Journal of Computational Systems and Applications

https://ojs.gospub.com/index.php/jcsa

Global Open Share Publishing

Article

Real-time Processing Optimization of Convolutional Neural Network in Edge Computing System

Shean Lai*

Henan University of Animal Husbandry and Economy, Zhengzhou, China

*Corresponding author: Shean Lai, laishean@hnuahe.edu.cn

Abstract

On the edge nodes of Azure Stack Edge, network interfaces often experience packet queuing during peak periods due to congestion and high load. To address this issue, MobileNet was used to analyze network traffic in real-time and detect congestion, aiming to balance node network load and reduce latency. Firstly, data was collected and key features were filtered during peak periods using Wireshark, including packet rate, bidirectional packet count, flag statistics, etc.; next, TensorFlow Lite was used to prune and lightweight the model, removing redundant network layers and reducing floating-point operation accuracy, making it suitable for edge devices; then, based on the optimized model, real-time classification of network traffic was performed using Apache Kafka to detect potential congestion situations; at the same time, the load balancing strategy was dynamically adjusted based on the detection results to optimize queue priority and traffic transfer, thereby alleviating congestion. The results showed that during peak hours, the average processing time for ICMP (Internet Control Message Protocol), TCP (Transmission Control Protocol), and UDP (User Datagram Protocol) traffic was reduced by 27.1%, 23.1%, and 20.7%, respectively. When the cache capacity was 16GB, the hit rate reached 94%. Under standard traffic conditions during peak hours, the packet loss rate dropped to 2.1%, which could significantly improve the performance of edge node networks.

Keywords

Edge computing, Convolutional neural networks, Real-time processing, Network traffic analysis, Apache Kafka

Article history

Received: 30 September 2024

Revised: 28 November 2024

Accepted: 12 December 2024

Available Online: 16 December 2024

1. Introduction

As a new technology, edge computing is gradually replacing the traditional cloud computing model. By transferring computing power from data centers to edge devices closer to data sources, this method aims to reduce data transmission latency and improve response speed [1-2]. However, this transformation has also brought about some new problems [3-4]: in high load network environments, the network interfaces of edge nodes are prone to congestion, leading to packet queuing during transmission and significantly increasing latency. In order to ensure the efficient and stable operation of the system, how to optimize network traffic management, reduce packet latency and improve real-time processing capability has become a research hotspot in the current edge computing field.

This article aims to improve the real-time data processing capability of edge computing systems in the face of high load. The Wireshark is used to collect network traffic data during peak periods and extract multiple features closely related to network performance, including packet size, arrival time interval, packet rate, etc. By pruning to reduce model complexity and using quantization aware training to adapt to low precision operations, the MobileNet model is optimized. To ensure efficient operation of the model on edge devices, GPU (Graphics Processing Unit) acceleration is used for computationally intensive tasks, and memory usage is optimized to reduce latency and computational bottlenecks. At the same time, a real-time stream processing architecture based on Kafka is designed to support efficient processing of network traffic data, and input into the optimized MobileNet model for real-time classification and congestion detection.

The experimental results show that the improved MobileNet model maintains high classification accuracy under different network loads, and the optimization scheme improves packet processing speed, reduces latency and queue length, and enhances network throughput. The applied method effectively improves the real-time processing capability and network congestion response capability of edge computing system. The main contribution of the research lies in that: 1) the MobileNet model is optimized and deployed on Azure Stack Edge, significantly improving the processing speed and accuracy of the model; 2) the Kafka is applied for real-time stream processing and combined with dynamic load balancing strategies to ensure priority processing of critical data streams, improving system responsiveness and resource utilization; 3) the model is continuously retrained and optimized to ensure that the system can adapt to constantly changing network conditions and maintain long-term stable high-performance operation.

2. Related Work

Various strategies have been proposed to address the issue of network congestion. Bi Jing [5] et al. combined SG (Savitzky-Golay) filters, TCN (Temporal Convolutional Network), and LSTM (Long Short-Term Memory) to achieve superior traffic prediction performance. Zhang Yongnan [6] et al. successfully alleviated the network congestion problem in big data scenarios by using deep convolutional neural networks on Spark Cloud for dynamic prediction of network flows. To achieve high-precision sudden traffic prediction, Yu Ao [7] et al. proposed a Spiking Neural Network (SNN) framework based on error feedback, which had good prediction performance while maintaining low complexity. For 5G low latency communication, Hu Long [8] et al. proposed an intelligent traffic prediction and cognitive caching method for F-RANs (Fog-computing-based Radio Access Networks); Serdaroglu Kemal Cagri [9] et al. proposed a network traffic scheduling strategy based on queue priority. However, these methods cannot effectively cope with sudden surges in network traffic under high load conditions [10-11]. Yang Zhong [12] et al. applied a Long Short-Term Memory (LSTM) traffic prediction model based on skip-gram embedding and optimized network management with caching strategy. However, it still faces dual challenges of real-time and accuracy in practical applications [13-14]. Although existing research provides multiple solutions, it is still difficult to handle real-time data streams and adapt to dynamic load changes.

At present, the application of Convolutional Neural Network (CNN) in network traffic analysis has been preliminarily explored. Using network stream data extracted from packet capture files for training, Wong Mei Ling [15] et al. proposed an anomaly detection framework that combined CNN and LSTM to process large-scale stream data in real-time with low latency. Wei Guanglu [16] proposed a hierarchical spatiotemporal feature learning method that combined CNN and RNN to address the issues of strong feature dependency and high false alarm rate in network anomaly detection, achieving an accuracy of 99.69% on the ISCX2012 dataset. Meanwhile, significant progress has been made in the application of CNN models in the field of network security: in 2021, Singh Kuljeet [17] et al. successfully classified network traffic into normal data and attack data using the 1D-CNN model, achieving precise attack detection; in 2023, Anitha T [18] and others realized malicious communication recognition through the BI-LSTM-CNN (Bidirectional Long Short-Term Memory-CNN) model and supported voice input to detect and classify malicious Internet traffic, with a detection rate of 99.62%; extracting features from IoT (Internet of Things) network traffic, Kumar Rakesh [19] et al. studied the importance of network traffic classification, emphasizing that accurate classification was particularly important for ensuring the normal operation of devices and detecting malicious activities; in 2024, Ullah Farhan [20] et al. proposed an Intrusion Detection System using Transformer-based Transfer Learning for Imbalanced Network Traffic (IDS-INT), aimed at handling imbalanced network traffic and achieving classification detection of different types of attacks through deep features. However, the existing research mostly focuses on traffic classification [21,22] rather than real-time congestion detection [23] and dynamic adjustment strategy [24], and the

method is still insufficient in dealing with peak load and dynamic changes in actual edge computing systems [25,26]. MobileNet is a lightweight CNN architecture [27,28] designed specifically for mobile device environments, providing efficient inference capabilities under limited computing resources [29,30]. This study attempted to use MobileNet for real-time analysis of network traffic at edge nodes, by detecting congestion and dynamically adjusting resource allocation, with the aim of balancing node network load and reducing latency.

In edge computing systems, Convolutional Neural Networks (CNNs) have become an effective tool for spatiotemporal modeling, widely applied in real-time network traffic processing and load prediction. Previous studies have shown significant achievements in spatiotemporal data modeling with CNNs, such as wildfire prediction and microfluidic device state forecasting. Cheng et al. proposed a wildfire prediction method combining forward and reverse modeling, improving prediction accuracy through machine learning [31]. Zhuang et al. applied integrated latent assimilation techniques with deep learning models for droplet interaction prediction in microfluidic devices [32]. Chen et al. used graph neural networks to build a global wildfire prediction model, enhancing understanding of complex environmental changes [33]. This study, based on an optimized MobileNet model, performs real-time classification and prediction of network traffic through spatiotemporal modeling, alleviating congestion and optimizing edge device performance, with a similar technical framework to the above research.

Quality of Service (QoS) and Quality of Experience (QoE) are key factors in ensuring network performance and user satisfaction. Laghari et al. reviewed the progress of QoE research in cloud computing, emphasizing its role in service quality optimization, especially in applications such as video streaming and real-time data transmission [34]. Yamazaki pointed out that QoE research is moving toward intelligent network optimization, particularly in 5G and next-generation networks [35]. Agarwal et al. discussed how QoE-driven optimization can enhance video service quality in 5G O-RAN environments, aligning with the goal of optimizing network services through real-time traffic classification and load balancing strategies in this study [36]. Gelenbe et al. proposed the concept of self-aware networks, which can dynamically optimize security, QoS, and energy efficiency, offering new approaches for QoS optimization in edge computing [37]. Putra et al. studied QoS assessment and user QoE perception in different regions, providing recommendations for improving network service quality [38].

This study optimizes the MobileNet model for real-time classification and monitoring of network traffic, reducing latency and congestion, and adjusting load balancing strategies according to QoS requirements, thus improving overall network performance and user experience.

3. Methods

3.1 Data Preprocessing and Network Traffic Feature Extraction

On edge nodes of the Azure Stack Edge, the Wireshark is used to collect network traffic data during peak network periods. The data covers different time periods and traffic types, capturing features closely related to network performance, including packet size, packet arrival time interval, packet rate, protocol type, flow duration, total number of packets in the flow, total number of bytes in the flow, number of bidirectional data packets, and flag statistics. After data collection is completed, custom scripts are used to perform preliminary screening on the raw data, removing invalid and incomplete records. Subsequently, outliers are interpolated to improve data consistency. In the study, outliers are defined as data points that are 3 times the standard deviation away from the mean of the dataset.

For key traffic feature extraction, traffic fluctuations and data transmission patterns are identified by calculating the size of each data packet (in bytes) and recording the packet size statistics of the flow (including maximum, minimum, average, and standard deviation). The arrival time interval of packets reflects the time interval between consecutive data packets, and the average, maximum, minimum, and standard deviation are calculated to determine the suddenness and stability of traffic. The packet rate directly indicates the density and pattern changes of traffic by calculating the number of data packets per unit time. The protocol types include TCP, UDP, ICMP, etc., used to analyze the proportion of different protocols in network traffic and their impact on network load. The duration of the flow is determined by calculating the time difference from the first packet to the last packet, providing the total duration of the flow to identify long-term traffic patterns and short-term burst traffic. The total number of packets and total number of bytes respectively represent the density and total amount of data flow, evaluating the overall load of traffic and its occupation of network resources. The number of bidirectional data packets reflects the communication direction and data exchange characteristics of traffic by counting the number of upstream and downstream packets between the client and server in the flow. The occurrence of flags is counted, including SYN (Synchronize Sequence Numbers), ACK (Acknowledgment), FIN (Finish), and RST (Reset). The statistical display of various features of the collected data stream within 1 minute is shown in Figure 1. Due to the significant differences in numerical values under each indicator, the data is subjected to logarithmic transformation processing.

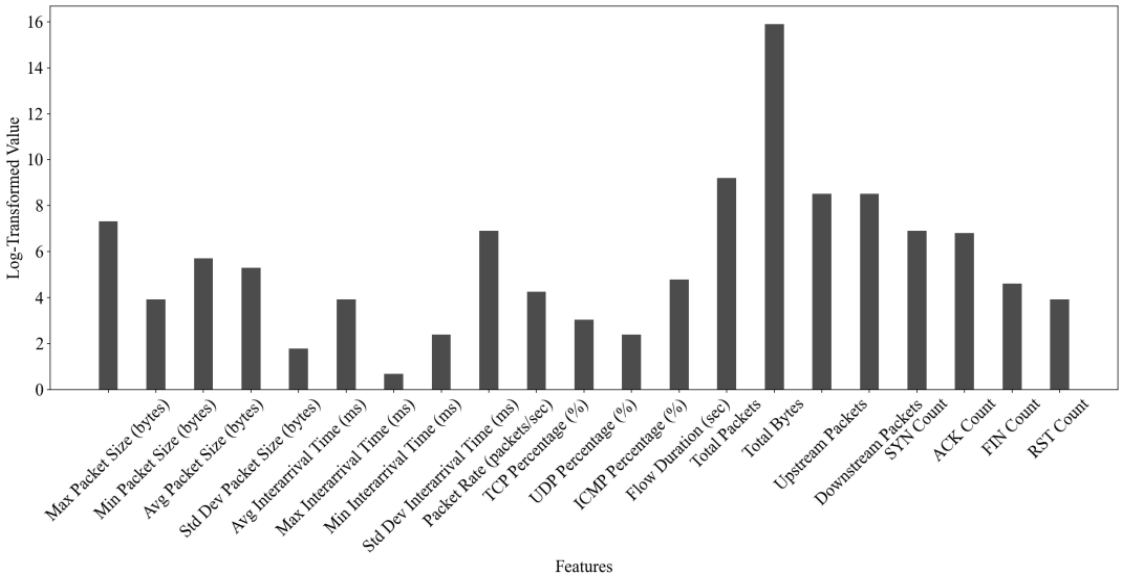

Figure 1. Statistical display of features within 1 minute of data flow

Data standardization techniques are applied to adjust feature data to a distribution with zero mean and one variance, in order to eliminate dimensional differences between features. By conducting correlation analysis and Principal Component Analysis (PCA), the correlation between features is evaluated and features that have a significant impact on network performance are selected, as shown in Figure 2.

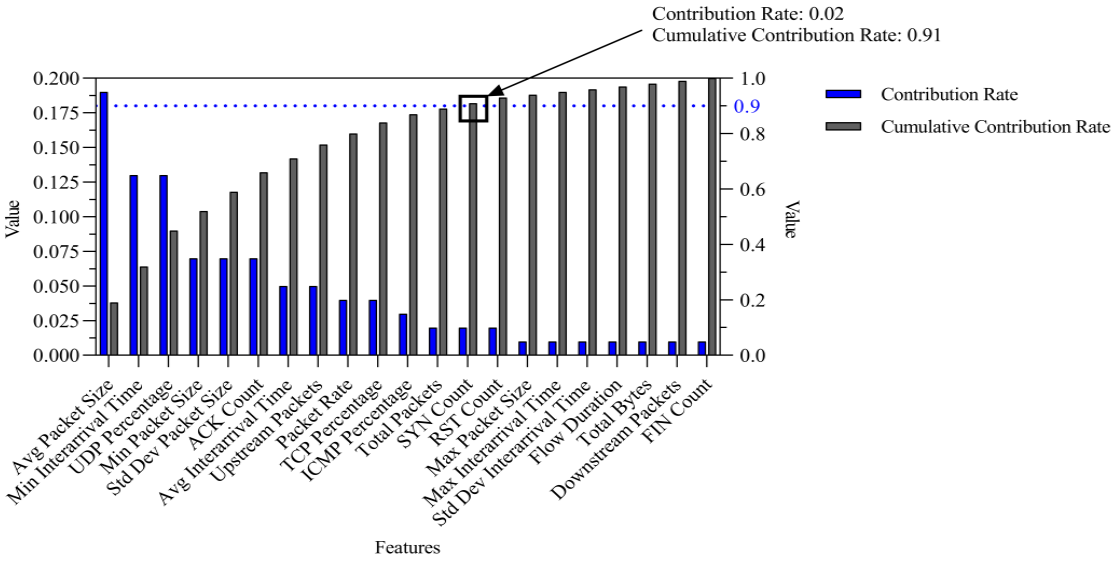

Figure 2. Calculation of contribution rates and cumulative contributions of each feature

In Figure 2, the horizontal axis represents various features; the left vertical axis represents contribution rate, and the right vertical axis represents cumulative contribution rate. It can be seen that after sorting the contribution rates of each feature from high to low, the first 13 out of 21 features can achieve a cumulative contribution rate of 0.91, with the 13th feature SYN Count having a single contribution rate of 0.02. Therefore, in this study, the first 13 features are selected as inputs for the subsequent model (Avg Packet Size, Min Interarrival Time, UDP Percentage, Min Packet Size, Std Dev Packet Size, ACK Count, Avg Interarrival Time, Upstream Packets, Packet Rate, TCP Percentage, ICMP Percentage, Total Packets, SYN Count).

3.2 Lightweight Deployment of MobileNet Model

The MobileNet model is optimized by calculating the L1 norm of each convolution kernel weight to evaluate its importance. For each convolution kernel in the convolutional layer, its weight matrix is $W \in \mathbb{R}^{H \times W \times C_{in} \times C_{out}}$. Among them, H and W are the height and width of the convolution kernel, and C_{in} and C_{out} are the number of input and output channels, respectively. The importance of weights is measured by calculating their L1 norm:

$$\left\|\mathbf{W}_{i,j}\right\|_1 = \sum_{k=1}^{c_{\text{in}}} \sum_{l=1}^{c_{\text{out}}} \left|W_{i,j,k,l}\right| \tag{1}$$

Here, $W_{i,j,k,l}$ represents the weights of the i -th row, j -th column, k -th channel input, and l -th channel output in the weight matrix. Convolution kernels with smaller L1 norm are chosen for pruning to reduce model complexity.

The percentage of convolution kernels that need to be removed is determined by the pruning ratio, as shown in Table 1. Without pruning, the initial model loss is 0.023; the model accuracy reaches 0.963; the number of parameters is 5183654.

Table 1. Model performance under different pruning ratios

Pruning Ratio	Post-Pruning L1 Norm Threshold	Model Loss Increase (Relative to Unpruned)	Accuracy Decrease (%)	Parameter Reduction (%)
20%	0.0015	0.012	1.2%	22%
30%	0.0012	0.024	2.4%	34%
40%	0.0007	0.033	3.1%	45%
50%	0.0005	0.048	4.5%	55%
60%	0.0003	0.053	5.8%	64%

Table 1 shows the performance changes of the MobileNet model under different pruning ratios, including the L1 norm threshold after pruning, model loss increase (relative to unpruning), percentage in accuracy decrease, and percentage in parameter reduction. As the pruning ratio increases, the L1 norm threshold of the model gradually decreases, while the model loss and accuracy decrease show an upward trend. At the same time, the reduction in parameter quantity is also more significant. At a pruning ratio of 20%, the model loss increase is 0.012; the accuracy decrease is 1.2%; the parameter reduction is 22%. When the pruning ratio is increased to 50%, although the parameter reduction is 55%, the model loss increase reaches 0.048 and the accuracy decrease reaches 4.5%. Considering the balance between model performance and parameter quantity, 40% is chosen as the pruning ratio in the study. In this proportion, although the model loss increase is 0.033 and the accuracy decrease is 3.1%, the parameter reduction is 45%, realizing the lightweight of the model to ensure efficient deployment in edge computing systems.

After pruning, the network structure is initialized so that the pruned convolution kernels are replaced with new weight matrices. After pruning, the model is retrained. The form of the loss function is:

$$\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_{\text{CE}}(y_i, \hat{y}_i) + \lambda\|\mathbf{W}\|_1 \tag{2}$$

$\mathcal{L}_{\text{CE}}$ is the cross-entropy loss function; y_i and $\hat{y}_i$ are the true and predicted labels, respectively; $\lambda\|\mathbf{W}\|_1$ is the L1 regularization term used to further reduce model redundancy. The Quantization-Aware Training (QAT) is adopted to optimize model performance. During the training phase, the model is adapted to low precision operations by simulating the quantization effect, and the quantization error is incorporated into the loss function to guide the model to adjust weights and activation values to reduce quantization errors. The training process is shown in Figure 3.

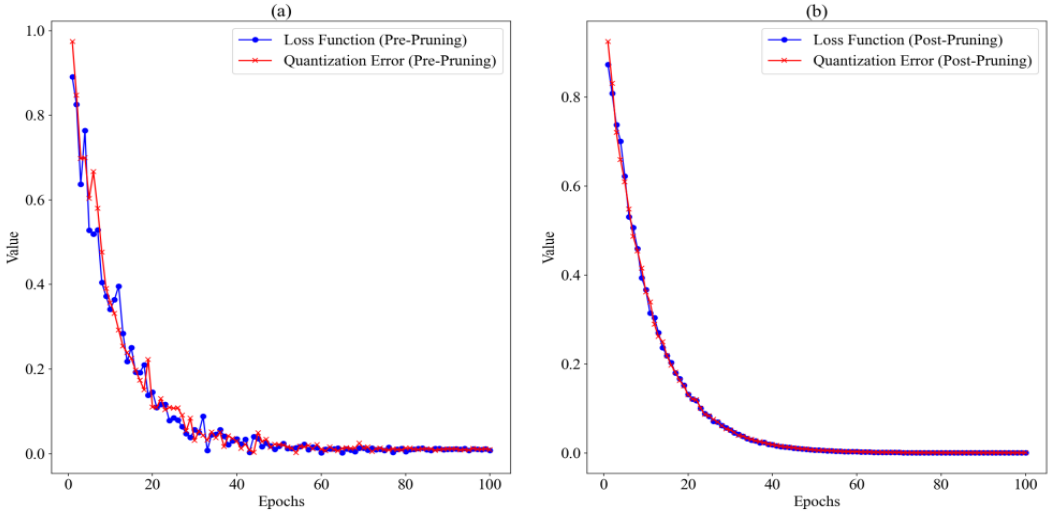

Figure 3. Model training process, Figure 3 (a): Before pruning, Figure 3 (b): After pruning.

Figure 3 (a) shows the training process of the model before pruning. The horizontal axis represents training epochs, and the vertical axis represents loss function values (blue dots) and quantization error values (red crosses). Both have high initial values, but both significantly decrease with increasing rounds. The data before pruning shows significant fluctuations, and although the noise gradually decreases with increasing training epochs, the process remains unstable. Figure 3 (b) shows the model training process after pruning, with the same horizontal and vertical axes as Figure 3 (a).

After pruning, the initial values of the loss function and quantization error are low and rapidly decrease, eventually stabilizing. It can be seen that pruning operations can effectively reduce fluctuations during the model training process, thereby improving training efficiency.

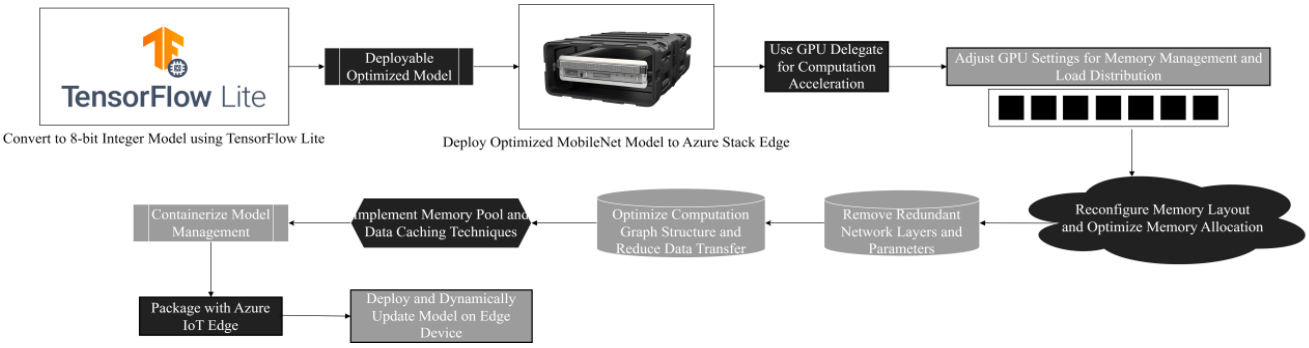

Figure 4. Model optimization and deployment process

The overall process of optimizing and deploying the model is shown in Figure 4. After completing the training, the TensorFlow Lite tool is used to convert the model into an optimization file suitable for edge devices. During the conversion process, the weights and activation values of the floating-point model are converted to 8-bit integers, generating a model file that can be directly deployed to edge devices.

After pruning the model, the memory layout is readjusted to ensure optimal memory usage. By removing redundant network layers and parameters, memory allocation and data caching strategies are optimized to reduce memory consumption. The structure of the computational graph is optimized to reduce unnecessary data transmission and intermediate calculation result storage. Memory pooling technology and data caching mechanism are adopted to further reduce memory access latency and data processing overhead. Finally, the optimized MobileNet model is containerized for management. Using Azure IoT Edge, models are packaged as container images, deployed on edge devices, and dynamically updated. This framework supports hot updates of models at runtime to reduce downtime.

3.3 Real-time Network Traffic Classification and Congestion Detection

The overall architecture of the classification and congestion detection process for real-time network traffic in the research is shown in Figure 5.

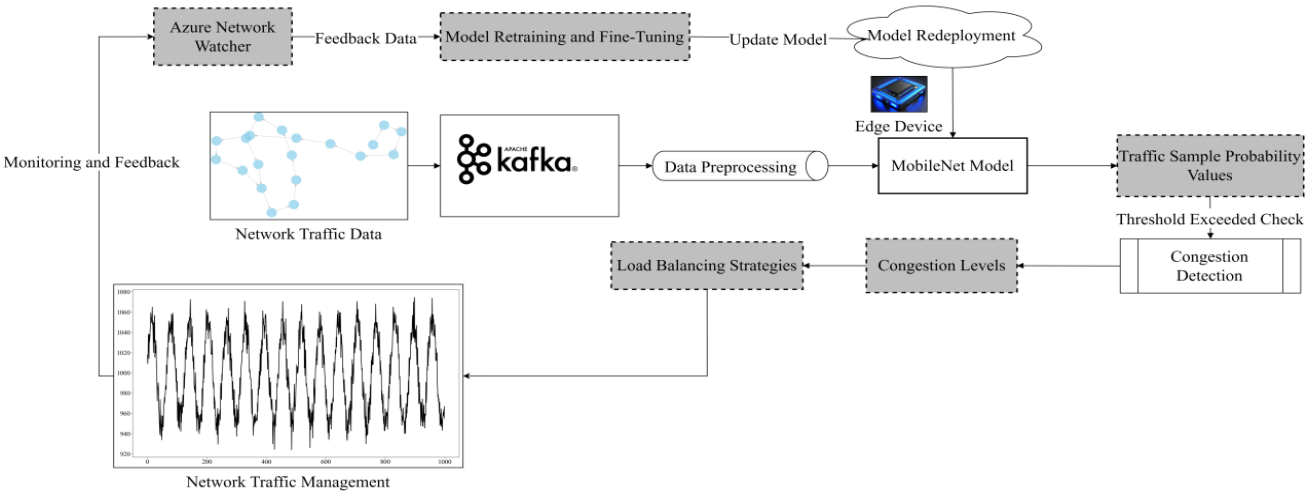

Figure 5. Network traffic classification and congestion detection architecture

Kafka is used as a relay station for real-time stream processing of network traffic data, ensuring efficient data processing based on its high throughput, scalability, and low latency characteristics. Network traffic data is published in the form of messages to Kafka topics for processing by Kafka consumers. After receiving the data, consumers perform preprocessing, including data cleaning and feature extraction, to input the data into the optimized MobileNet model.

Forward inference in the MobileNet model is performed to obtain the probability values of each traffic sample belonging to different categories. For congestion detection, the probability value output by the model is used to determine whether there is congestion in the network. Based on historical data and network traffic patterns, when the probability of detecting network traffic exceeds a preset threshold, it is judged as network congestion. The congestion level is further divided into “normal”, “mild congestion”, and “severe congestion” in the study.

To ensure the effectiveness of load balancing strategies, Azure Network Watcher is used for monitoring network traffic

and node performance. Monitoring data is used to provide feedback on the execution effectiveness of load balancing strategies and provide optimization basis. The system regularly retrains and fine-tunes the MobileNet model to improve its ability to recognize new network traffic and congestion patterns. The latest network traffic data and congestion labels are collected from the system, and these data are used to train the MobileNet model, update model weights, and redeploy the updated model to edge devices. Continuous model training and optimization can further enhance the network performance of edge nodes, ensuring that the system can adapt to constantly changing network conditions.

3.4 Dynamic Adjustment of Load Balancing Strategy

Based on the real-time classification results of network traffic using the MobileNet model, different load balancing strategies are applied, as shown in Table 2.

Table 2. Summary of Load Balancing Strategies

Strategy	Description	Objective
Queue Priority Adjustment [39]	Adjust the queue priority when mild congestion is detected based on real-time network traffic classification, ensuring that critical traffic is processed first.	Alleviate mild congestion, maximize processing rate for critical traffic, and reduce delay for lower-priority queues.
Traffic Path Adjustment [40]	Adjust the traffic path priority using Azure Traffic Manager to ensure high-priority traffic is processed first.	Prioritize critical traffic and reduce the impact of congestion on important business.
Queue Processing Capacity Enhancement [41]	Increase the processing capacity of high-priority queues, reducing queue length and optimizing network resource usage.	Improve the processing capability of high-priority queues during peak traffic, reducing latency for high-priority traffic.
Traffic Migration [42]	Use Azure Load Balancer to redirect some of the traffic to backup nodes to avoid overloading the primary node.	Balance the load, relieve the pressure on the primary node, and ensure backup nodes can handle some traffic.
Bandwidth Throttling [43]	Implement bandwidth throttling for specific traffic, adjusting bandwidth allocation when usage exceeds the threshold, to prioritize critical business traffic.	Control network congestion, prioritize bandwidth for critical business traffic, and avoid excessive bandwidth consumption.

When detecting mild network congestion, the congestion is alleviated by adjusting queue priority. At this point, Azure Traffic Manager is used to adjust the priority of traffic paths, ensuring that important data traffic is processed first. The priority adjustment of traffic allocation within time period t is expressed as: $\text{Priority}_i(t) = \text{BasePriority}_i + \Delta\text{Priority}_i(t)$.

Among them, BasePriority_i is the benchmark priority for traffic category i, and $\Delta\text{Priority}_i(t)$ is the priority increment adjusted according to the network status within time t. The priority adjustment goal is to maximize the processing speed of important data traffic and reduce the processing latency of low priority queues.

During peak traffic periods, the processing capacity of high priority queues is increased, and the queue length changes are as follows:

$$L_i(t) = \frac{\lambda_i(t)}{\mu_i(t)} \tag{3}$$

$L_i(t)$ is the length of queue i within time t; $\lambda_i(t)$ is the traffic arrival rate; $\mu_i(t)$ is the service rate. Improving queue processing capability means increasing $\mu_i(t)$, which in turn reduces $L_i(t)$. In the face of severe congestion, two measures are adopted: traffic transfer and bandwidth limitation. Traffic transfer redirects a portion of the traffic to a backup node through Azure Load Balancer. The traffic transfer configuration rules are represented by the following allocation model:

$$F_j(t) = F_i(t) \cdot \frac{R_j(t)}{R_i(t)} \tag{4}$$

Among them, $F_j(t)$ and $F_i(t)$ respectively represent the traffic allocation of the backup node j and the main node i at time t, and $R_j(t)$ and $R_i(t)$ respectively represent the load capacity of the backup node and the main node. Based on the model, the intelligent scheduling of traffic is ensured according to the load capacity of nodes, and normal operations of nodes are maintained through health check mechanisms. To further control severe congestion, bandwidth restrictions are implemented on specific traffic:

$$B_k(t) = B_{\max} - \alpha \cdot \left(C_k(t) - C_{threshold} \right) \tag{5}$$

$B_k(t)$ is the bandwidth limit for a specific traffic k within time t; $B_{\max}$ is the maximum bandwidth; α is the bandwidth adjustment coefficient; $C_k(t)$ is the current bandwidth usage of traffic k; $C_{threshold}$ is the bandwidth threshold. When the

bandwidth usage of traffic k exceeds the threshold, bandwidth limit $B_k(t)$ can be adjusted according to the excess to prioritize ensuring the bandwidth requirements of critical business traffic. The adjustment process for real-time traffic of edge node networks within 150 seconds is shown in Figures 6 (a-f). Among them, circles are used to represent 50 edge nodes, and the color depth of the edges represents the real-time traffic size (normalized to a 0-1 range for ease of display).

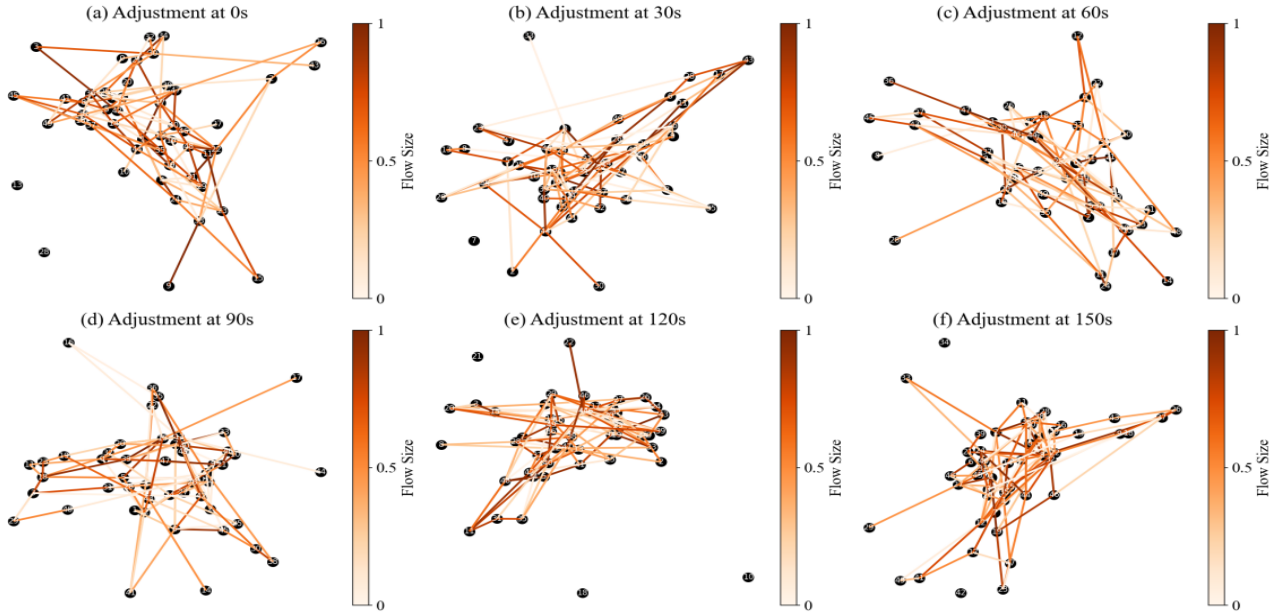

Figure 6. Real-time adjustment of edge node network traffic. (Figure 6 (a): Adjustment at 0 seconds; Figure 6 (b): Adjustment at 30 seconds; Figure 6 (c): Adjustment at 60 seconds; Figure 6 (d): Adjustment at 90 seconds; Figure 6 (e): Adjustment at 120 seconds; Figure 6 (f): Adjustment at 150 seconds).

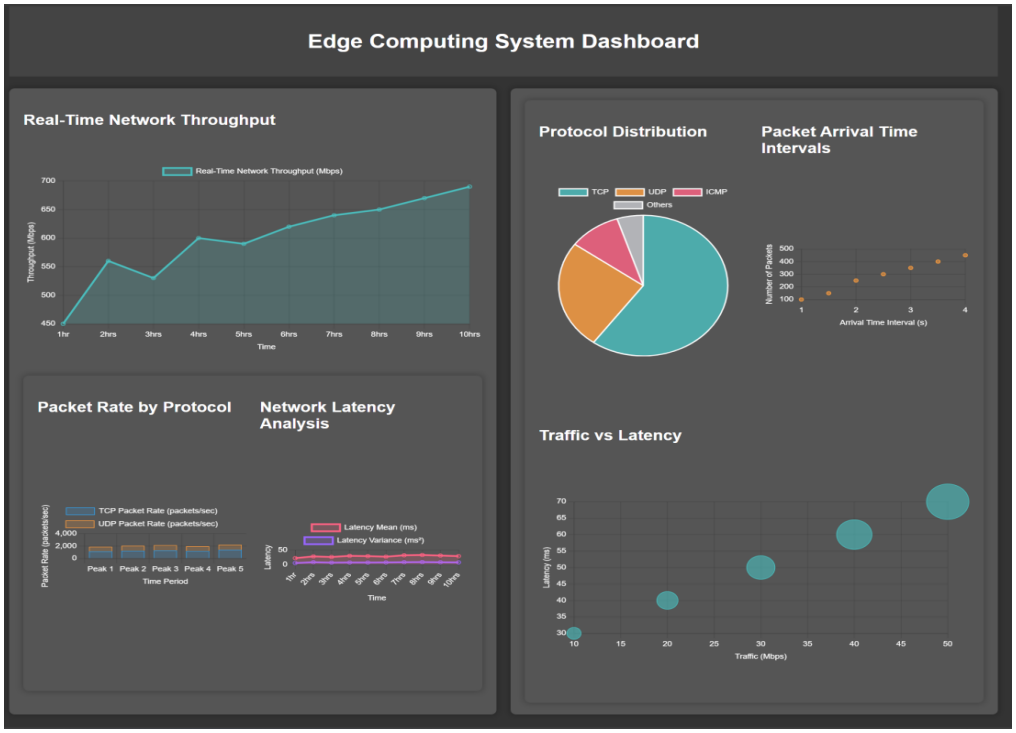

Figure 7. Edge computing system interface display

The system automatically triggers policy adjustments through Azure Functions. Based on event-driven processes, the real-time execution of policies is ensured. When the model detects congestion, an event notification Azure Functions is generated and the corresponding API (Application Programming Interface) is called to implement policy adjustments. The effectiveness of the strategy is monitored in real-time through Azure Network Watcher, and feedback data after load balancing adjustments is collected, including indicators such as network throughput, queue length, latency, and packet loss rate. Feedback data is used to evaluate the effectiveness of strategies and optimize them as necessary. An example of the real-time traffic monitoring and processing optimization interface of the overall edge computing system is shown in Figure 7.

Figure 7 shows the real-time monitoring and performance optimization interface of the edge computing system, which mainly includes several modules: the real-time network throughput module monitors the data transmission speed and stability; the packet rates of different protocols are compared using the packet rate module classified by protocol; the network latency analysis module displays the mean and variance of latency to identify trends and stability in latency variation. The protocol distribution module displays the proportion of various network protocols; the data packet arrival time interval module analyzes the time interval of data packet arrival, and identifies traffic patterns and potential congestion; the traffic and latency module optimizes network performance by analyzing the relationship between traffic and latency.

4. Results and Discussion

The experimental setup in this paper covers multiple aspects: 1) The network traffic classification accuracy experiment aims to assess the robustness of the model by comparing the accuracy of the improved MobileNet model with other lightweight deep learning models during low, medium, and high traffic periods. Data collection will take place over 10 days, and model performance will be compared by calculating the classification accuracy for each traffic type. 2) The latency optimization experiment evaluates the change in latency before and after optimization by measuring the processing time of ICMP, TCP, and UDP traffic. The experiment will be conducted during peak and off-peak periods, with a focus on comparing the reduction in latency before and after optimization. 3) The queue length experiment evaluates the impact of the optimization scheme on queue management by monitoring the variation in queue length across different time periods. Data collection will be conducted during peak and off-peak periods, and the effect will be analyzed by comparing the average queue lengths before and after optimization. 4) The network throughput experiment measures the network throughput during different periods, comparing network transmission efficiency before and after optimization. A 30-second throughput measurement will be taken every 30 minutes during peak periods to capture traffic fluctuations and system response. 5) The cache hit rate experiment evaluates the impact of cache strategy optimization on hit rate by setting different cache capacities. Data will be collected during low, medium, and high peak periods, with tests conducted at cache capacities ranging from 512MB to 16GB. 6) The packet loss rate experiment assesses the impact of the optimization scheme on data transmission reliability by calculating the packet loss rate under different traffic loads. The experiment will be conducted during both peak and off-peak periods, in standard and dense traffic environments. 7) The performance experiment compares the network latency of different edge computing technologies (e.g., Azure Stack Edge, AWS Outposts, Google Anthos) under varying device numbers to evaluate their performance under different load conditions. The experiment will be conducted during low, medium, and high peak periods, with data collected for device numbers ranging from 50 to 450.

4.1 Accuracy of Network Traffic Classification

The improved MobileNet model is compared with models such as SqueezeNet, ShuffleNet, EfficientNet, and NasNet Mobile under different network conditions within 10 days, and the consistency between the model classification results and actual labels is compared. The classification accuracy is evaluated by calculating the proportion of correctly classified traffic to the total traffic, as shown in Figure 8. Low peak value refers to periods of low network load, where network traffic is below 10 Mbps per second and packet arrival frequency is low. The mild peak value describes a situation where the network load is moderate, with network traffic between 10-50 Mbps per second. The high peak value represents the period of the highest network load, with network traffic exceeding 50 Mbps per second.

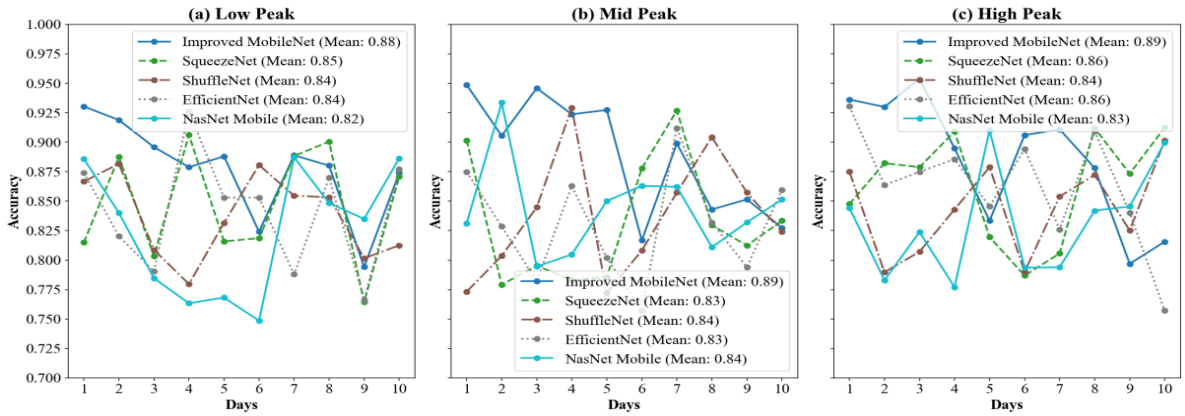

Figure 8. Accuracy of traffic classification. (a): Low peak value; (b): Mid peak value; (c): High peak value.

Figures 8 (a-c) show a comparison of traffic classification accuracy under different network load conditions. Under all testing conditions within 10 days, the average classification accuracy of the improved MobileNet model reaches 0.88 (Low Peak), 0.89 (Mid Peak), and 0.89 (High Peak), respectively, demonstrating good robustness and stability in detecting network congestion. Compared with other lightweight deep learning models such as SqueezeNet, ShuffleNet, EfficientNet, and NasNet Mobile, the improved MobileNet maintains high accuracy under various network load

conditions, and its performance advantage is more obvious under high network load conditions: under high load conditions, the accuracy of SqueezeNet, ShuffleNet, EfficientNet, and NasNet Mobile are 0.86, 0.84, 0.86, and 0.83, respectively, while the improved MobileNet model maintains 0.89, demonstrating superior performance in network traffic classification.

4.2 Latency Reduction

Latency evaluation is conducted by measuring the average time from packet reception to processing completion. Table 3 presents a comparison of packet processing time for ICMP, TCP, and UDP traffic during different time periods before and after optimization.

Table 3. Packet processing time

Period	Traffic Type	Average Processing Time Before Optimization (ms)	Average Processing Time After Optimization (ms)	Latency Reduction (%)
Peak Period	ICMP	85	62	27.1%
Peak Period	TCP	78	60	23.1%
Peak Period	UDP	82	65	20.7%
Off-Peak Period	ICMP	45	33	26.7%
Off-Peak Period	TCP	42	32	23.8%
Off-Peak Period	UDP	48	35	27.1%

During peak hours, the average processing time for ICMP, TCP, and UDP traffic decreases from 85ms, 78ms, and 82ms before optimization to 62ms, 60ms, and 65ms after optimization, respectively. The corresponding latency reduction percentages are 27.1%, 23.1%, and 20.7%, indicating that under high load conditions, the optimized system can significantly improve packet processing speed. During off-peak period, although the network load is low, optimization still has a certain degree of performance improvement. The processing time for ICMP, TCP, and UDP traffic types has been reduced by 26.7%, 23.8%, and 27.1%, respectively, to 33ms, 32ms, and 35ms, proving that the optimization scheme is not limited to high load periods and is effective.

4.3 Queue Length

The queue length monitoring is completed by real-time tracking of the number of queued packets in the queue, as shown in Table 4.

Table 4. Statistics on queue length optimization

Time Slot	Real-Time Monitoring Point	Pre-Optimization Average Queue Length (Packets)	Post-Optimization Average Queue Length (Packets)	Minimum Queue Length (Packets)	Maximum Queue Length (Packets)
Peak - 10:00 AM	Monitoring Point 1	128	84	68	148
Peak - 10:15 AM	Monitoring Point 2	132	87	70	155
Peak - 10:30 AM	Monitoring Point 3	134	90	72	160
Peak - 10:45 AM	Monitoring Point 4	136	92	75	165
Peak - 11:00 AM	Monitoring Point 5	130	89	73	158
Off-Peak - 3:00 PM	Monitoring Point 1	78	56	52	99
Off-Peak - 3:15 PM	Monitoring Point 2	75	53	49	95
Off-Peak - 3:30 PM	Monitoring Point 3	76	54	50	98
Off-Peak - 3:45 PM	Monitoring Point 4	74	52	47	93
Off-Peak - 4:00 PM	Monitoring Point 5	72	50	45	90

Table 4 shows the changes in queue length at the five highest load monitoring points during different time periods before and after optimization. The data shows that during peak periods, the average queue length before optimization is between 128-136 packets, but after optimization it decreases to 84-92 packets, with a minimum queue length of 68-75 packets and a maximum queue length of 148-165 packets. By adopting MobileNet-based traffic detection and corresponding load balancing strategy adjustments, network congestion can be significantly reduced under high load conditions. During off-peak monitoring periods, the queue length fluctuates within the range of 72-78 packets before optimization, but decreases to 50-56 packets after optimization. The minimum queue length is 45-52 packets, and the maximum queue length is 90-99 packets. Even in situations where network usage is low, this optimization still helps improve system response speed and overall performance.

4.4 Network Throughput

The network throughput is measured by calculating the total amount of successfully transmitted data per unit time. During the peak hours of 10:00 am to 2:30 pm, 30 seconds are selected every 30 minutes for testing. Short-term data is collected during high traffic periods to capture traffic fluctuations and system reactions. The results are shown in Figure 9.

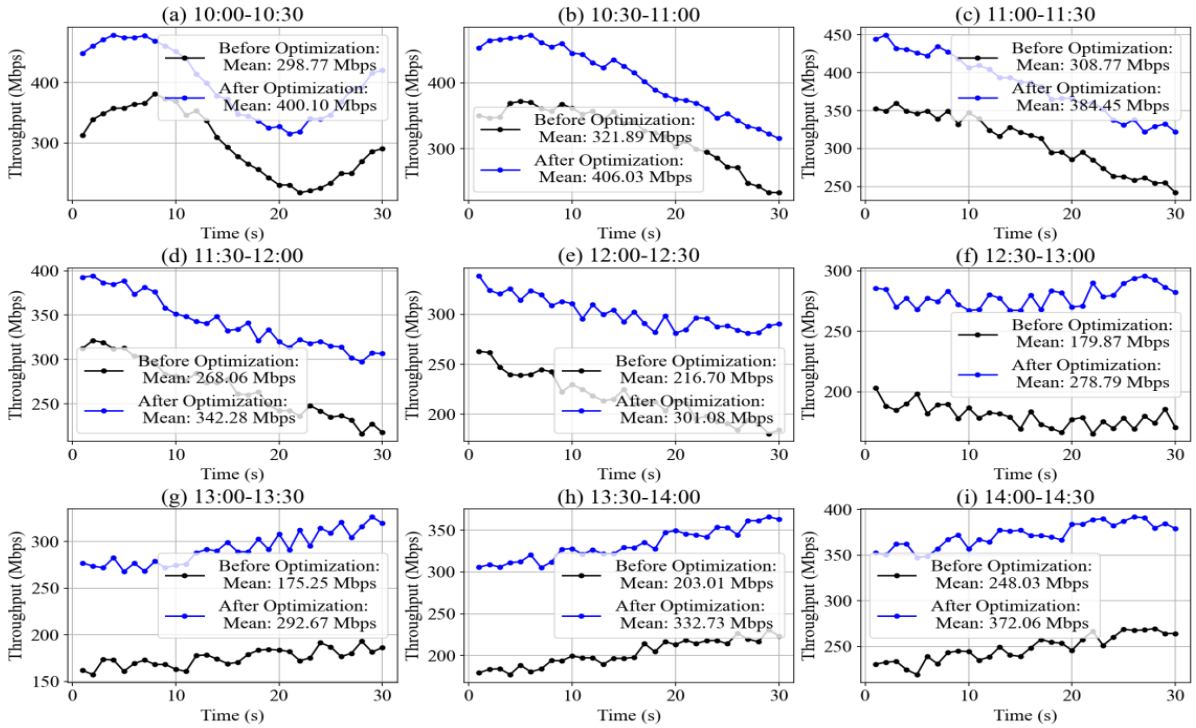

Figure 9. System network throughput. (Figure 9 (a): 10:00-10:30; Figure 9 (b): 10:30-11:00; Figure 9 (c): 11:00-11:30; Figure 9 (d): 11:30-12:00; Figure 9 (e): 12:00-12:30; Figure 9 (f): 12:30 to 13:00; Figure 9 (g): 13:00-13:30; Figure 9 (h): 13:30-14:00; Figure 9 (i): 14:00-14:30)

From Figures 9 (a-i), it can be seen that the optimized network throughput has significantly improved. During the period from 10:00 to 10:30 (Figure 9 (a)), the average throughput before optimization is 298.77Mbps, which increases to 400.1Mbps after optimization; during the period of 10:30 to 11:00 (Figure 9 (b)), the throughput increases from 321.89Mbps to 406.03Mbps; between 11:00 and 11:30 (Figure 9 (c)), the throughput increases from 308.77Mbps to 384.45Mbps. The data shows that under high network load, the optimization scheme significantly improves network transmission efficiency. As the time period progresses to the afternoon, although the overall traffic decreases, optimization measures still play an important role. During the period from 13:30 to 14:00 (Figure 9 (h)), the throughput increases from 203.01Mbps to 332.73Mbps. The optimized system performs better in network throughput than before, which indicates that the applied method is more effective in improving the real-time processing capacity of edge computing system and coping with network congestion.

4.5 Cache Hit Rate

By measuring the ratio of cache hits to total requests, the improvement of cache strategy optimization on data access efficiency is evaluated, and the results are shown in Figure 10.

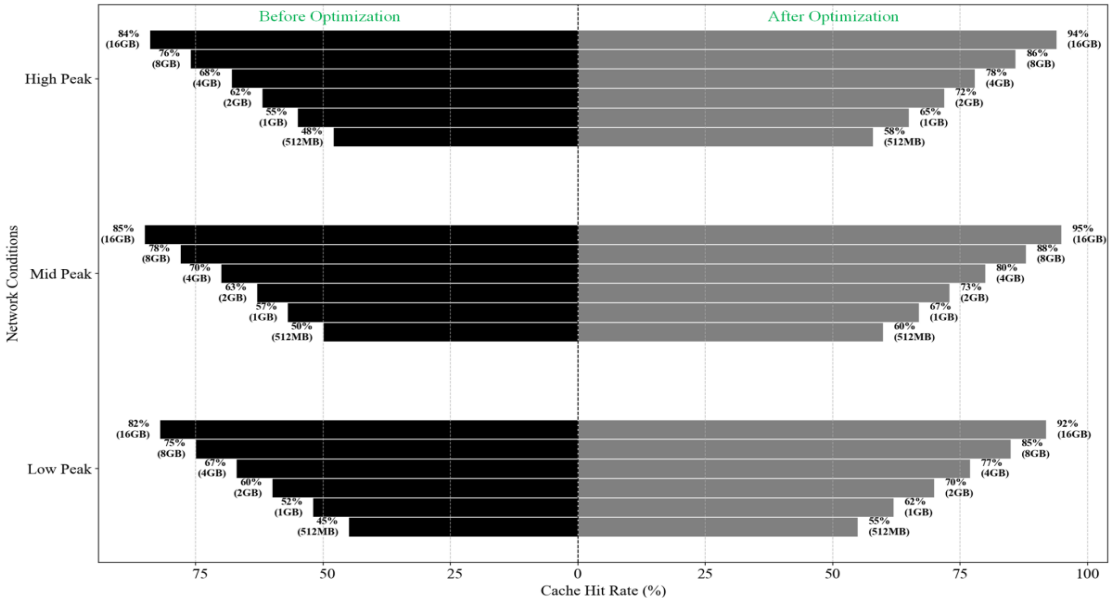

Figure 10. System cache hit rate

Figure 10 shows the changes in cache hit rate before and after optimization under different network load conditions (Low Peak, Mid Peak, High Peak) and different cache size settings. Under all testing conditions, the optimized caching strategy achieves higher cache hit rates. Under a cache size of 512MB, the cache hit rates before optimization are 45%, 50%, and 48%, respectively, and after optimization, they increase to 55%, 60%, and 58%. When the cache size increases to 1GB, the hit rates before and after optimization increase from 52%, 57%, and 55% to 62%, 67%, and 65%, respectively. As the cache capacity further expands to 2GB, 4GB, 8GB, and even 16GB, the trend becomes more apparent. When the cache capacity reaches 16GB, the pre-optimization hit rates are 82%, 85%, and 84%, and the post optimization hit rates are 92%, 95%, and 94%. By combining MobileNet with traffic prediction and optimizing the load balancing strategy, it can significantly improve the efficiency of cache usage and enhance the data access performance of edge computing systems when dealing with high load network traffic.

4.6 Packet Loss Rate

The ratio of the number of data packets lost during data transmission to the total number of transmitted data packets is calculated to evaluate the impact of congestion management and traffic optimization on data transmission reliability. The results are shown in Table 5.

Table 5. System packet loss rate

Test Condition	Packet Loss Rate Before Optimization (%)	Packet Loss Rate After Optimization (%)	Improvement (%)	Traffic Size (Mbps)
Peak, Standard Traffic	5	2.1	2.9	200
Peak, Dense Traffic	6.2	2.5	3.7	400
Off-Peak, Standard Traffic	3.1	1	2.1	100
Off-Peak, Dense Traffic	4	1.8	2.2	150
Mid-Peak, Standard Traffic	2.8	0.9	1.9	120
Mid-Peak, Dense Traffic	3.5	1.3	2.2	250

Under standard traffic conditions during peak hours, the packet loss rate before optimization is 5%, but after optimization it decreases to 2.1%, an increase of 2.9 percentage points; in response to the dense traffic scenario during peak hours, the packet loss rate is reduced from 6.2% to 2.5%, an improvement of 3.7 percentage points. Under off-peak standard traffic conditions, the packet loss rate decreases from 3.1% to 1%, an increase of 2.1 percentage points; under off-peak traffic conditions, the packet loss rate decreases from 4% to 1.8%, an improvement of 2.2 percentage points. In a medium peak standard traffic environment, the packet loss rate decreases from 2.8% to 0.9%, an increase of 1.9 percentage points; for medium peak dense traffic, the packet loss rate decreases from 3.5% to 1.3%, an improvement of 2.2 percentage points. The data shows that the application of MobileNet-based congestion management and traffic optimization technology significantly reduces packet loss, and effectively improves data transmission reliability and the overall performance of edge computing system.

4.7 Performance Under Varying Conditions

The network latency (in milliseconds) of different edge computing technologies (Azure Stack Edge, Amazon AWS Outposts, and Google Anthos) under low, mid, and high peak network conditions, as well as with varying numbers of edge devices, is compared as follows:

Table 6. Network Latency of Edge Computing Technologies under Varying Conditions

Edge Devices	A	B	C	D	E	F	G	H	I
50	3.5	5.3	8.2	5.5	7.3	11	5.7	7.5	11.3
100	3.8	5.8	9.1	6	7.5	12	6.2	8	12.2
150	4.2	6.3	10.2	6.3	8	13	6.5	8.3	13
200	4.6	7	11.3	6.8	8.4	14	7	8.6	14.2
250	5	7.5	12.4	7	8.7	15	7.2	9	15.4
300	5.3	8	13	7.5	9	16	7.8	9.2	16.5
350	5.6	8.4	14.2	7.8	9.4	17	8.1	9.5	17.8
400	6	8.9	15.3	8	9.7	18	8.3	9.8	18.2
450	6.4	9.3	16.4	8.3	10	19	8.6	10.2	19

In Table 6, A-I represent the network latency of different technologies (Azure Stack Edge, AWS Outposts, and Google Anthos) under low, mid, and high peak periods. To evaluate the performance of different edge computing technologies during network peak periods, the study compares the latency performance of Azure Stack Edge, Amazon AWS Outposts, and Google Anthos under varying network loads. As the number of edge devices increased from 50 to 450, the latency of all three technologies rose, but Azure Stack Edge performed the best under the same conditions. With 200 devices, Azure Stack Edge's latency during off-peak hours was 4.6ms, while AWS Outposts and Google Anthos had latencies of 6.8ms and 7ms, respectively. During peak hours, Azure Stack Edge's latency was 11.3ms, whereas AWS Outposts and Google Anthos both reached 14.2ms. The results demonstrate that Azure Stack Edge can effectively maintain lower latency in high-concurrency scenarios, thanks to its use of MobileNet for real-time network traffic analysis and dynamic

load balancing, which optimizes queue priorities and traffic scheduling, alleviates network congestion, and enhances edge node performance.

To ensure that the model can continuously adapt to changing network conditions and maintain long-term high performance, this study implements dynamic adjustment and real-time optimization mechanisms. By using the optimized MobileNet model for real-time classification and congestion detection of network traffic, the system can quickly respond to fluctuations in network load. As the number of edge devices increases, network load and latency also change accordingly. Under varying load conditions, the model continuously learns and adjusts, dynamically updating load balancing strategies to optimize queue priorities and traffic forwarding. This mechanism effectively reduces latency growth and improves network response speed under high-load conditions. During peak network hours, although latency increases, the overall performance shows significant improvement compared to the unoptimized state through intelligent load balancing adjustments. This mechanism ensures that the edge computing system maintains high performance and stability during prolonged operation in complex environments.

5. Conclusions

Aiming at the data processing challenges of edge computing systems under high load conditions, this article applied a real-time traffic analysis and congestion detection method based on optimized MobileNet. By applying this method on Azure Stack Edge nodes, Wireshark was used to collect peak network traffic data and extract key features from it. The MobileNet model was pruned and lightweighted to meet the computing needs of edge devices. Through the real-time stream processing architecture based on Apache Kafka, the model can dynamically detect and alleviate network congestion. The experimental results show that the optimized MobileNet model can maintain high classification accuracy under different network loads, significantly improve processing speed, effectively improve system response speed and resource utilization, and reduce latency and queue length. Although research has achieved certain results, there is still room for improvement. Future work can focus on further optimizing the model to adapt to more network scenarios and exploring more load balancing strategies to cope with more complex network environments. In addition, the continuous retraining and adaptability of the model are also important directions for future research to ensure that the system maintains long-term stable high-performance performance under constantly changing network conditions.

References

[1] Liu GX, Shi H, Kiani A, Khreishah A, Lee J, et al. Smart traffic monitoring system using computer vision and edge computing. IEEE Transactions on Intelligent Transportation Systems. 2022, 23(8), 12027-12038. DOI: 10.1109/TITS.2021.3109481

[2] Carvalho G, Cabral B, Pereira V, Bernardino J. Edge computing: current trends, research challenges and future directions. Computing. 2021, 103(5), 993-1023. DOI: 10.1007/s00607-020-00896-5

[3] Yuan XM, Chen JH, Yang JY, Zhang N, Yang TT, et al. Fedstn: Graph representation driven federated learning for edge computing enabled urban traffic flow prediction. IEEE Transactions on Intelligent Transportation Systems. 2023, 24(8), 8738-8748. DOI: 10.1109/TITS.2022.3157056

[4] Wang Z, Hu J, Min GY, Zhao ZW, Wang J. Data-augmentation-based cellular traffic prediction in edge-computing-enabled smart city. IEEE Transactions on Industrial Informatics. 2021, 17(6), 4179-4187. DOI: 10.1109/TII.2020.3009159

[5] Bi J, Zhang X, Yuan HT, Zhang J, Zhou MC. A hybrid prediction method for realistic network traffic with temporal convolutional network and LSTM. IEEE Transactions on Automation Science and Engineering. 2022, 19(3), 1869-1879. DOI: 10.1109/TASE.2021.3077537

[6] Zhang YN, Zhou YH, Lu HP, Fujita H. Traffic network flow prediction using parallel training for deep convolutional neural networks on spark cloud. IEEE Transactions on Industrial Informatics. 2020, 16(12), 7369-7380. DOI: 10.1109/TII.2020.2976053

[7] Yu A, Yang H, Nguyen KK, Zhang J, Cheriet M. Burst traffic scheduling for hybrid E/O switching DCN: An error feedback spiking neural network approach. IEEE Transactions on Network and Service Management. 2021, 18(1), 882-893. DOI: 10.1109/TNSM.2020.3040907

[8] Hu L, Miao YM, Yang J, Ghoneim A, Hossain MS, et al. If-rans: intelligent traffic prediction and cognitive caching toward fog-computing-based radio access networks. IEEE Wireless Communications. 2020, 27(2), 29-35. DOI: 10.1109/MWC.001.1900368

[9] Serdaroglu KC, Baydere S. An efficient multipriority data packet traffic scheduling approach for fog of things. IEEE Internet of Things Journal. 2022, 9(1), 525-534. DOI: 10.1109/JIOT.2021.3084502

[10] Admassu T, Bizuneh HD. Improving network performance with an integrated priority queue and weighted fair queue scheduling. Indonesian Journal of Electrical Engineering and Computer Science. 2020, 19(1), 241-247. DOI: 10.11591/ijeecs.v19.i1.pp241-247

[11] Samal T, Kabat MR. A prioritized traffic scheduling with load balancing in wireless body area networks. Journal of King Saud University-Computer and Information Sciences. 2022, 34(8), 5448-5455. DOI: 10.1016/j.jksuci.2020.12.023

[12] Yang Z, Liu YW, Chen Y, Zhou JT. Deep learning for latent events forecasting in content caching networks. IEEE Transactions on Wireless Communications. 2022, 21(1), 413-428. DOI: 10.1109/TWC.2021.3096747

[13] Tang B, Kang LY. EICache: A learning-based intelligent caching strategy in mobile edge computing. Peer-to-Peer Networking and Applications. 2022, 15(2), 934-949. DOI: 10.1007/s12083-021-01266-4

[14] Hardegen C, Pfülb B, Rieger S, Gepperth A. Predicting network flow characteristics using deep learning and real-world network traffic. IEEE Transactions on Network and Service Management. 2020, 17(4), 2662-2676. DOI: 10.1109/TNSM.2020.3025131

[15] Wong ML, Arjunan T. Real-Time Detection of Network Traffic Anomalies in Big Data Environments Using Deep Learning

Models. Emerging Trends in Machine Intelligence and Big Data. 2024, 16(2), 1-11.
[16] Wei GL, Wang ZH. Adoption and realization of deep learning in network traffic anomaly detection device design. Soft Computing. 2021, 25(2), 1147-1158. DOI: 10.1007/s00500-020-05210-1
[17] Singh K, Mahajan A, Mansotra V. 1D-CNN based Model for Classification and Analysis of Network Attacks. International Journal of Advanced Computer Science and Applications. 2021, 12(11), 604-613. DOI: 10.14569/IJACSA.2021.0121169
[18] Anitha T, Aanjankumar S, Poonkuntran S, Nayyar A. A novel methodology for malicious traffic detection in smart devices using BI-LSTM–CNN-dependent deep learning methodology. Neural Computing and Applications. 2023, 35(27), 20319-20338. DOI: 10.1007/s00521-023-08818-0
[19] Kumar R, Swarnkar M, Singal G, Kumar N. IoT network traffic classification using machine learning algorithms: An experimental analysis. IEEE Internet of Things Journal. 2022, 9(2), 989-1008. DOI: 10.1109/JIOT.2021.3121517
[20] Ullah F, Ullah S, Srivastava G, Lin JCW. IDS-INT: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic. Digital Communications and Networks, 2024, 10(1), 190-204. DOI: 10.1016/j.dcan.2023.03.008
[21] Shahraki A, Abbasi M, Taherkordi A, Jurcut AD. Active learning for network traffic classification: a technical study. IEEE Transactions on Cognitive Communications and Networking. 2021, 8(1), 422-439. DOI: 10.1109/TCCN.2021.3119062
[22] Li WZ, Gao SH, Li X, Xu YT, Lu SL. Tcp-neuroc: Neural adaptive tcp congestion control with online changepoint detection. IEEE Journal on Selected Areas in Communications. 2021, 39(8), 2461-2475. DOI: 10.1109/JSAC.2021.3087247
[23] Salman O, Elhajj IH, Kayssi A, Chehab A. A review on machine learning–based approaches for Internet traffic classification. Annals of Telecommunications. 2020, 75(11), 673-710. DOI: 10.1007/s12243-020-00770-7
[24] Nascita A, Montieri A, Aceto G, Ciuonzo D, Persico V, et al. XAI meets mobile traffic classification: Understanding and improving multimodal deep learning architectures. IEEE Transactions on Network and Service Management. 2021, 18(4), 4225-4246. DOI: 10.1109/TNSM.2021.3098157
[25] Wang K, Yu XY, Lin WL, Deng ZL, Liu X. Computing aware scheduling in mobile edge computing system. Wireless Networks. 2021, 27(6), 4229-4245. DOI: 10.1007/s11276-018-1892-z
[26] Yang JY, Chen YX, Xue KP, Han JP, Li J, et al. IEACC: an intelligent edge-aided congestion control scheme for named data networking with deep reinforcement learning. IEEE Transactions on Network and Service Management. 2022, 19(4), 4932-4947. DOI: 10.1109/TNSM.2022.3196344
[27] Shingai R, Hiraga Y, Fukuoka H, Mitani T, Nakada T, Nakashima Y. Construction of an Efficient Divided/Distributed Neural Network Model Using Edge Computing. IEICE TRANSACTIONS on Information and Systems. 2020, 103(10), 2072-2082. DOI: 10.1587/transinf.2019EDP7326
[28] Zhao Y, Yin Y, Gui G. Lightweight deep learning based intelligent edge surveillance techniques. IEEE Transactions on Cognitive Communications and Networking. 2020, 6(4), 1146-1154. DOI: 10.1109/TCCN.2020.2999479
[29] Burra LR, Karuna A, Tumma S, Marlapalli K, Tumuluru P. MobileNetV2-based Transfer Learning Model with Edge Computing for Automatic Fabric Defect Detection. Journal of Scientific & Industrial Research. 2023, 82(1), 128-134. DOI: 10.56042/jsir.v82i1.69928
[30] Le KH, Le-Minh KH, Thai HT. Brainyedge: An ai-enabled framework for iot edge computing. ICT Express. 2023, 9(2), 211-221. DOI: 10.1016/j.icte.2021.12.007
[31] Cheng SB, Jin YF, Harrison SP, Quilodrán-Casas C, Prentice IC, et al. Parameter flexible wildfire prediction using machine learning techniques: Forward and inverse modelling. Remote Sensing. 2022, 14(13), 3228. DOI: 10.3390/rs14133228
[32] Zhuang YL, Cheng SB, Kovalchuk N, Simmons M, Matar OK, et al. Ensemble latent assimilation with deep learning surrogate model: application to drop interaction in a microfluidics device. Lab on a Chip. 2022, 22(17), 3187-3202. DOI: 10.1039/D2LC00303A
[33] Chen DY, Cheng SB, Hu JW, Kasoar M, Arcucci R. Explainable Global Wildfire Prediction Models using Graph Neural Networks. arXiv preprint arXiv:2402.07152, 2024. DOI: 10.48550/arXiv.2402.07152
[34] Laghari AA, Zhang XB, Shaikh ZA, Khan A, Estrela VV, et al. A review on quality of experience (QoE) in cloud computing. Journal of Reliable Intelligent Environments. 2024, 10(2), 107-121. DOI: 10.1007/s40860-023-00210-y
[35] Yamazaki T. Quality of experience (QoE) studies: Present state and future prospect. IEICE Transactions on Communications. 2021, 104(7), 716-724. DOI: 10.1587/transcom.2020CQI0003
[36] Agarwal B, Togou MA, Ruffini M, Muntean GM. Qoe-driven optimization in 5g o-ran-enabled hetnets for enhanced video service quality. IEEE Communications Magazine. 2023, 61(1), 56-62. DOI: 10.1109/MCOM.003.2200229
[37] Gelenbe E, Domanska J, Fröhlich P, Nowak MP, Nowak S. Self-aware networks that optimize security, QoS, and energy. Proceedings of the IEEE. 2020, 108(7), 1150-1167. DOI: 10.1109/JPROC.2020.2992559
[38] Putra FPE, Ubaidi U, Aziz M, Irfan M, Alim R. Improving Network Service Quality in parts of Sampang City: QoS Evaluation and User Perception of QoE. Brilliance: Research of Artificial Intelligence. 2024, 4(1), 408-412. DOI: 10.47709/brilliance.v4i1.4311
[39] Yuan YZ, Cao X, Liu ZX, Chen CL, Guan XP. Adaptive priority adjustment scheduling approach with response-time analysis in time-sensitive networks. IEEE Transactions on Industrial Informatics. 2022, 18(12), 8714-8723. DOI: 10.1109/TII.2022.3150044
[40] Velayutham A. Methods and algorithms for optimizing network traffic in next-generation networks: Strategies for 5g, 6g, sdn, and iot systems. Journal of Intelligent Connectivity and Emerging Technologies. 2021, 6(5), 1-26.
[41] Jung S, Kim J, Kim JH. Intelligent active queue management for stabilized QoS guarantees in 5G mobile networks. IEEE Systems Journal. 2020, 15(3), 4293-4302. DOI: 10.1109/JSYST.2020.3014231
[42] Anwar MR, Wang S, Akram MF, Raza S, Mahmood S. 5G-enabled MEC: A distributed traffic steering for seamless service migration of internet of vehicles. IEEE Internet of Things Journal. 2022, 9(1), 648-661. DOI: 10.1109/JIOT.2021.3084912
[43] Perera S, Gupta V, Buckley W. Management of online server congestion using optimal demand throttling. European Journal of Operational Research. 2020, 285(1), 324-342. DOI: 10.1016/j.ejor.2020.02.008