *23*, 9224–9239. [CrossRef]
28. Pu, Y.; Li, Z.; Yu, J.; Lu, L.; Guo, B. An elastic framework construction method based on task migration in edge computing. *Softw. Pract. Exp.* **2024**, *54*, 1811–1830. [CrossRef]
29. Chen, Y.; Luo, X.; Liang, P.; Han, J.; Xu, Z. Priority-based dag task offloading and secondary resource allocation in iot edge computing environments. *Computing* **2024**, *106*, 3229–3254. [CrossRef]
30. Liao, L.; Lai, Y.; Yang, F.; Zeng, W. Online computation offloading with double reinforcement learning algorithm in mobile edge computing. *J. Parallel Distrib. Comput.* **2023**, *171*, 28–39. [CrossRef]
31. Huang, B.; Wang, L.; Liu, X.; Huang, Z.; Yin, Y.; Zhu, F.; Wang, S.; Deng, S. Reinforcement learning based online scheduling of multiple workflows in edge environment. *IEEE Trans. Netw. Serv. Manag.* **2024**, *21*, 5691–5706. [CrossRef]
32. Wang, X.; Shen, M.; Yang, K. On-edge high-throughput collaborative inference for real-time video analytics. *IEEE Internet Things J.* **2024**, *11*, 33097–33109. [CrossRef]
33. Albonesi, D.H. Selective cache ways: On-demand cache resource allocation. In Proceedings of the MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture, Haifa, Israel, 16–18 November 1999; pp. 248–259.
34. Suh, G.E.; Devadas, S.; Rudolph, L. Analytical cache models with applications to cache partitioning. In Proceedings of the ACM International Conference on Supercomputing 25th Anniversary Volume, Munich, Germany, 10–13 June 2014; pp. 323–334.
35. Kennedy, R.K.; Khoshgoftaar, T.M.; Villanustre, F.; Humphrey, T. A parallel and distributed stochastic gradient descent implementation using commodity clusters. *J. Big Data* **2019**, *6*, 16. [CrossRef]
36. Bottou, L. Large-scale machine learning with stochastic gradient descent. In Proceedings of the COMPSTAT'2010: 19th International Conference on Computational Statistics, Paris, France, 22–27 August 2010; Keynote, Invited and Contributed Papers; Springer: Berlin/Heidelberg, Germany, 2010; pp. 177–186.
37. Chih-Chung, C. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* **2011**, *2*, 1–27. [CrossRef]
38. Liu, X.; Xu, Z.; Qin, Y.; Tian, J. A discrete-event-based simulator for distributed deep learning. In Proceedings of the 2022 IEEE Symposium on Computers and Communications (ISCC), Rhodes, Greece, 30 June–3 July 2022; pp. 1–7.
39. Liu, S.; Liu, Z.; Xu, Z.; Liu, W.; Tian, J. Hierarchical decentralized federated learning framework with adaptive clustering: Bloom-filter-based companions choice for learning non-iid data in iov. *Electronics* **2023**, *12*, 3811. [CrossRef]

40. Liu, X.; Dong, Z.; Xu, Z.; Liu, S.; Tian, J. Enhanced decentralized federated learning based on consensus in connected vehicles. *arXiv* **2022**, arXiv:2209.10722.
41. Lang, K. Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*; Prieditis, A., Russell, S., Eds.; Morgan Kaufmann: San Francisco, CA, USA, 1995; pp. 331–339.