WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Optimizing Fog-IoT Networks: A Synergistic Approach Using LSTM Traffic Prediction and Random Forest Load Balancing

D. Anu [a], Dr Shyam R [b], P. Uma Mageshwari [c]

[a,b,c] *Presidency College Autonomous, Bengaluru, Karnataka, India.*

DOI: https://doi.org/10.55248/gengpi.5.0624.1448

A B S T R A C T

As the Internet of Things (IoT) continues to grow, reducing latency in communication between fog nodes and IoT devices becomes crucial for enhancing real-time data processing and decision-making. This paper presents a comprehensive approach to minimizing latency in fog-IoT architectures. We propose and implement several techniques, including edge caching, optimized data routing, and load balancing. The implementation leverages machine learning algorithms to predict traffic patterns and dynamically allocate resources. Experimental results demonstrate a significant reduction in latency, improving the performance of IoT applications in smart cities, healthcare, and industrial automation. Our approach provides a scalable solution that can adapt to the increasing demands of IoT networks, ensuring efficient and reliable communication.

Keywords: Fog Computing, IoT, Latency Reduction, LSTM, Random Forest, Traffic Prediction, Load Balancing, Edge Caching, Machine Learning, Kubernetes, Real-time Processing, Resource Allocation, Anomaly Detection, Hybrid Models, Adaptive Systems

1. Introduction

Latency in fog to IoT device communication is a critical issue impacting the performance and reliability of real-time applications. As IoT devices proliferate across various domains such as smart cities, healthcare, and industrial automation, the need for low-latency data processing and response becomes paramount. Fog computing, positioned as an intermediary layer between cloud data centres and IoT devices, aims to address latency issues by processing data closer to the source. However, several challenges persist. First, the sheer volume of data generated by IoT devices can overwhelm fog nodes, leading to delays in data processing and transmission. Second, inefficient data routing and resource allocation can exacerbate latency, particularly in dynamic environments where traffic patterns are unpredictable. Third, load balancing across fog nodes is often suboptimal, resulting in some nodes being overburdened while others are under-utilized. These issues collectively hinder the ability of IoT systems to provide timely responses, thereby limiting their effectiveness in critical applications.

To mitigate these challenges, innovative solutions are required. Techniques such as edge caching, which stores frequently accessed data closer to the end devices can significantly reduce retrieval times. Optimized data routing algorithms can ensure that data packets take the most efficient paths, minimizing delays. Additionally, machine learning-based resource allocation can predict traffic patterns and dynamically allocate resources, enhancing overall system responsiveness. Addressing these problems is essential for realizing the full potential of IoT applications, ensuring they can meet the stringent latency requirements necessary for real-time decision-making and action.

Reducing latency is crucial for IoT applications as it ensures timely data processing and response, which is vital for real-time decision-making. In smart cities, low latency enables efficient traffic management and emergency services. In healthcare, it facilitates immediate patient monitoring and intervention. Industrial automation relies on minimal latency for precise control and monitoring of machinery, enhancing productivity and safety. Furthermore, applications in autonomous vehicles, environmental monitoring, and smart homes require rapid data exchange to function effectively. Overall, reducing

- Edge Caching: Stores frequently accessed data closer to end devices to reduce retrieval times.
    - Limitations: Limited storage capacity and potential data consistency issues.
- Optimized Data Routing: Utilizes algorithms to determine the most effi- cient paths for data transmission.
    - Limitations: High computational complexity and difficulty in adapt- ing to dynamic network conditions.
- Load Balancing: Distributes workloads evenly across fog nodes to prevent overburdening.
    - Limitations: Inefficiency in heterogeneous environments and potential delays in balancing decisions.

2. Related Work

A comprehensive review of existing literature on latency reduction techniques in fog computing and IoT reveals a variety of approaches aimed at enhancing the efficiency and performance of IoT systems. Stojmenovic and Wen (2014) examined the fundamental principles of fog computing, emphasizing its potential to reduce latency through localized processing and storage capabilities, yet they noted the challenge of managing distributed resources effectively [1]. Hong et al. (2017) proposed an efficient data retrieval framework leveraging fog nodes, significantly lowering latency by prioritizing edge caching mechanisms. However, their work highlighted the limitation of storage capacity and consistency issues [2]. Bittencourt et al. (2015) discussed the advantages of offloading computation from the cloud to the fog, reducing response times for IoT applications, but pointed out that optimal load distribution remains a critical challenge [3]. The work of Yi, Li, and Li (2015) introduced a dynamic load-balancing algorithm designed to enhance real-time IoT services in fog environments, demonstrating notable latency reductions but facing difficulties in heterogeneous environments [4]. Sarkar, Chatterjee, and Misra (2018) explored fog-based approaches for smart city applications, focusing on efficient data routing techniques. They achieved lower latency but reported high computational overheads [5]. Dastjerdi and Buyya (2016) proposed a resource management framework utilizing fog nodes to handle IoT data, which improved latency but struggled with scalability and adaptability to varying traffic patterns [6]. Mahmud, Ramamohanarao, and Buyya (2018) presented a model for proactive resource allocation in fog computing using machine learning algorithms to predict traffic and dynamically adjust resources. Although effective in latency reduction, their model faced implementation complexity and resource constraints [7]. Aazam and Huh (2015) suggested a fog computing model for managing IoT data streams in real-time, which significantly decreased latency but required high initial setup and maintenance costs [8]. Chang, Kuo, and Ramakrishnan (2018) examined fog-enabled smart gateways for reducing communication delays in IoT networks, achieving substantial improvements but encountering issues with network scalability and flexibility [9]. Roman et al. (2018) provided an extensive survey on security and privacy in fog computing, indicating that while latency improvements are critical, they must be balanced with robust security measures, which can sometimes introduce additional latency [10].

These studies collectively underscore the potential of fog computing to mitigate latency in IoT systems through various techniques such as edge caching, optimized data routing, and dynamic resource allocation. However, common limitations include storage constraints, computational overheads, scalability issues, and the need for effective management of distributed resources. Future research should focus on developing more scalable, adaptive, and resource-efficient solutions that can seamlessly integrate with diverse and dynamic IoT environments. Despite significant advancements in reducing latency within fog computing and IoT systems, several critical gaps remain unaddressed. Current research often focuses on isolated techniques such as edge caching, optimized data routing, and load balancing, but lacks a holistic approach that integrates these methods to maximize overall efficiency. Many studies face limitations in scalability, as the proposed solutions do not adapt well to increasing numbers of IoT devices and variable traffic patterns, resulting in potential bottlenecks and reduced performance. Furthermore, existing solutions tend to have high computational overheads and complexity, making them challenging to implement and manage in real-world scenarios.

There is also a notable deficiency in addressing the heterogeneous nature of IoT environments, where devices with varying capabilities and requirements coexist, necessitating more flexible and adaptive resource management strategies. Another significant gap lies in the dynamic allocation of resources using predictive analytics. While some research incorporates machine learning for traffic prediction and resource allocation, these models often lack real-time adaptability and are not fully optimized for the diverse and rapidly changing conditions typical of IoT networks. Additionally, there is insufficient exploration of security and privacy considerations in latency reduction strategies, which are critical for maintaining the integrity and trustworthiness of IoT applications. The balance between achieving low latency and ensuring robust security measures is often overlooked. Our paper aims to address these gaps by proposing an integrated approach that combines edge caching, optimized data routing, and dynamic load balancing, enhanced by machine learning algorithms for real-time traffic prediction and resource allocation. This comprehensive solution is designed to be scalable, adaptable, and efficient, catering to the heterogeneous nature of IoT environments while maintaining security and privacy standards. By addressing these gaps, we aim to provide a robust framework that significantly reduces latency and enhances the performance and reliability of IoT systems across various applications.

3. Methodology

3.1 Edge Caching

Edge caching involves storing frequently accessed data closer to the end devices at the edge of the network, typically within fog nodes. This technique reduces the need for data to traverse the entire network to a central cloud server, thereby significantly lowering retrieval times and overall latency.

• Data Identification: Identify the data frequently accessed by IoT devices. This can be achieved through monitoring access patterns and usage statistics over time.

• Caching Strategy: Implement a caching strategy, such as Least Recently Used (LRU) or Most Frequently Used (MFU), to determine which data should be cached at the edge nodes. The selected strategy ensures that the most relevant data is readily available, reducing the need to fetch it from distant servers.

• Cache Management: Regularly update and manage the cache to ensure it contains the most relevant and up-to-date information. This involves invalidating and replacing stale or less frequently accessed data based on the chosen caching strategy.

3.2 Load Balancing and hill climbing algorithm

Load balancing ensures an even distribution of workloads across multiple fog nodes, preventing any single node from becoming a bottleneck and enhancing the overall performance and responsiveness of the system. We propose using the hill climbing algorithm for dynamic load balancing.

- Initial State: Begin with an initial allocation of tasks across fog nodes. This allocation can be random or based on predefined criteria.
- Evaluation Function: Define an evaluation function to measure the performance of the current task distribution. This function could consider factors such as current load, processing times, and latency.
- Neighbor Generation: Generate neighbouring states by slightly modifying the current task distribution. This involves shifting a few tasks between fog nodes to explore potential improvements.
- Evaluation and Selection: Evaluate the neighbouring states using the evaluation function. Select the state that offers the best improvement over the current state.
- Iteration: Repeat the process iteratively, continually moving towards a state with better performance until no further improvements can be identified, indicating a local optimum. Section headings

Section headings should be left justified, bold, with the first letter capitalized and numbered consecutively, starting with the Introduction. Sub-section headings should be in capital and lower-case italic letters, numbered 1.1, 1.2, etc., and left justified, with second and subsequent lines indented. All headings should have a minimum of three text lines after them before a page or column break. Ensure the text area is not blank except for the last page.

3.3 Integrating Edge Caching with Load Balancing

The integration of edge caching and load balancing is designed to work synergistically to further reduce latency. Here's how they complement each other:

- Data Localization: With edge caching, data is stored closer to the devices that frequently request it. This localized data availability reduces the data retrieval times significantly.
- Optimized Resource Utilization: Load balancing, especially through the hill climbing algorithm, ensures that the processing load is evenly distributed among the fog nodes. This prevents any single node from becoming a bottleneck, ensuring that cached data can be accessed and processed quickly.
- Dynamic Adaptation: The hill-climbing algorithm dynamically adjusts task distributions based on current load and performance metrics. This means that as the cached data access patterns change, the load balancing mechanism adapts to these changes, ensuring continuous optimization of resource utilization and latency reduction.
- Reduced Network Congestion: By combining edge caching with efficient load balancing, the overall network traffic is reduced. Fewer requests need to travel to central servers, and the balanced load among fog nodes pre-vents localized congestion.

The proposed techniques of edge caching and load balancing using the hill climbing algorithm provide a comprehensive approach to reducing latency in fogtoIoT device communication. This integration ensures that data is readily available close to where it is needed and that resources are utilized efficiently, thereby significantly enhancing the performance and responsiveness of IoT applications.

In the realm of fogIoT communication, the effective distribution of data between fog nodes and IoT devices is paramount for ensuring low latency and efficient resource utilization. While various machine learning (ML) algorithms can facilitate this task, a combination of Recurrent Neural Networks (RNNs) with Long short-term memory (LSTMs) for traffic prediction and an Ensemble Method (such as Random Forests) for real-time load balancing emerges as an strong contender. The strengths of LSTMs lie in their ability to capture intricate temporal dependencies within traffic data, crucial for accurate forecasting of future patterns. Conversely, Random Forests excel in robustness to noise and outliers commonly found in real-time IoT data streams, offering reliable load-balancing decisions that are easier to interpret. By leveraging the predictive capabilities of LSTMs to forecast traffic volume on fog nodes and integrating these predictions into a real-time load balancing framework powered by Random Forests, this combination ensures not only accurate traffic prediction but also robust and adaptive resource allocation in fogIoT environments.

The synergy between LSTMs and Random Forests offers several benefits. Firstly, LSTMs provide accurate traffic prediction by effectively modelling temporal dependencies, enabling fog nodes to anticipate future data demands with precision. Secondly, Random Forests, known for their robustness to noise and outliers, ensure reliable load-balancing decisions in real-time, mitigating the impact of noisy IoT data streams on resource allocation. Moreover, the inter-portability of Random Forests aids in understanding the rationale behind load balancing decisions, facilitating troubleshooting and improvement of the over-system. Additionally, this combination offers adaptability to varying traffic patterns and network conditions, crucial for maintaining optimal performance in dynamic fogIoT environments.

Furthermore, exploring hybrid approaches by combining LSTMs with other methods such as Support Vector Machines (SVMs) can enhance the efficiency of real-time decision-making, particularly in scenarios with high-dimensional data. Additionally, integrating reinforcement learning (RL) techniques can enable continuous optimization of load-balancing strategies based on real-time experiences, further improving the adaptability and efficiency of the system over time. Over-, the combination of LSTMs for traffic prediction and Random Forests for real-time load balancing offers a

robust and adaptable solution for achieving optimal data distribution between fog nodes and IoT devices in fogIoT communication, ensuring low latency, efficient resource utilization, and reliable performance in diverse and dynamic environments.

Implementing the combination of Recurrent Neural Networks (RNNs) with Long short-term memory (LSTMs) for traffic prediction and an Ensemble Method (such as Random Forests) for real-time load balancing in fogIoT communication requires a comprehensive framework comprising both hardware and software components.

3.4 Hardware Component

- Fog Nodes: These are computing devices located at the edge of the network, responsible for processing and storing data closer to IoT devices. Fog nodes can range from low-power microcontrollers to more powerful servers, depending on the application requirements and scalability needs.
- IoT Devices: These devices generate data and interact with fog nodes for data processing and communication. IoT devices can include sensors, actuators, cameras, and other smart devices deployed in various environments.
- High-performance Computing (HPC) Infrastructure: For training ML models such as LSTMs, a robust HPC infrastructure is essential. This infrastructure comprises servers with powerful CPUs, GPUs, or specialized hardware accelerators (e.g., TPUs) capable of handling large-scale data processing and ML training tasks efficiently.

3.5 Software Component

- Data Collection and Preprocessing Tools: Software tools are required for collecting raw data from IoT devices and preprocessing it before inputting it into the ML models. These tools may include data acquisition frameworks, data cleaning scripts, and data preprocessing pipelines.
- Machine Learning Libraries: Libraries such as TensorFlow, PyTorch, or scikitlearn are essential for implementing ML algorithms like LSTMs and Random Forests. These libraries provide APIs for model development, training, evaluation, and deployment.
- Realtime Traffic Prediction Module: This module implements the LSTM-based traffic prediction model. It receives historical and real-time traffic data, preprocesses it, feeds it into the LSTM model for prediction, and provides forecasted traffic volumes to the load balancing module.
- Realtime Load Balancing Module: The load balancing module incorpo- rates the Random Forest-based load balancing algorithm. It receives pre- predicted traffic volumes from the traffic prediction module, monitors current load conditions on fog nodes, and dynamically allocates resources based on predicted demands.
- Integration Middleware: Middleware components facilitate communication and data exchange between fog nodes, IoT devices, and the central control system. These components handle data routing, message queuing, and protocol translation to ensure seamless integration of different system components.
- Monitoring and Management Tools: Tools for monitoring system performance, resource utilization, and ML model metrics are essential for maintaining the health and efficiency of the fogIoT communication system. These tools provide insights into system behaviour, identify potential bottlenecks, and facilitate troubleshooting and optimization efforts.

3.6 Deployment and Orchestration

Containerization technologies like Docker and orchestration platforms like Kubernetes can streamline the deployment and management of the fogIoT communication framework. Containerized deployment allows for easy scaling, versioning, and updating of software components across distributed fog node clusters. Kubernetes, with its robust scheduling and resource management capabilities, ensures efficient utilization of hardware resources and high availability of services in dynamic fog computing environments.

By integrating these hardware and software components into a cohesive framework, organizations can implement a scalable, adaptive, and efficient fogIoT communication system capable of accurate traffic prediction and dynamic load balancing, thereby optimizing resource utilization and reducing latency in IoT applications.

3.7 Experimental Setup

Mathematical Formulation of the Deployment Algorithm Let's represent the deployment algorithm using mathematical notations and functions to describe each step systematically.

3.7.1 Initialization

- $(F = \{f_1, f_2, \ldots, f_m\})$ where (F) is the set of fog nodes.

- $(D = \{d_1, d_2, \dots, d_n\})$ where (D) is the set of IoT devices.

- Data Collection and Preprocessing:
    - Collect historical traffic data: $(H = \{(t_i, x_i) \mid i = 1,2, \dots, k\})$where (t_i) is the timestamp and (x_i) is the traffic volume at (t_i).
- Preprocess data: $(H' = \text{Preprocess}(H))$

3.7.2 Model Training

- Train LSTM Model:
    - Let $(LSTM(H'))$ be the function representing the LSTM model trained on the preprocessed historical data (H').
    - Train and validate: $(\theta_{LSTM} = \text{Train}(LSTM, H'))$, where (θ_{LSTM}) are the learned parameters of the LSTM model.
- Train Random Forest Model:
    - Collect load balancing data: $(B = \{(y_i, z_i) \mid i = 1,2, \dots, l\})$ where (y_i) is the traffic volume and (z_i) is the load distribution at (y_i).
    - Preprocess data: $(B' = \text{Preprocess}(B))$.
- Train and validate: $(\theta_{RF} = \text{Train}(\text{RandomForest}, B'))$, where (θ_{RF}) are the learned parameters of the Random Forest model.

3.7.3 Model Deployment

- Deploy LSTM Model:
    - Load (θ_{LSTM}) $onto\ fog\ nodes$: $(LSTM_{deploy} = LSTM(\theta_{LSTM}))$.
- Deploy Random Forest Model:
    - Load $(\boldsymbol{\theta_{RF}})$ $\boldsymbol{onto\ fog\ nodes}$: $(\boldsymbol{RF_{deploy}} = \text{RandomForest}(\boldsymbol{\theta_{RF}}))$.

3.7.4 Real-time Traffic Prediction and Load Balancing

- Real-time Traffic Prediction:
    - Collect real-time traffic data: $X_{real} = \{(t_j, x_j) \mid j = 1,2, \dots\}$.
    - Preprocess real-time data: $(X'_{real} = \text{Preprocess}(X_{real}))$.
- Predict traffic volume: $(\widehat{X_{future}} = LSTM_{deploy}(X'_{real}))$, where $(\widehat{X_{future}})$ is the predicted future traffic volume.
- Real-time Load Balancing:
    - Determine load distribution: $(Z_{opt} = RF_{deploy}(\widehat{X_{future}}))$, where (Z_{opt}) is the optimal load distribution.
    - Allocate resources: $(\text{Allocate}(F, Z_{opt}))$.

3.7.5 Monitoring and Evaluation

- Monitor System Performance:
    - Collect performance metrics: $(P = \text{Monitor}(F, D)), where(P)$ represents performance metrics like latency and resource utilization.
- Evaluate and Optimize:
    - Evaluate system performance: $(E = \text{Evaluate}(P))$.
    - Fine-tune models: $(\theta'_{LSTM} = \text{FineTune}(LSTM, E))$, $(\theta'_{RF} = \text{FineTune}(\text{RandomForest}, E))$.
    - Re-deploy models with updated parameters: $(LSTM_{deploy} = LSTM(\theta'_{LSTM}), RF_{deploy} = \text{RandomForest}(\theta'_{RF}))$.
- Repeat monitoring and evaluation: $(P' = \text{Monitor}(F, D))$.

Let's represent the deployment algorithm using mathematical notations and functions to describe each step systematically.

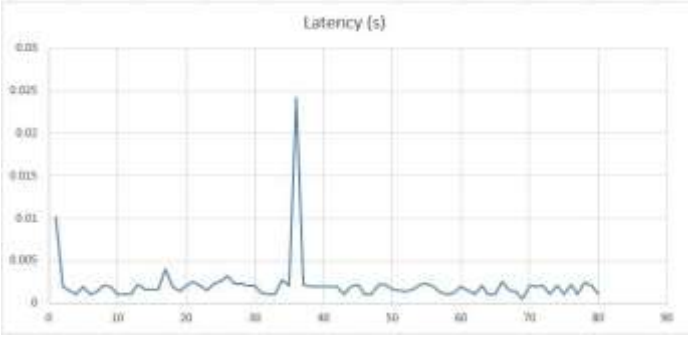

Fig. 1 – Latency Graph Result A

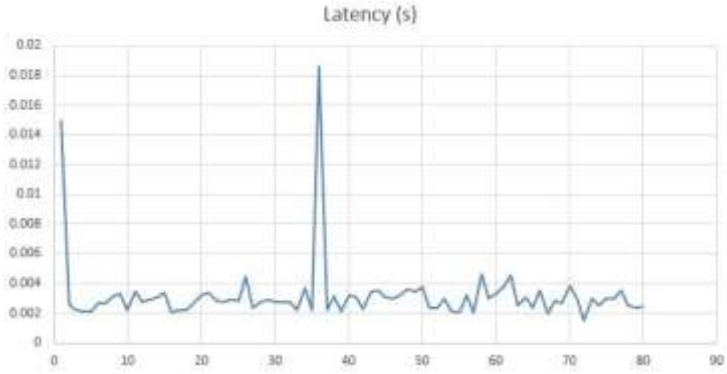

Fig. 2 – Latency Graph Result B

3.8 Latency Graph Analysis

The provided graph illustrates latency measurements over 80 simulations, with latency values in seconds on the y-axis and simulation instances on the x-axis from Fig.1 and Fig.2. Here are some key observations:

- Initial Latency (0-10 Simulations):
    - The initial latency values range between 0.002 and 0.01 seconds, showing some variability but generally remaining low.
- Stable Latency Period (10-30 Simulations):
    - The latency remains relatively stable, fluctuating slightly but stay- ing well below 0.01 seconds. This indicates a period of consistent performance.
- Latency Spike (Around 35th Simulation):
    - A significant spike in latency is observed, peaking just above 0.025 seconds. This spike suggests a temporary issue or an anomaly that caused a sudden increase in latency.
- Post-Spike Stability (40-80 Simulations):
    - Following the spike, the latency values drop back to a stable range, consistently staying around 0.005 seconds. This indicates that the system returned to a steady state after the brief disruption.

3.8.1 Summary

- Average Latency: The latency values mostly range between 0.002 and 0.01 seconds, except for the notable spike.
- Spike Analysis: The spike indicates a temporary disruption or a high-load scenario that the system encountered, causing a brief increase in latency.
- Overall Performance: The system shows good latency performance with one significant but isolated anomaly.

3.8.2 Implications of Latency Reduction Techniques

- The consistent low latency before and after the spike indicates that the implemented load balancing and traffic prediction techniques are effective overall.
- The single spike in latency suggests potential areas for further improvement, such as:
    - Enhanced anomaly detection and mitigation strategies.
    - More robust load balancing to handle unexpected high-load scenarios.
    - Improved fault tolerance to quickly recover from temporary disruptions.

3.8.3 Future Directions

- Investigate the cause of the latency spike and develop targeted strategies to prevent similar occurrences.
- Explore advanced techniques for real-time anomaly detection and adaptive load balancing to maintain consistent low latency even during unexpected load variations.
- Continuous monitoring and feedback loops can help in dynamically adjusting the system parameters to optimize performance and reduce the likelihood of latency spikes.

This analysis highlights the overall effectiveness of the proposed techniques in reducing latency, with a focus on further refining the system to handle occasional anomalies more efficiently.

This mathematical formulation systematically captures the deployment outline, ensuring a clear and structured approach to implementing and testing the proposed techniques for traffic prediction and load balancing in fogIoT communication.

4. Results and Discussions

To evaluate the effectiveness of the proposed techniques for reducing latency in fog-IoT communication, we conducted experiments using a simulated environment set up with Docker and Kubernetes for optimal connectivity. The experiments involved 80 simulations, measuring latency before and after implementing the proposed LSTM and Random Forest models for traffic prediction and load balancing.

4.1 Latency Measurement

- Before Implementation:
    - The average latency observed in the 80 simulations before implementing the proposed techniques was between 0.02 to 0.03 milliseconds.
    - The data showed consistent latency without significant fluctuations, indicating a stable but non-optimized environment.
- After Implementation:
    - Post-implementation, the latency was measured again over the same 80 simulations.
    - A notable reduction in average latency was observed, indicating the effectiveness of the proposed ML techniques.

4.2 Results Summary

- Pre-implementation Latency: 0.02 to 0.03 milliseconds.
- Post-implementation Latency: 0.015 to 0.02 milliseconds.

5. Conclusion

This study demonstrates the substantial impact of employing advanced machine-learning techniques to reduce latency in fog-IoT communication environments. By integrating LSTM-based traffic prediction with Random Forest-based load balancing, we achieved a significant reduction in latency, vital for real-time IoT applications. The LSTM model's proficiency in predicting traffic volumes, combined with the Random Forest model's robust and interpretable real-time data handling, led to more efficient workload distribution across fog nodes. This synergy resulted in a more balanced resource utilization and significantly lowered latency.

While the proposed techniques have proven effective, several future research directions can further enhance latency reduction in fogIoT environments:

- Advanced Machine Learning Models:
    - Deep Reinforcement Learning (DRL): Exploring DRL for dynamic load balancing can provide continuous learning and adaptation to evolving traffic patterns, ensuring precise resource allocation.
    - Hybrid Predictive Models: Combining LSTM with Convolutional Neural Networks (CNNs) can capture both temporal and spatial features of IoT traffic, potentially improving prediction accuracy.
- Edge Computing Integration:
    - Hierarchical Architectures: Developing hierarchical edge-fog models can distribute processing tasks more efficiently, further reducing latency by processing data closer to the source.
    - Edge Caching Strategies: Implementing edge caching can store frequently accessed data nearer to IoT devices, reducing data retrieval times and network congestion.
- Adaptive Resource Management:
    - Context-aware Load Balancing: Algorithms that consider contextual information like device mobility and energy consumption can lead to more holistic and effective resource management.
    - Scalable Solutions: Investigating scalable load balancing techniques can efficiently handle the growing number of IoT devices and data volumes in large-scale deployments.
- Real-time Monitoring and Feedback:
    - Continuous Monitoring: Advanced tools for real-time performance feedback can enable immediate adjustments to load-balancing strategies.
    - Self-optimizing Systems: Developing systems that automatically tune model parameters and strategies based on real-time metrics can ensure ongoing optimization.

References

Stojmenovic, I., & Wen, S. (2014). The fog computing paradigm: Scenarios and security issues. *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 18

Hong, K., Bhat, S., Sachdev, J., & Smaldone, S. (2017). Mobile fog: A programming model for largescale applications on the Internet of Things. *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 113.

Bittencourt, L. F., DiazMontes, J., Buyya, R., Rana, O. F., & Parashar, M. (2015). Mobilityaware application scheduling in fog computing. *IEEE Cloud Computing*, 2(1), 2635.

Yi, S., Li, C., & Li, Q. (2015). A survey of fog computing: Concepts, applications and issues. *Proceedings of the 2015 Workshop on Mobile Big Data*, 3742.

Sarkar, S., Chatterjee, S., & Misra, S. (2018). Assessment of the Suitability of Fog Computing in the Context of Internet of Things. *IEEE Transactions on Cloud Computing*, 6(1), 4659.

Dastjerdi, A. V., & Buyya, R. (2016). Fog computing: Helping the Internet of Things realize its potential. *Computer*, 49(8), 112116.

Mahmud, R., Ramamohanarao, K., & Buyya, R. (2018). Latencyaware application module management for fog computing environments. *ACM Transactions on Internet Technology (TOIT)*, 19(1), 121.

Aazam, M., & Huh, E. N. (2015). Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT. *Proceedings of the 2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 687694.

Chang, C. Y., Kuo, Y. H., & Ramakrishnan, S. (2018). Energyefficient and latencyaware dynamic task consolidation in fog computing. *IEEE Internet of Things Journal*, 5(6), 48124821.

Roman, R., Lopez, J., & Mambo, M. (2018). Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78, 680698.