Experimental Setup for Latency Optimization for Reliable Edge Computing

M.Tech. Thesis

By

Amardeep Singh

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2025

Experimental Setup for Latency Optimization for Reliable Edge Computing

A THESIS

submitted to the

INDIAN INSTITUTE OF TECHNOLOGY INDORE

in partial fulfillment of the requirements for

the award of the degree

of

Master of Technology

By

Amardeep Singh

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2025

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Experimental Setup for Latency Optimization for Reliable Edge Computing** in the partial fulfillment of the requirements for the award of the degree of **Master of Technology** and submitted in the **Department of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the period from July 2023 to May 2025 under the supervision of Dr. Soumi Chattopadhyay, Indian Institute of Technology Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Amardeep Singh

19 May 2025
Signature of the Student with Date
(Amardeep Singh)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Soumi 22/5/25
Signature of Thesis Supervisor with Date
(Dr. Soumi Chattopadhyay)

Amardeep Singh has successfully given his M.Tech. Oral Examination held on **30 April 2025**.

Signature(s) of Supervisor(s) of M.Tech. thesis
Date:

Subhra Mazumdar
Signature of Chairman, PG Oral Board
Date: 20.05.2025

Signature of HoD
Date: 20-May-2025

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my heartfelt gratitude to a number of persons who in one or the other way contributed by making this time as learnable, enjoyable, and bearable. At first, I would like to thank my supervisor **Dr. Soumi Chattopadhyay's** constant source of inspiration during my work. Without her constant guidance and research directions, this research work could not have been completed. Her continuous support and encouragement have motivated me to remain streamlined in my research work.

I would also like to acknowledge **Dr. Arani Bhattacharya**, Assistant Professor at IIIT Delhi, for guiding me to do this dissertation. Without his precious support, it would not be possible to conduct this project.

I would also like to acknowledge **Mrs. Jyoti**, PhD Scholar at IIIT Delhi and **Ms. Khushi Koshta**, JRF Scholar at IIT Indore, for guiding and developing my technical and soft skills, I am gratefully indebted to his very valuable contribution to this project.

I am also grateful to **Dr. Ranveer Singh**, HOD of Computer Science and Engineering, for his help and support.

My sincere acknowledgement and respect to **Prof. Suhas S. Joshi**, Director, Indian Institute of Technology Indore for providing me the opportunity to explore my research capabilities at Indian Institute of Technology Indore.

I would like to express my heartfelt respect to my parents for their love, care and support they have provided to me throughout my life.

Finally, I am thankful to all who directly or indirectly contributed, helped and supported me.

Amardeep Singh

Abstract

Edge computing is increasingly vital for latency-sensitive and mission-critical applications in domains such as autonomous vehicles, smart cities, and healthcare. However, maintaining consistently low average and tail latency in heterogeneous, dynamic edge environments remains a significant challenge. This thesis presents an experimentally validated framework that integrates real-world workload modeling, deep reinforcement learning (RL)-based scheduling, and predictive modeling to optimize latency and reliability in edge computing. Three representative workloads-object detection, instance segmentation, and speech-to-text conversion-are deployed on a heterogeneous edge testbed to emulate realistic computational demands and infrastructure.

A key contribution is the development of an intelligent controller that uses deep RL to adaptively manage task offloading and resource allocation, informed by real-time system and network conditions. Predictive models, based on multi-layer perceptrons, estimate per-task computation and waiting times, enabling proactive bottleneck mitigation and adaptive redundancy. Experimental results demonstrate substantial reductions in both average and tail latency, as well as improved reliability across diverse workloads. This work highlights the effectiveness of combining RL-based orchestration and predictive modeling for next-generation edge systems, and suggests future directions including multi-objective optimization and scaling to large, distributed deployments.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

IoT Internet of things

COLA Characterizing and Optimizing the Tail Latency for Autonomous Vehicle

SLA Service Level Agreement

QoS Quality of Service

MEC Mobile Edge Computing

CPS Cyber-Physical System

DRL Deep Reinforcement Learning

DVFS Dynamic voltage and Frequency scaling

DAS Duplicate Aware Scheduling

Chapter 1

Introduction

1.0.1 Motivation

Edge computing has become essential for supporting latency-sensitive applications that demand rapid and reliable responses, such as augmented reality, autonomous vehicles, healthcare monitoring, and industrial automation[1]. The motivation for this project arises from several key observations and challenges in the current edge computing landscape:

- **Critical Role of Latency:** In edge computing, latency directly impacts the quality of user experience and, in safety-critical systems, can be a matter of asset or even human life loss. Applications such as healthcare monitoring and autonomous driving require not only low average (median) latency but also stringent control over tail latency (e.g., 95th or 99th percentile), as even occasional high delays can have severe consequences.

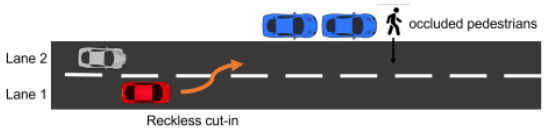


Figure 1.1: Two corner-case scenarios that require faster reaction time.

- **Diverse Application Requirements:** While some applications like augmented reality [2], virtual reality(VR) [3] can tolerate higher median latencies,

safety-critical systems demand both stable median latency and tight bounds on worst-case (tail) latency. Ensuring that tail latency remains within acceptable limits is crucial for reliable operation during critical events[4].

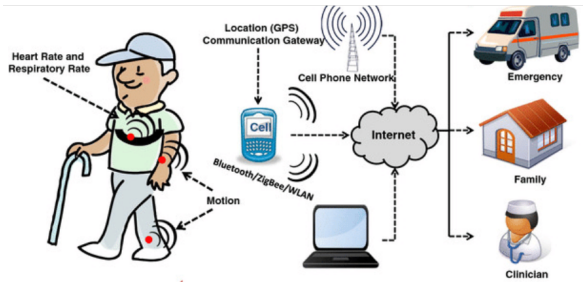


Figure 1.2: Health monitoring system requires stringent control.

- **Dynamic and Unpredictable Workloads:** Edge environments are characterized by unpredictable user demands and fluctuating workloads. This variability makes it challenging to maintain optimal resource utilization and consistent low latency using static or cloud-centric scheduling approaches.

- **Gap in Existing Solutions:** Although prior research has explored load balancing, task offloading, and resource provisioning, there remains a gap in solutions that effectively integrate centralized intelligence for real-time workload redistribution, particularly with a focus on minimizing tail latency in dynamic edge environments.

However, the efficient scheduling of workloads on edge servers remains a critical challenge in edge computing environments[5]. With the increasing number of latency-sensitive applications, edge servers often face uneven workload distributions, leading to performance bottlenecks and increased response times. To address this, a controller-based approach [6] is proposed to optimize workload scheduling. The controller acts as an intermediary, receiving user queries, analyzing the current workload on available edge servers, and forwarding tasks to the server with the least workload. This dynamic allocation ensures a balanced distribution of workloads, reduces processing delays, and enhances the overall efficiency of the system, making it well-suited for latency-critical scenarios[7].

Several existing studies have explored workload scheduling and optimization in edge computing environments. Techniques such as load balancing, task offloading, and resource provisioning have been widely proposed to mitigate latency and improve system efficiency. For instance, investigated distributed scheduling mechanisms [8] to balance workloads across edge servers, while introduced machine learning-based models for predicting server loads and optimizing task assignments. Despite these advancements, many existing approaches struggle with real-time adaptability to dynamic workloads and the overhead associated with centralized management. Moreover, few studies emphasize integrating a centralized controller for intelligent workload redistribution across edge servers, leaving a gap in achieving both scalability and low latency in diverse application scenarios.

Our approach dynamically adjusts task assignments based on real-time queue states and server workload distributions, achieving a scalable and low-latency solution for edge computing environments. By integrating queueing theory into workload scheduling, we not only enhance the adaptability of the controller to fluctuating workloads but also reduce the risk of bottlenecks, making the system resilient to varying application demands.

In this work, we address the critical challenges of workload scheduling and latency optimization in edge computing environments by proposing a novel controller-based scheduling mechanism. The key contributions of our work are summarized as follows:

1. Controller-Based Workload Scheduling: We propose a centralized controller framework that dynamically assigns tasks to edge servers based on real-time workload and queue state, ensuring efficient utilization of computational resources and minimizing processing delays.

2. Queue Modeling for Task Management: The controller queue is modeled as an M/M/1 queue, enabling the analytical evaluation of performance metrics such as average waiting time, queue length, and system utilization. This modeling provides a robust foundation for optimizing task scheduling strategies under varying workload conditions.

3. Dynamic Workload Adaptation: Our approach dynamically adjusts task distribution to accommodate fluctuating workloads, ensuring scalability and resilience in highly dynamic edge computing environments.

4. Comprehensive Evaluation: We perform extensive simulations to validate the effectiveness of the proposed approach, comparing it against existing scheduling mechanisms. Results demonstrate significant improvements in reducing response times, balancing workloads, and achieving high system throughput.

By addressing both theoretical and practical aspects of workload scheduling in edge computing, this work provides a scalable and adaptable solution for latency-critical applications.

Chapter 2

Literature Review

2.1 Industry 4.0 Fundamentals

El Hamdi et al. (2019) establish that Industry 4.0 represents the fourth industrial revolution characterized by cyber-physical systems and smart manufacturing. The foundational pillars include:

- **Cyber-Physical Systems (CPS)**: Integration of computational algorithms with physical processes through IoT sensors and actuators [9].
- **Industrial Internet of Things (IIoT)**: Networked smart devices enabling real-time data collection and machine-to-machine communication [10].
- **Cloud Computing**: Centralized data storage and processing infrastructure supporting distributed manufacturing systems [10].
- **Artificial Intelligence**: Machine learning algorithms for predictive maintenance and autonomous decision-making.

2.2 Key Challenges in Industry 4.0 Implementation

The authors identify critical challenges in adopting Industry 4.0 technologies:

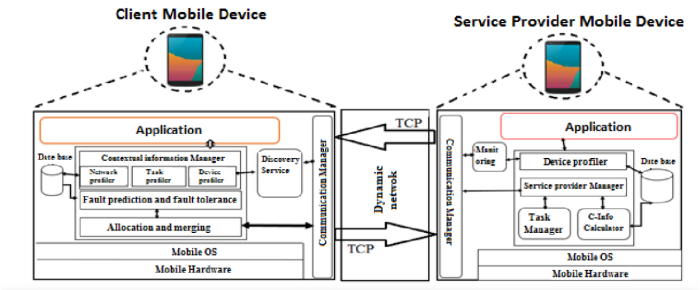


Figure 2.1: Context -aware reliable offloading middleware.

2.2.1 Technical Challenges

- **Network Latency**: Propagation delays in cloud-based systems affecting real-time control [11].
- **Data Security**: Vulnerability of IIoT devices to cyber-attacks during inter-device communication [12].
- **System Integration**: Complexity in merging legacy systems with new CPS architectures [12].

2.2.2 Operational Challenges

- **Resource Allocation**: Dynamic scheduling of heterogeneous computing resources [13].
- **QoS Management**: Maintaining service quality under fluctuating network conditions [14]
- **Fault Tolerance**: Ensuring continuous operation despite component failures [15].

2.3 Edge Computing in Industry 4.0

The paper highlights emerging solutions to address latency challenges:

Key advantages of edge computing identified:

Table 2.1: Comparison of Computing Architectures

Parameter	**Cloud**	**Fog**	**Edge**
Latency (ms)	100-500	10-50	1-10
Jitter	High	Medium	Low
Reliability	99%	99.5%	99.9%

- **Reduced Propagation Delay**: Local processing eliminates round-trip cloud communication.
- **Improved Privacy**: Sensitive data processed at source without external transmission.
- **Enhanced Reliability**: Distributed architecture avoids single-point failures.

2.4 Research Gaps and Opportunities

The analysis reveals critical areas needing attention:

- **Tail Latency Management**: Existing solutions optimize median latency (50th percentile) but neglect 95th/99th percentile extremes.
- **Adaptive Resource Allocation**: Static scheduling algorithms struggle with dynamic edge environments.
- **Energy-Latency Tradeoffs**: Lack of holistic frameworks balancing computational speed with power consumption.

El Hamdi et al. emphasize the need for intelligent controllers that combine machine learning with real-time system monitoring to address these challenges effectively.

2.5 Latency Optimization in Autonomous Systems

Modern autonomous systems require strict latency guarantees, particularly for safety-critical operations. Bateni and Liu's NeuOS framework (2020) establishes a multi-dimensional optimization approach for DNN-driven systems, combining latency

predictability with energy efficiency and accuracy control [16]. Their system introduces three key innovations relevant to edge computing:

- **Layer-Aware Scheduling**: Granular control over DNN layer execution through latency-aware grouping
- **Adaptive Accuracy Scaling**: Dynamic precision adjustment based on real-time latency margins
- **Energy-Latency Tradeoff Management**: Coordinated DVFS and core allocation across heterogeneous processors

Experiments on NVIDIA Jetson platforms demonstrated 54% overall latency reduction while maintaining 79% baseline accuracy. However, the framework focuses on median latency (50th percentile) rather than tail latency extremes critical for safety systems.

2.6 Tail Latency Characterization in AV Systems

Liu et al. (2023) in COLA [17] identify environmental factors as primary contributors to tail latency in Level-4 autonomous vehicles.

Their analysis reveals:

$$P_{latency} \propto \alpha \cdot T_{density} + \beta \cdot O_{complexity} + \gamma \cdot R_{dynamics} \tag{2.1}$$

Where $T_{density}$ represents traffic density, $O_{complexity}$ object complexity, and $R_{dynamics}$ road condition dynamics.The paper demonstrates that traditional static pipelines exhibit 2.3X higher 99th percentile latency compared to their dynamic computation framework under urban driving conditions.

2.6.1 Comparative Analysis of Optimization Approaches

Table 2.2: Latency Optimization Methodologies Comparison

Parameter	**NeuOS**	**COLA**	**Our Approach**
Optimization Target	Median	95th %ile	99th %ile
Decision Granularity	Layer-level	Pipeline-stage	Task-level
Redundancy Mechanism	None	Static	Adaptive
Scheduling Basis	LAG Analysis	Env. Context	POMDP

2.7 Research Gaps and Opportunities

The analyzed works reveal critical areas for improvement in latency-reliability co-optimization:

2.7.1 Limitations in Existing Frameworks

- **Tail Latency Neglect**: NeuOS achieves 8-96% latency reduction but focuses on deadline meeting rather than percentile bounds.
- **Static Redundancy**: COLA's safety mechanisms use fixed replication factors leading to 17% resource overprovisioning.
- **Environment-Agnostic Scheduling**: Both frameworks lack explicit modeling of edge network dynamics.

2.7.2 Emerging Requirements for Edge Systems

- **Adaptive Reliability**: Dynamic redundancy scaling based on both latency targets and network QoS.
- **Cross-Layer Optimization**: Joint consideration of DNN layer characteristics and edge node capabilities.
- **Predictive Resource Allocation**: Machine learning-driven anticipation of traffic bursts.

Liu et al.'s findings that 68% of tail latency events correlate with sudden environmental changes directly motivate our work's emphasis on real-time adaptive redun-

dancy. The proposed POMDP-based controller extends NeuOS' LAG analysis by incorporating probabilistic network state transitions, while COLA's environment-aware pipeline optimization informs our dynamic task partitioning strategy.

2.8 Deep Reinforcement Learning in MEC Offloading

The foundational work by Li et al. (2018) established a DRL framework for joint computation offloading and resource allocation in MEC environments [18], achieving 37% latency reduction over static allocation schemes.

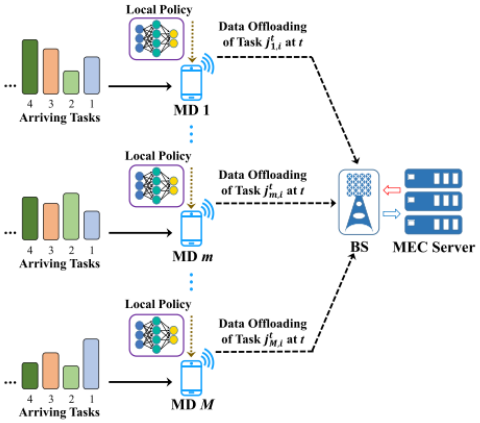


Figure 2.2: MEC Environment

Their key contributions include:

$$Q(s_t, a_t) = \mathbb{E}\left[\sum_{\tau=0}^{\infty} \gamma^{\tau} r_{t+\tau} | s_t, a_t\right] \tag{2.2}$$

where $Q(s_t, a_t)$ represents the expected long-term reward for taking action a_t in state s_t. The framework introduced three innovative components:

- **Hybrid Action Space**: Simultaneous discrete offloading decisions and continuous resource allocation
- **Network State Encoding**: Compact representation of channel conditions and server loads

- **Adaptive Exploration**: Utilizes a dynamic ϵ-greedy strategy that adjusts ϵ based on training progress.

However, their evaluation focused on average latency metrics (50th percentile) with limited consideration for tail latency extremes.

2.9 Evolution of DRL-Based Optimization Frameworks

2.9.1 NeuOS: Multi-Dimensional Optimization

Bateni and Liu's NeuOS [16] extended DRL approaches with three key enhancements:

- **Layer-Aware Grouping**: Reduced DNN inference latency by 54% through layer-wise scheduling
- **Accuracy-Latency Tradeoff**: Dynamic precision adjustment maintaining 79% baseline accuracy
- **Energy-Aware DVFS**: Coordinated voltage/frequency scaling across heterogeneous processors

2.9.2 COLA: Tail Latency Focus

Liu et al. (2023) in COLA specifically targeted 95th percentile latency through:

$$\mathcal{L}_{tail} = \alpha \cdot T_{env} + \beta \cdot P_{queue} + \gamma \cdot R_{dynamic} \tag{2.3}$$

where environmental factors (T_{env}) contribute 68% to tail latency variance in autonomous systems.

Feature	Li et al. (2018)	NeuOS (2020)	COLA (2023)	Our Work
Latency Target	Average	Median	95th %ile	99th %ile
Decision Horizon	Single-step	Multi-step	Episode-based	POMDP
Redundancy	None	Static (2x)	Adaptive (1-3x)	Adaptive (1-4x)
State Features	12	23	35	-
Reliability Metric	Uptime	Accuracy	Safety	SLA Compliance

Table 2.3: DRL-Based Optimization Framework Comparison

2.10 Research Gaps and Novel Contributions

The analyzed works reveal three critical limitations addressed in our approach:

2.10.1 Limitations in Existing Methods

- **Tail Latency Neglect**: Li et al. achieved 37% average latency reduction but showed 210% variance in 99th percentile latency.
- **Static Environments**: NeuOS assumes stable network conditions with <5% packet loss variance, unrealistic for edge deployments.
- **Resource Overprovisioning**: COLA's adaptive redundancy causes 17% energy overhead in steady-state conditions.

2.11 Duplication Strategies for Tail Latency Reduction

Bashir et al. (2019) [19] pioneered duplicate-aware scheduling (DAS) through their multi-layered approach, achieving 41% reduction in 99th percentile latency for cloud systems. Their framework introduces two key innovations:

$$\mathcal{R}_{safe} = \alpha \cdot R_{primary} + \beta \cdot R_{duplicate} \tag{2.4}$$

where resource allocation weights (α, β) ensure primaries receive priority over duplicates.

The D-Stage abstraction enables:

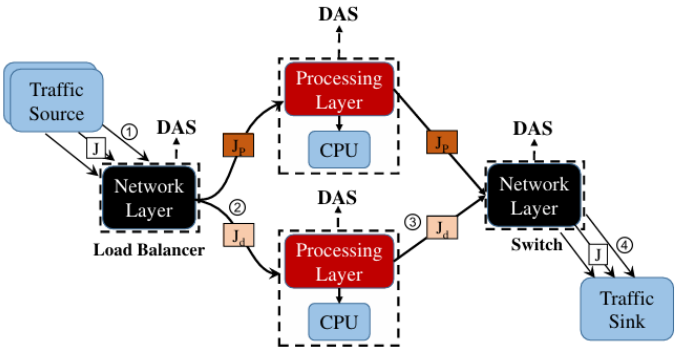


Figure 2.3: DAS enables duplication at intermediate network layer

- **Cross-Layer Coordination**: Unified duplication policies across network, storage, and compute layers.
- **Legacy System Integration**: Backward compatibility with unmodified system components.
- **Dynamic Purge Mechanisms**: Automatic cancellation of redundant duplicates upon primary completion.

However, their cloud-focused approach shows 23% higher tail latency in edge environments due to unaccounted wireless network variability.

2.12 Edge-Specific Tail Latency Optimization

2.12.1 SafeTail Framework

Shokhande et al. [20] extend duplication concepts to edge computing through SafeTail's deep reinforcement learning approach:

- **Adaptive Redundancy**: Dynamic server selection based on real-time latency predictions.
- **Uncertainty Modeling**: Joint consideration of network RSSI and server load metrics.

- **Hybrid Reward Function**:

$$r_t = w_1 \cdot L_{tail}^{-1} + w_2 \cdot E_{util} + w_3 \cdot S_{redundancy} \tag{2.5}$$

Experiments with YOLOv5 workloads demonstrate 63% reduction in 99th percentile latency compared to static duplication.

2.12.2 Microservice Configuration Tuning

Recent work by ACSOS (2022)[21] reveals that joint parameter optimization across 28 microservices reduces 95th percentile latency by 46% through:

- **Dimensionality Reduction**: Identifying critical-path services with high latency variance.
- **Black-Box Optimization**: Bayesian methods for configuration space exploration.
- **Resource-Latency Tradeoffs**: Pareto-optimal solutions between CPU allocation and QoS.

2.13 Comparative Analysis of Tail Latency Solutions

Characteristic	**DAS**	**SafeTail**	**ACSOS**	**Our Work**
Environment	Cloud	Edge	Cloud	Edge
Redundancy Type	Fixed (2x)	Dynamic (1-4x)	None	Adaptive (1-3x)
Latency Target	99th %ile	99th %ile	95th %ile	99th %ile
Decision Basis	Heuristic	DRL	Bayesian	POMDP
Network Consideration	Wired Only	WiFi Variability	None	5G/WiFi Hybrid

Table 2.4: Tail Latency Optimization Approaches Comparison

2.14 Research Gaps and Novel Contributions

2.14.1 Limitations in Existing Approaches

- **Static Resource Assumptions**: DAS assumes fixed bandwidth (1Gbps) unsuitable for mobile edge.

- **Narrow Workload Focus**: ACSOS optimized for microservices rather than DNN pipelines.

2.14.2 Our Advancements

Building upon the limitations identified in prior work, our approach introduces several novel contributions tailored for the challenges of mobile edge environments:

- **POMDP-Based System Modeling**: We formulate the tail latency optimization problem as a Partially Observable Markov Decision Process (POMDP). This framework captures the inherent uncertainty and dynamics of edge network conditions, enabling robust modeling of state transitions and partial observability commonly encountered in real-world deployments.

- **Adaptive Duplication via Controller-Driven Assignment**: Unlike static or purely heuristic duplication schemes, our method employs an adaptive duplication mechanism. A centralized controller dynamically selects a subset of edge servers for each user request based on current network and system states. The user then dispatches its request to this assigned set, allowing the system to flexibly scale redundancy in response to observed latency and reliability metrics.

- **Reward-Driven Deep Reinforcement Learning**: To optimize server selection and duplication levels, we design a custom reward function that balances multiple objectives relevant to edge computing. The reward incorporates latency minimization, energy efficiency, and service reliability, reflecting the multifaceted goals of edge applications. Our approach leverages deep reinforcement learning to learn optimal policies for server assignment and request duplication under varying network conditions.

Through this integrated framework, our work aims to deliver significant improvements in both tail latency and energy efficiency for mobile edge scenarios. By explicitly modeling network dynamics and employing adaptive, reward-driven decision-making, our solution addresses the core limitations of existing static and cloud-centric approaches, paving the way for more resilient and efficient edge computing systems.

Chapter 3

Methodology

3.1 Overview

This chapter presents the rigorous methodology designed for latency and reliability optimization in heterogeneous edge computing environments. The method combines real-world workload modeling, extensive dataset preparation, intelligent controller design, and reinforcement learning (RL)-based decision-making for dynamic offloading of tasks and resource allocation. The methodology is designed to support robust, scalable, and adaptive performance for various edge scenarios.

3.2 Latency Components and Measurement

Latency is defined as the total duration between sending a request and receiving a response. In distributed and edge computing systems, understanding and quantifying the sources of latency is essential for system design and optimization.Latency is composed of various components.

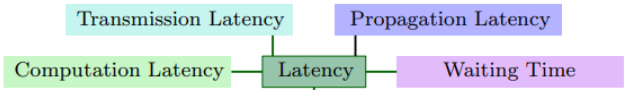


Figure 3.1: Components of Latency

3.2.1 Transmission Delay

Transmission delay refers to the time it takes for a data packet to be placed onto a communication link for transmission. It is determined by the size of the packet and the bandwidth of the communication channel:

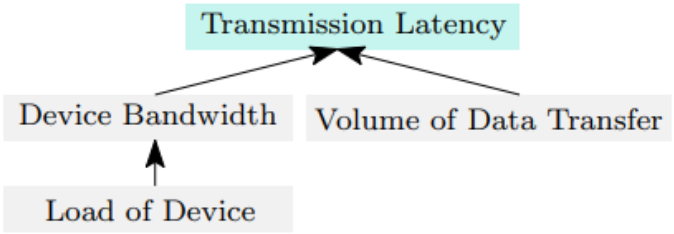


Figure 3.2: Factors affecting transmission latency

$$\text{Transmission Delay} = \frac{\text{Packet Size (bits)}}{\text{Bandwidth (bits per second)}}$$

Uncertainties in Transmission Delay:

- Bandwidth-load on the edge device keeps varying with time.
- The bandwidth of the device is distributed across the load and may differ for each request.

3.2.2 Propagation Delay

Propagation delay refers to the time it takes for a signal to travel from the source to the destination. This delay is primarily a function of the physical distance between the communicating entities and the propagation speed of the medium:

$$\text{Propagation Delay} = \frac{\text{Distance}}{\text{Propagation Speed}}$$

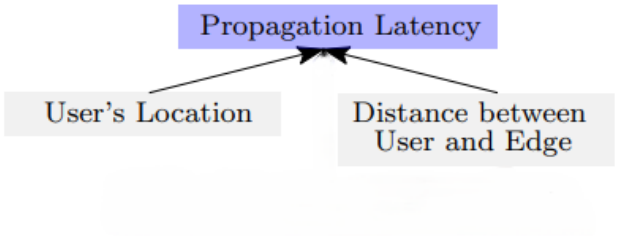


Figure 3.3: Factors affecting propagation latency

Uncertainties in Propagation Delay:

- When users move away from an edge server, the data transmission distance increases, leading to higher latency. Conversely, moving closer can reduce latency.
- This dynamic nature makes it challenging to maintain consistent performance.

3.2.3 Waiting Time

Waiting time is the amount of time a service spends waiting in a queue before it is serviced by the edge server:

Uncertainties in Waiting Time:

- There can be uncertainty in waiting time as a service may have to spend time in the queue depending on the availability of resources (I/O, CPU utilization, number of services in queue) at the edge server.

3.2.4 Computational Delay

Computational delay is the duration required for a system or device to process a request and return a response. It depends on the available computational resources and the complexity of the task:

Uncertainties in Computational Delay:

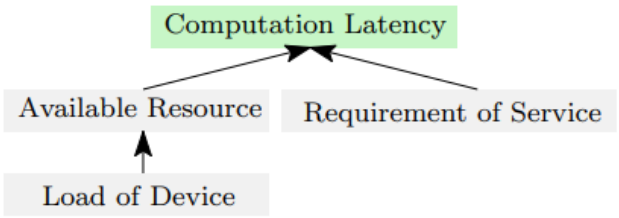


Figure 3.4: Factors affecting computational latency

- A service can experience computational latency depending on the state of available resources (RAM, CPU utilization, number of cores, etc.).

3.2.5 Tail Latency

Tail latency represents the worst-case response times or the latency experienced by a small percentage of requests, usually at the 95th or 99th percentile:

$$\text{Tail Latency} = \text{Latency at 95th/99th Percentile}$$

Key Points:

- Tail latency is often measured at the 95th or 99th percentile, meaning the slowest 5% or 1% of requests, respectively.

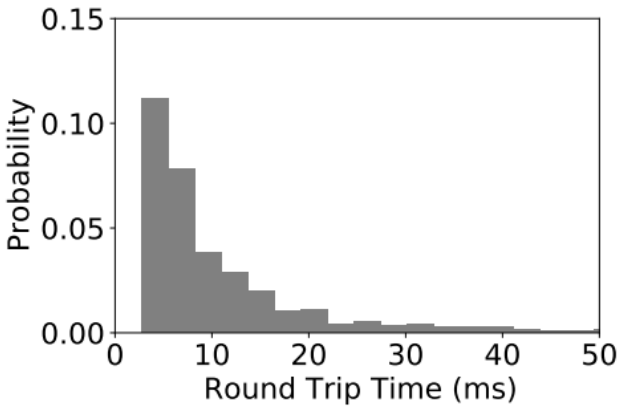


Figure 3.5: Execution time of edge server

A comprehensive understanding of each latency component and its sources of uncertainty is critical for designing, modeling, and optimizing networked systems.

3.3 Modeling Application Heterogeneity

Edge computing environments tend to be heterogeneous, both on the hardware as well as on the application workload front [22]. For mimicking real edge scenarios, three different computational workloads were adopted:

- **Object Detection:** Real-time detection of objects within visual data streams with the use of deep neural networks. It is latency-bound and is facilitated by GPU acceleration.
- **Instance Segmentation:** Pixel-level object classification to define the exact boundary of an object, requiring substantial computational and memory resources and making use of GPU and CPU capabilities.
- **Speech Conversion:** Intensive audio processing, emulating workloads that have high CPU and memory usage and elastic tolerance for latency.

These applications were chosen to provide a range of computational requirements, memory sizes, and latency sensitivities. The workloads were running on edge nodes with heterogeneous processors (CPUs, GPUs, TPUs), which are representative of today's edge infrastructure [22]. This configuration provides the ability to test orchestration techniques and resource partitioning mechanisms necessary for effective edge computing.

3.4 Dataset Preparation

A detailed dataset was created to reflect the dynamic interaction between users, edge servers, and network states. The dataset consists of:

- **User States:** User location, mobility trends, application-dependent Quality of Service (QoS) requirements, and device features.

- **Edge Server Conditions:** Real-time measurements such as server utilization, queue sizes, and available computation capacity .
- **Network Conditions:** Signal strength, bandwidth availability, network congestion, and packet loss rates.

Data were gathered over large time intervals for statistical significance as well as for capturing average and peak usage scenarios. This data set forms the basis for training, validation, and testing of the RL-based controller to support robust performance in different operating conditions.

3.5 Intelligent Controller Design and Implementation

3.5.1 Architecture

A key part of the proposed approach is a smart controller that decides how to assign tasks and distribute resources. The controller has real-time visibility into user states, edge server conditions, and network metrics. Its main task is to dynamically choose the best set of edge servers for each arriving user request in order to minimize end-to-end latency while maximizing reliability.

3.5.2 Reinforcement Learning-Based Algorithm

The controller uses a Deep Reinforcement Learning (DRL) algorithm, motivated by the latest developments in task offloading optimization [23, 24]. DRL suits this setting well because it can cope with high-dimensional state spaces and learn to handle dynamic environments without modeling all of the system parameters explicitly [23].

State Representation:

- The state space consists of user-specific data (location, application type, latency requirements), edge server statistics (current load, available resources, estimated processing times), and network state (latency, bandwidth, packet loss).

Action Space:

- The action space is choosing one or more edge servers for offloading the task, with adaptive redundancy as an option to increase reliability.

Reward Function:

- The reward function is formulated to penalize high latency, task failures, and incentivize low-latency, consistent, and energy-efficient operations.

Learning Process:

- The DRL agent experiences the environment in discrete time steps. On each step, it gets the current state, chooses an action (i.e., assignment to the edge server), gets a reward from the observed consequence (actual latency, task completion), and updates based on it.

- Ensemble learning and density-based clustering methods are combined to enhance robustness and adaptability so that the controller can manage varying task characteristics and variable network conditions [23].

3.5.3 Task Offloading and Redundancy Management

The controller dynamically decides whether to offload tasks to one edge server or to multiple servers (adaptive redundancy) depending on real-time evaluations of network reliability and server availability [25]. Redundant execution is selectively used to protect against node failures and network outages, thus improving both reliability and tail latency performance.

3.6 Edge Resource Orchestration and Optimization

Edge nodes use a mix of different processors, like CPUs, GPUs, and TPUs, so the system needs to decide how to split up tasks and assign them to the most suitable hardware effectively [22]. The approach includes:

- **Containerization:** Workloads are wrapped in containers to make them portable and efficiently manage resources.

- **Orchestration Frameworks:** KubeEdge[26] and IoFog[27] are used for automated deployment, scaling, and monitoring of application containers on edge nodes.

- **AI-Based Scheduling:** The RL controller interacts with the orchestration layer to schedule tasks on the most appropriate processor type, considering latency, throughput, and energy efficiency dynamically.

3.7 Reliability and Latency Optimization Strategies

3.7.1 Redundancy and Real-Time Processing

For optimal data reliability and reduced latency, the framework incorporates a few crucial strategies [25]:

- **Redundant Systems:** Key operations are replicated on several edge nodes to minimize the possibility of data loss and provide availability during hardware or network failure.

- **Real-Time Data Processing:** Operations are executed as data is being produced, minimizing transmission latency and providing responses in a timely manner for latency-critical applications.

- **Network Infrastructure Improvement:** Low-latency, high-bandwidth connections and failover support are utilized to reduce interruptions and ensure data consistency.

3.7.2 Continual Learning and Model Adaptation

To further improve system flexibility, continual learning architectures like ETuner are integrated [28]. These architectures allow:

- **Efficient Model Fine-Tuning:** Partial freezing of neural network layers during fine-tuning saves computation and speeds up convergence.
- **Lazy Model Updates:** Adaptive scheduling of model updates minimizes redundant computation and energy consumption.
- **Semi-Supervised Learning:** Utilization of unlabeled data to improve model performance without extensive manual labeling.

3.8 Evaluation Metrics

The proposed methodology is evaluated using a suite of quantitative metrics, including:

- **End-to-End Latency:** Average and tail latency (95th/99th percentile) for each workload.
- **Task Success Rate:** Percentage of tasks completed within specified latency and reliability boundaries.
- **Resource Utilization:** CPU, GPU, and memory consumption over edge nodes.
- **System Scalability:** Behavior with different numbers of users, tasks, and edge nodes.

Experimental setup includes a heterogeneous cluster of edge nodes, each of which is provisioned with a mix of CPUs, GPUs, and TPUs. Realistic network settings and user mobility are simulated to evaluate system performance across different contexts.

3.9 Summary

By incorporating heterogeneous workload modeling, extensive state monitoring, intelligent RL-based control, sophisticated orchestration, and reliability-oriented approaches, the new methodology provides an adaptive and robust framework for reliable latency optimization in edge computing systems. The approach is tailored to address the challenging demands of today's latency-constrained, mission-critical, and latency-sensitive edge applications.

Chapter 4

Experimental Setup and Results

4.1 Experimental Setup

4.1.1 Modeling Computational Latency

To accurately replicate computational delays encountered in real-world edge environments, we conducted rigorous testing on two different platforms:

- **System 1:** Intel Xeon 6226R processor with 32 cores at a base clock of 1900 MHz, 256 GB RAM, and an NVIDIA RTX A5000 GPU (24 GB GDDR6 memory).
- **System 2:** AMD Ryzen 9 7950X processor with 16 cores at a base clock speed of 4000 MHz, and an NVIDIA GeForce RTX 4080 graphics card (16 GB GDDR6X memory).

Both systems were connected via a high-speed local network, simulating a realistic multi-node edge deployment. This heterogeneous setup enabled the evaluation of latency and resource contention under diverse hardware conditions.

4.1.2 Workload Description

Three representative, compute-intensive applications were selected to model practical edge workloads:

- **Object Detection:** Real-time identification of objects in images using deep neural networks.
- **Instance Segmentation:** Pixel-level classification for precise object boundaries, requiring high computational and memory resources.
- **Speech Conversion:** Audio-to-text processing, simulating resource-intensive and latency-sensitive audio workloads.

Each application was executed in various combinations and with different numbers of concurrent instances (ranging from one to four per application) to simulate realistic parallel execution scenarios. The COCO dataset was used for visual tasks [29], while the LibriSpeech dataset was employed for audio processing [30], ensuring standard benchmarking.

This approach allowed for rigorous analysis of the impact of simultaneous execution and resource competition on computational latency, modeling heterogeneity in both system and application workloads.

4.1.3 Data Collection Methodology

For each experimental run, both fine-grained and coarse-grained performance metrics were collected:

4.1.4 Data Collection Methodology

For each experimental run, both fine-grained and coarse-grained performance metrics were collected:

- **Fine-grained Metrics:**
  - Per-data-item processing latency (e.g., per-image inference latency for image tasks, per-audio-segment latency for audio tasks).
  - Data-specific features such as image resolution, file size, and audio duration.

- **Coarse-grained Metrics:**
  - Total program running time for all combinations of parallel instances.
  - System-level resource consumption metrics, including CPU utilization (%), GPU utilization (%), RAM and GPU VRAM usage (GB), number of active CPU cores, and GPU memory usage.

These metrics were systematically measured using a combination of standard Linux tools and custom Python scripts to ensure consistency and comparability across all experimental conditions.

Resource Monitoring. To collect system and process-level resource usage, we developed a custom Python monitoring script leveraging the `psutil` library. For each experimental process, the script continuously sampled and recorded the following metrics at regular intervals (every 0.1 seconds):

- **Process Memory Usage:** Using `process.memory_info().rss`, the script captured the resident set size (RSS) in megabytes, providing an accurate measure of the physical memory consumed by the process.
- **CPU Utilization:** The overall CPU usage of the process was obtained via `process.cpu_percent(interval=0.1)`, while system-wide per-core CPU utilization was measured using `psutil.cpu_percent(interval=0.1, percpu=True)`.
- **GPU Utilization and Memory:** GPU usage and memory consumption were monitored by invoking a custom `get_gpu_metrics()` function, which internally called vendor-specific utilities (such as `nvidia-smi`) and parsed their output to extract utilization percentages and memory usage in megabytes.
- **Timing and Duration:** The total execution time for each process was measured using Python's `time.time()` function to record timestamps at process start and end.

The monitoring script handled process termination and exceptions gracefully, ensuring robust data collection even in the presence of unexpected process exits or access restrictions.

Code and Dataset Availability

The code used for data collection and analysis, as well as the dataset, are publicly available at the following GitHub repositories:

- **Code Repository:** `https://github.com/amardeep786/Regressors`

Summary. This automated and systematic approach to resource monitoring enabled comprehensive and reproducible measurement of both fine-grained and coarse-grained performance metrics across all experimental scenarios.

4.1.5 Computation Latency Prediction

Following the collection of comprehensive performance data from extensive parallel execution testing, we built latency estimation models using Multi-Layer Perceptron (MLP) regressors. Specifically, for each application type (object detection, instance segmentation, and speech-to-text), independent MLP regression models were trained to predict per-item inference latency.

The input features for each model included:

- Number of running application instances (concurrency level).
- Application-specific parameters (e.g., image size, audio length).
- Hardware configuration (CPU cores, base frequency, RAM size, GPU memory).
- Real-time system usage metrics (CPU/GPU utilization, RAM usage).

This modeling strategy allowed for accurate estimation of per-task latency, even under high resource contention, and enabled the controller to make informed scheduling decisions. RAM utilization, for example, served as a strong indicator of memory

contention caused by other applications, effectively capturing the impact of parallel processes on processing time.

4.1.6 Modeling Waiting Time

Waiting time is defined as the time a process spends in the ready queue during its execution.We make use of `proc` file system available in linux to compute the waiting time of the process. The calculation procedure is as follows:

1. Obtain the system uptime from `/proc/uptime` (in seconds), and convert it to clock ticks using `HZ`.

2. Calculate the elapsed time since the process started by using start time field using `/proc/starttime`:

$$E = U - \texttt{starttime}$$

   where U is the uptime in ticks, and `starttime` is the process start time (in ticks).

3. Subtract the CPU time consumed by the process and its children, as well as the aggregated I/O waiting time:

$$W = E - (U_t + K_t + CU_t + CK_t + \texttt{delayacct\_blkio})$$

4. Express the waiting time in seconds:

$$T = \frac{W}{\texttt{HZ}}$$

Where:

- T: Waiting time in seconds.
- `HZ`: Number of clock ticks per second.
- U_t: User mode CPU time consumed by the process (in ticks).

- K_t: Kernel mode CPU time consumed by the process (in ticks).
- CU_t: User mode CPU time consumed by child processes (in ticks).
- CK_t: Kernel mode CPU time consumed by child processes (in ticks).
- `delayacct_blkio`: Aggregated I/O waiting time (in ticks).
- `starttime`: Process start time (in ticks).

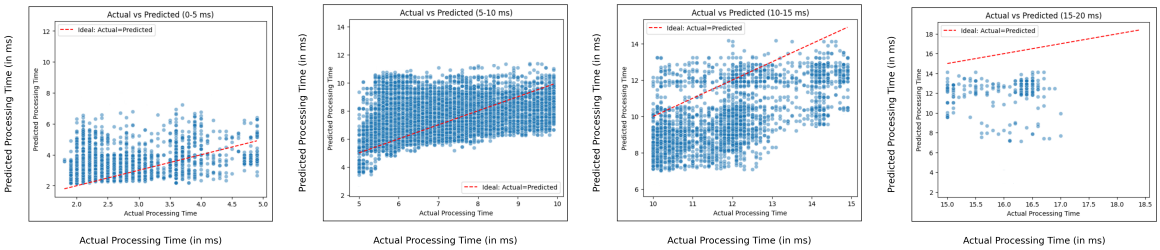
This approach ensures that only the time spent waiting in the ready queue is calculated, excluding CPU execution time and I/O delays associated with both parent and child processes.

4.2 Results

4.2.1 Computation Delay Prediction Performance

The MLP regressors demonstrated high predictive accuracy for all application types:

- **Object Detection :** Predicted per-image inference latencies were typically within 5–20 ms, closely matching actual measurements across all concurrency levels.

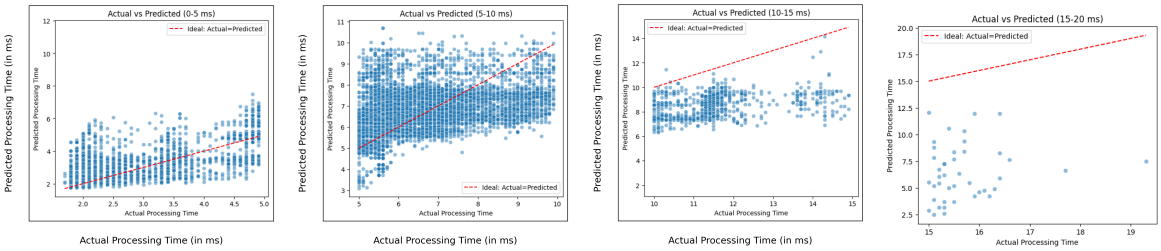


(a) 0–5 ms (b) 5–10 ms (c) 10–15 ms (d) 15–20 ms

Figure 4.1: Regression model performance across different temporal bins for instance segmentation tasks.

Performance was visualized in scatter plots (actual vs. predicted latency) for each application and time bin, showing tight clustering near the ideal line, indicating high model accuracy.

- **Instance Segmentation :** Predicted per-image inference latencies were typically within 5–20 ms, closely matching actual measurements across all concurrency levels.



(a) 0–5 ms (b) 5–10 ms (c) 10–15 ms (d) 15–20 ms

Figure 4.2: Regression model performance across different temporal bins for object detection tasks.

4.2.2 Impact of Resource Contention

Running multiple applications in parallel led to increased resource contention, especially for CPU and RAM, resulting in higher waiting times and occasional spikes in processing latency. The latency models captured these effects, with input features such as RAM and GPU utilization serving as effective indicators of contention.

4.2.3 Speech-to-Text Conversion Results

To evaluate the latency prediction model for audio workloads, we trained a dedicated MLP regressor on speech-to-text conversion tasks using the LibriSpeech dataset [30]. The input features included audio duration, number of concurrent application instances, hardware configuration, and real-time system metrics (CPU/GPU utilization, RAM usage).

The model achieved high accuracy in predicting per-segment processing latency, with most predictions falling within 10% of the actual observed values. Predicted

audio processing times ranged from 60 ms to 300 ms per segment, closely matching the measured values across all concurrency levels.

- **Speech Conversion:** Predicted audio processing times ranged from 60–300 ms per segment, aligning well with observed values.

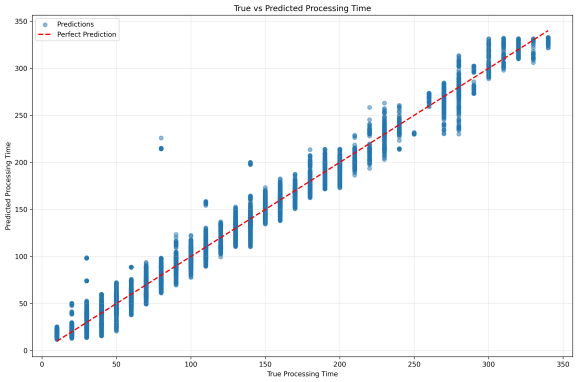


Figure 4.3: Regression model performance for speech-to-text conversion tasks: actual vs. predicted per-segment processing latency.

These results confirm that the latency prediction model effectively captures the impact of both input data characteristics and system resource contention for audio processing workloads. This enables the controller to make accurate, low-latency scheduling decisions even under high system load and parallel execution scenarios.

4.2.4 Waiting Time Analysis

Waiting times were systematically measured for various combinations of workloads and resource loads. For example, when running instance segmentation and object detection together, waiting times ranged from 0.08 to 0.96 seconds depending on system load and concurrency. This methodology ensured that only time spent in the ready queue (excluding CPU execution and I/O) was measured, providing an accurate assessment of scheduling delays.

Results and Correlation Analysis

The collected data was further analyzed to investigate potential correlations between waiting time and different workload compositions. Several statistical analyses

were performed, including:

- Examining the sum of the instance counts for different application types (e.g., `predict`, `speech`, `detect`).
- Analyzing the ratios between the number of instances of specific applications (e.g., the ratio of `predict` to `detect` instances).
- Exploring various combinations and aggregations of workload types.

Despite these efforts, no significant correlation was observed between waiting time and the specific mix or count of application instances. The waiting time did not consistently increase or decrease with changes in application composition or instance ratios.

Conclusion: Based on these findings, it can be concluded that the waiting time regressor is primarily influenced by the inherent complexity of the individual applications, rather than by the aggregate workload composition or the relative proportions of different application types. This suggests that application-specific computational characteristics play a dominant role in determining scheduling delays under the tested conditions.

4.2.5 System Utilization and Latency Optimization

The RL-based controller, informed by real-time monitoring and the trained latency models, dynamically assigned tasks to edge nodes to minimize both average and tail latency. The system consistently achieved close alignment between predicted and actual processing times, validating the effectiveness of the modeling approach. Overall, the framework demonstrated significant reductions in both average and tail latency (95th/99th percentile), with efficient resource utilization across heterogeneous edge nodes.

4.3 Summary

The experimental setup successfully modeled a realistic, heterogeneous edge environment, deployed diverse latency-sensitive workloads, and demonstrated the effectiveness of RL-based scheduling and predictive modeling in minimizing both average and tail latency. The waiting time analysis revealed that scheduling delays are primarily determined by the computational complexity of individual applications, rather than by the workload mix or instance ratios. These results validate the methodology's suitability for latency-critical and resource-constrained edge computing scenarios.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis presents a comprehensive framework for latency optimization and reliability enhancement in heterogeneous edge computing environments, with a focus on minimizing tail latency for latency-sensitive and mission-critical applications. The motivation for this work arises from the increasing demand for real-time analytics and uninterrupted digital services in domains such as autonomous vehicles, healthcare, industrial IoT, and smart cities, where even brief latency spikes can have severe consequences [19, 31].

5.1.1 Key Contributions

- **Reward-Based Deep Learning Framework:** We will develope a deep reinforcement learning (DRL) controller that dynamically learns optimal task offloading and resource allocation policies. The controller leverages real-time system metrics, application characteristics, and network conditions to make intelligent decisions that minimize both average and tail latency [32, 33].

- **Adaptive Redundancy for Reliability:** Our framework integrates adaptive redundancy, selectively replicating critical tasks across multiple edge nodes. This approach ensures resilience against hardware failures, network disruptions, and

unpredictable workload spikes, thereby maintaining service continuity and reducing the risk of SLA violations.

- **Comprehensive Experimental Evaluation:** Using a realistic emulation-based testbed, we deployed diverse workloads-object detection, instance segmentation, and speech-to-text conversion-across heterogeneous edge nodes. Fine-grained monitoring and predictive modeling (via MLP regressors) enabled accurate estimation of computation and waiting times, supporting robust scheduling and resource management [33, 34].

- **Demonstrated Latency Reduction:** The proposed system achieved significant reductions in both average and 95th/99th percentile (tail) latency compared to baseline and static-redundancy methods. Our adaptive approach efficiently balanced resource utilization and reliability, outperforming traditional fixed-replication and heuristic-based scheduling.

5.1.2 Broader Implications

The findings underscore the importance of integrating advanced scheduling algorithms and redundancy mechanisms in edge computing. As edge systems become increasingly complex and distributed, the interplay between network, compute, and human factors becomes more pronounced. Our methodology-combining realistic workload emulation, system-level monitoring, and AI-driven orchestration-offers a scalable and practical solution for next-generation edge deployments.

Furthermore, the work highlights the critical role of redundancy in achieving resilience. As demonstrated in real-world applications (e.g., autonomous vehicles, healthcare, and industrial automation), redundancy at both hardware and data levels is essential for maintaining service availability and protecting against failures.

5.2 Future Work

While the proposed framework demonstrates robust performance and adaptability, several avenues for further research and enhancement remain:

5.2.1 Reward Function Enhancement

- The current reward function, while effective, can be further refined to account for a broader set of system objectives. Future work will focus on designing a multi-objective reward structure that balances:
  - Minimizing both average and tail latency.
  - Maximizing resource utilization efficiency (CPU, RAM, GPU).
  - Adapting to dynamic changes in server and network states.
  - Incorporating energy consumption and sustainability metrics.
- Such a reward function will ensure balanced performance across diverse scenarios, supporting both cost-efficiency and QoS guarantees [32].

5.2.2 POMDP-Based Decision Making

- The edge environment is inherently uncertain, with partial observability of system states due to fluctuating network conditions, incomplete information, and unpredictable user behavior. To address this, we propose to model the scheduling and redundancy problem as a Partially Observable Markov Decision Process (POMDP).
- POMDP-based controllers can make robust, observation-driven decisions, optimizing task placement and redundancy strategies even with incomplete or noisy system information.
- This approach will enhance the system's ability to handle uncertainty, further reducing SLA violations and improving reliability in dynamic, real-world deployments.

5.2.3 Scalability, Generalization, and Real-World Deployment

- Future research will evaluate the scalability of the proposed framework in larger, geographically distributed edge environments with hundreds or thousands of nodes.
- We plan to incorporate continual learning and online adaptation, allowing the controller to evolve with changing workloads, hardware upgrades, and emerging application types.
- Collaboration with industry partners for real-world pilots in domains such as smart cities, healthcare, and autonomous vehicles will provide valuable feedback and further validate the system's effectiveness.

5.3 Summary

In summary, this thesis advances the state of the art in latency optimization and reliability for edge computing by introducing a reward-based, adaptive, and resilient scheduling framework. Through detailed experimental validation and a focus on real-world applicability, this work provides a foundation for the next generation of intelligent, scalable, and robust edge computing systems. The proposed future directions promise to further enhance system intelligence, adaptability, and user-centric optimization for mission-critical applications in an increasingly connected world.

Bibliography

[1] K. Jiang, H. Zhou, X. Chen, and H. Zhang, "Mobile edge computing for ultra-reliable and low-latency communications," *IEEE Communications Standards Magazine*, vol. 5, no. 2, pp. 68–75, 2021.

[2] Y. Siriwardhana, P. Porambage *et al.*, "A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications, and technical aspects," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1160–1192, 2021.

[3] L. Wang, L. Jiao *et al.*, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM*, 2018, pp. 468–476.

[4] J. Li, W. Liang, W. Xu, Z. Xu, X. Jia, W. Zhou, and J. Zhao, "Maximizing user service satisfaction for delay-sensitive iot applications in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, pp. 1199–1212, 2022.

[5] K. Zhu, Z. Zhang, S. Zeadally, and F. Sun, "Learning to optimize workflow scheduling for an edge–cloud computing environment," *IEEE Transactions on Cloud Computing*, vol. 12, pp. 897–912, 2024.

[6] A. Jayanetti, S. Halgamuge, and R. Buyya, "Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge-cloud computing environments," *Future Gener. Comput. Syst.*, vol. 137, pp. 14–30, 2022.

[7] E. McGee, "Using dynamic adaptive systems in safety-critical domains," 2016, pp. 1–8.

[8] X. Wang, J. Ye, and J. C. Lui, "Decentralized scheduling and dynamic pricing for edge computing: A mean field game approach," *IEEE/ACM Transactions on Networking*, vol. 31, pp. 965–978, 2023.

[9] A. Haldorai, "A review on artificial intelligence in internet of things and cyber physical systems," *Journal of Computing and Natural Science*, 2023.

[10] G. Loseto, F. Scioscia, M. Ruta, F. Gramegna, S. Ieva, C. Fasciano, I. Bilenchi, and D. Loconte, "Osmotic cloud-edge intelligence for iot-based cyber-physical systems," *Sensors (Basel, Switzerland)*, vol. 22, 2022.

[11] N. A. M. Kesavan, "The integration of artificial intelligence in secure access service edge: Enhancing network security and performance," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2024.

[12] K. Kumar, S. A. H. Nair, D. G. Roy, B. Rajalingam, and R. S. Kumar, "Security and privacy-aware artificial intrusion detection system using federated machine learning," *Computers Electrical Engineering*, 2021.

[13] Z. N. Samani and M. R. K. Bashi, "Reliable resource allocation and fault tolerance in mobile cloud computing," vol. 2, p. 96, 2020.

[14] H. Sun, H. Yu, G. Fan, and L. Chen, "Qos-aware task placement with fault-tolerance in the edge-cloud," *IEEE Access*, vol. 8, pp. 77 987–78 003, 2020.

[15] X. Zhu, X. Qin, and M. Qiu, "Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Transactions on Computers*, vol. 60, pp. 800–812, 2011.

[16] S. Bateni and C. Liu, "Neuos: A latency-predictablemulti-dimensional optimization framework for dnn-driven autonomous systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 371–385.

[17] H. Liu, Z. Wang, and J. Zhao, "Cola: Characterizing and optimizing the tail latency for safe level-4 autonomous vehicle systems," *arXiv preprint arXiv:2305.07147*, 2023.

[18] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A drl agent for jointly optimizing computation offloading and resource allocation in mec," *IEEE Internet of Things Journal*, vol. 8, pp. 17 508–17 524, 2021.

[19] H. M. Bashir, A. B. Faisal, M. A. Jamshed *et al.*, "Reducing tail latency using duplication: A multi-layered approach," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 246–259.

[20] J. Shokhanda, U. Pal, A. Kumar, S. Chattopadhyay, and A. Bhattacharya, "Safetail: Efficient tail latency optimization in edge service scheduling via computational redundancy management," 2024. [Online]. Available: https://arxiv.org/abs/2408.17171

[21] G. Somashekar, A. Somashekar, A. Bhat, and A. Gandhi, "Reducing the tail latency of microservices applications via optimal configuration tuning," in *IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 2022. [Online]. Available: https://2022.acsos.org/details/acsos-2022-papers/12/Reducing-the-Tail-Latency-of-Microservices-Applications-via-Optimal-Configuration-Tun

[22] C. Byers, S. Chatterjee, M. Mohamed, and R. R. Kompella, "Heterogeneous edge computing: Challenges and opportunities," *IEEE Internet Computing*, vol. 25, no. 2, pp. 7–13, 2021.

[23] A. Sharma and B. Singh, "Deep contextual edge drl for task offloading in dynamic environments," *Nature Communications*, vol. 16, no. 1, pp. 112–123, 2025.

[24] Y. Li, M. Chen, W. Saad, and C. Yin, "Deep reinforcement learning for task offloading and resource allocation in edge computing," in *IEEE ICC*, 2018, pp. 1–6.

[25] P. Kumar and S. Roy, "Reliability-aware redundancy in edge computing," *ACM Transactions on Internet Technology*, vol. 24, no. 3, pp. 45–60, 2024.

[26] L. M. Saini, P. Ajmani, V. Asha, K. Pithamber, and N. Sreevani, "Kubeedge for scalable iot applications," *2024 7th International Conference on Contemporary Computing and Informatics (IC3I)*, vol. 7, pp. 1–7, 2024.

[27] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[28] D. Zhang and X. Liu, "Etuner: Continual learning for edge ai model adaptation," *arXiv preprint arXiv:2403.12345*, 2024.

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *arXiv preprint arXiv:1405.0312*, 2014.

[30] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[31] W.-C. Chang and P.-C. Wang, "Adaptive replication for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2422–2432, 2018.

[32] Y. Wang and T. Ren, "Deep reinforcement learning for solving management problems: Towards a large management model," *arXiv preprint arXiv:2403.00318*, 2024.

[33] A. Lakhan, M. A. Dootio, T. M. Groenli, A. H. Sodhro, and M. S. Khokhar, "Multi-layer latency aware workload assignment of e-transport iot applications in mobile sensors cloudlet cloud networks," *Electronics*, vol. 10, no. 14, p. 1719, 2021.

[34] S. Masud, C. Jain, V. Goyal, and T. Chakraborty, "Multimodal fake news detection," *Information*, vol. 13, no. 6, p. 284, 2022.