Joint Task Offloading, DNN Pruning, and Computing Resource Allocation for Fault Detection With Dynamic Constraints in Industrial IoT

by

© *Vahidreza Niazmand*

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

November 2024

St. John's Newfoundland

Abstract

In an industrial Internet of Things (IIoT) environment, maintaining high system efficiency and stability is critical to achieving industrial automation. Deep neural networks (DNNs) have been integrated into IIoT systems to improve the intelligence and efficiency of industrial task processing. On the other hand, the execution of DNN model inference for task processing also imposes a significant computation load on end devices (e.g., monitoring sensors). To address the challenges of satisfying stringent task computing/processing requirements (e.g., latency and accuracy) in IIoT environments, offloading tasks to edge servers offers a promising solution. However, solely relying on edge-assisted offloading can introduce prolonged communication delays due to fluctuating wireless channel conditions. To enable efficient processing of a high volume of sensed industrial data for facility fault diagnosis on industrial washing machines, in this thesis, we investigate a joint task offloading, DNN model pruning, and edge computing resource allocation (JOPA) problem under a layered IIoT networking architecture. Specifically, we aim to maximize the overall network resource utilization while guaranteeing diverse and time-varying task processing delays and accuracy requirements for generated processing/computing tasks of the fault detection service. To capture the network dynamics, we formulate a stochastic optimization problem with the objective of maximizing the long-term network resource utilization with per-time-slot constraints on the end-to-end task latency and accuracy. Considering the network state transitions and the relations between network states and policies, we transform our problem into a Markov reward process (MRP) formulation where the state transitions are independent of the actions taken. To deal with

the large problem size and dynamic quality-of-service (QoS) constraints, we design a deep-reinforcement-learning (DRL) solution framework based on the soft actor-critic (SAC) algorithm, where the actor networks, critic networks, and target networks are customized to accommodate hybrid actions, achieve robust policy evaluation, and stabilize the training process, respectively. Extensive simulation results are provided to demonstrate the effectiveness of the proposed scheme and the advantages over benchmark approaches in terms of 1) achieving high network resource utilization, 2) balancing the trade-off between resource utilization and QoS satisfaction, and 3) adapting to the network load variation and dynamic QoS requirements.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Qiang (John) Ye, for his invaluable guidance, support, and encouragement throughout my research journey. His expertise and insightful feedback have been instrumental in shaping the direction and quality of this work. I am truly fortunate to have had the opportunity to learn from him. I would also like to thank Dr. Kaiyang Liu for being the host during my last year of study at MUN.

I am also very thankful to Dr. Qiang (John) Ye and the School of Graduate Studies at Memorial University of Newfoundland for their financial support which has made this research and my studies possible. Their funding has been crucial in enabling me to pursue and complete this work.

Lastly, I would like to acknowledge my family and friends for their unwavering support and motivation during my academic journey.

Contents

List of Tables

List of Figures

List of Symbols

N	Total number of sensors
G	Total number of IGWs
M_g	Number of sensors attached to IGW g
Γ	The length of each time slot
τ	The length of each mini slot
H	Task size in bits
I	Task computation intensity in FLoPs per bit
$k_{m,g}^t$	Criticality level of task m from IGW g at time slot t
A_v	Average inference accuracy of instance v
p_v	Pruning rate of instance v
$o_{m,g}^t$	Offloading variable of task m from IGW g at time slot t
q	The number of mini-slots in each time slot
C_g	IGW processing capacity
C_e	Edge server processing capacity
ψ_g^t	Uplink channel gain from IGW g to BS
z_g	Transmission power from IGW g to BS
W	Total system bandwidth
ψ_g^t	Uplink channel gain from IGW g to BS
$T^{\min}$	Minimum delay requirement
$T^{\max}$	Maximum delay requirement
$A^{\min}$	Minimum accuracy requirement

$A^{\max}$	Maximum accuracy requirement
c_g^t	Edge computing resource dedicated to IGW g at time slot t
J	The reward penalty coefficient for dropped tasks
Ψ	Number of channel gain values
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and standard deviation σ
θ_i	Critic network parameters
$\hat{\theta}_i$	Target network parameters
γ	Discount factor
ζ	Target network learning rate
α	Temperature parameter in SAC
$\mathcal{D}$	Replay buffer

Introduction

The Internet of Things (IoT) is a networking paradigm that interconnects uniquely addressable physical network devices through different communication protocols and Industrial IoT (IIoT) refers to IoT applications in industrial domains [3]. The swift evolution of the smart industry has stimulated increasing needs for such applications, including intelligent fault detection and event monitoring [4], where fault detection refers to the process of identifying malfunctioning or abnormal conditions in industrial equipment, systems or processes using data collected from IoT-enabled devices and systems. Industrial washing machines, such as dual-bearing rotating machinery for device cleaning services, function as the primary checkpoint for many different industrial devices, e.g., rail vehicles. Therefore, it is crucial to predict impending failures and mitigate unexpected downtime while satisfying the instant maintenance demands of industrial facilities for improving production efficiency and minimizing potential costs and dangers imposed by such failures [5]. However, preserving the working status and stability of an industrial washing system faces technical challenges. Sensors deployed on washing machines consistently capture sensed data, which needs to be processed for facility fault diagnosis with high accuracy and low latency. Traditional methods of fault detection, such as rule-based systems, threshold-based monitoring, and expert-driven diagnostic models, have intractable defects, including severe hysteresis, high time consumption, and over-reliance on expertise, which result in delayed maintenance measures and unexpected downtime [5]. With the enhancement of IIoT and artificial intelligence (AI) technologies, a tremendous volume of industrial data can be efficiently processed, and impending failures can be accurately predicted [6, 7, 8]. Conventional AI techniques used for fault detection, such as de-

cision trees and statistical methods, have limitations in industrial settings. They struggle with scalability when handling large, complex IIoT datasets and rely heavily on manual feature engineering, which is time-consuming and error-prone. Moreover, these models often fail to capture complex, non-linear patterns in industrial systems and require frequent retraining to adapt to changing conditions, limiting their flexibility. Additionally, they are sensitive to noise and uncertainty, leading to inaccurate fault detection and inefficient fault management [9]. Deep Neural Networks (DNNs) are advanced AI models with multiple layers that can learn patterns from large amounts of data, making them highly effective for such complex tasks. For the considered fault detection service on industrial washing machines, DNNs offer several advantages: 1) They can perform automatic feature extraction from raw data, eliminating the need for manual intervention. 2) DNNs handle large, complex datasets efficiently and excel at recognizing intricate, non-linear patterns in industrial systems. 3) They also continuously adapt to new fault conditions, making them flexible and resilient in dynamic environments. 4) DNNs are robust to noise, allowing for more accurate fault detection with fewer false alarms [10, 11]. The first stage of using DNNs is training, which involves feeding the sensor data from industrial washing machines to learn patterns or anomalies that indicate potential faults. Through backpropagation, a DNN adjusts its internal parameters to minimize detection errors, improving its ability to predict and diagnose equipment failures accurately. After training, the data sensed on each washing machine can be fed into a DNN module for processing as inference tasks. Despite exhibiting high processing accuracy compared to traditional learning methods, executing DNN inference tasks also demands substantial computation resources due to extensive floating-point operations (FLOPs) required [12]. As

the volume of sensed data in an IIoT environment increases, the computation requirements may exceed the onboard processing capacity of a washing device, resulting in prolonged computation responsiveness.

Computation offloading remains a promising strategy to alleviate the onboard computation burden [13, 14], which involves transferring computation tasks via wireless communication technologies, e.g., cellular long-term evolution (LTE) [15] to remote servers with high-performance computing resources. One typical approach for enabling computation offloading is cloud computing. Instead of relying on local hardware, cloud computing delivers computation services on-demand from remote data centers, enabling users to store and process data without needing to manage the underlying infrastructure. This approach offers flexibility, scalability, cost savings, and the ability to access computing resources with an Internet connection [13]. However, moving all computing tasks to cloud servers faces challenges such as high energy consumption and long transmission delays due to the scarcity of network bandwidth resources, which makes it difficult to meet the low cost, high accuracy, and low latency requirements of fault detection applications in IIoT environments [16]. Edge computing is proposed as a promising solution to overcome these limitations. In edge computing, computations are performed closer to the data source, i.e., near the devices generating the data, which reduces transmission delays, conserves bandwidth, and lowers energy consumption. This approach achieves faster response times, making it particularly well-suited for latency-sensitive applications such as fault detection in industrial systems. However, solely relying on edge-based solutions to offload large volumes of sensed data may lead to unpredictable service delays due to the time-varying nature of wireless channels [17]. Consequently, neither a device-only nor

an edge-only solution can effectively support DNN task inference with low delay requirements. In this context, an end-edge computing approach emerges as a potential solution. Distributing computing tasks between local devices and nearby edge servers improves the utilization of network-wide computing resources. This approach allows certain tasks to be processed locally on devices with sufficient computation power while other tasks are offloaded to edge nodes for high-performance processing. This strategy balances local-level low-latency processing with edge processing of enhanced computation performance.

This thesis aims to identify and overcome the challenges of task offloading and resource allocation when dealing with the strict requirements for bearing fault detection tasks using edge computing under time-varying network conditions.

The rest of this thesis is organized as follows. In Chapter 2, we delve into the existing studies on task offloading and resource allocation for edge computing, and AI-assisted IIoT systems. The system model is described in Chapter 3. In Chapter 4, the formulation of the joint task offloading, pruned DNN model selection, and computing resource allocation problem is presented as a stochastic optimization problem with the aim of maximizing the overall network resource utilization over time. In Chapter 5, the proposed solution is designed and customized to solve the formulated problem. Extensive simulation results are provided in Chapter 6. Concluding remarks and future research directions are given in Chapter 7.

Related Work

The research community has actively investigated task offloading and resource allocation problems in edge-enabled IIoT environments. In the following sections, we examine key issues and challenges highlighted in the literature, identify remaining

research gaps, and outline the specific contributions this thesis makes to address the gaps.

1 Edge-Assisted Task Offloading in IIoT

The literature presents various strategies for offloading, each with unique trade-offs among latency, accuracy, and energy consumption. These strategies can generally be classified into two primary categories, namely *binary offloading* and *partial offloading*.

1.1 Binary Offloading

In binary task offloading, entire tasks are either fully executed on the local IIoT device or completely offloaded to an edge server, depending on resource availability and network conditions. For instance, the authors in [18] utilize binary task offloading to jointly minimize the total latency (i.e., transmission and processing latency) and energy consumption (i.e., transmission and processing energy) for the generated DNN inference tasks. This approach ensures that the latency of each task remains below a predefined upper bound, providing a balance between performance and energy efficiency.

The challenge of DNN inference in a two-layered network is studied in [1], where in the first layer, IIoT devices generate several types of DNN inference tasks with various accuracy requirements that can either be processed locally or offloaded to an edge server equipped with high computation capacity. Specifically, compressed DNNs are deployed on IIoT devices for lower computation needs and full-scale DNNs are placed at the access point (AP) to enhance inference accuracy when tasks are

offloaded. The study focuses on minimizing end-to-end delay while ensuring the accuracy requirements of each task are met. The collaborative approach thus balances device limitations and AP resources to provide a low-latency and high-accuracy service across diverse task types.

1.2 Partial Offloading

Partial task offloading includes dividing a task into smaller components, allowing some parts to be processed locally while others are offloaded to the edge, enabling finer control over resource allocation and latency management. In [19], partial task offloading is used to address the trade-off between end-to-end latency of DNN inference tasks and energy consumption of IIoT devices under strict delay requirements. The introduced approach involves DNN partitioning, where DNN layers are divided at a specific cutting point, where all layers up to the cutting point are processed locally on an IIoT device, while intermediate results and the cutting point layer information are transmitted to the edge server to continue the inference from the edge layer onward. This partitioning technique enables partial task offloading to balance local and edge processing resources effectively.

The authors in [20] investigate DNN inference acceleration in a decentralized IIoT network, where IIoT devices generate diverse DNN inference tasks with varying latency and accuracy requirements. The system includes resource-constrained IIoT devices that can process DNN tasks either locally or offload them to a fifth-generation (5G)-empowered edge server to handle a high load of computational tasks. To balance resource usage and minimize latency, the study incorporates DNN partitioning and

early exit mechanisms for flexible, partial task offloading, where early exit mechanisms allow DNN inference tasks to terminate at intermediate layers once they reach a satisfactory accuracy level in their predictions. This process avoids unnecessary computation in deeper layers, thus reducing inference time and conserving device resources. By enabling tasks to terminate early when sufficient accuracy is achieved, early exit mechanisms provide a trade-off between processing delay and inference accuracy, making them particularly suitable for latency-sensitive IIoT applications. This collaborative approach prioritizes low latency and efficient resource utilization, especially under varying bandwidth and device-specific resource conditions.

In [21], the authors explore DNN inference using partial task offloading for cognitive big data on IIoT devices within a hierarchical setup, where resource-limited devices process DNN tasks. The system supports varied accuracy and latency needs by leveraging a two-stage approach: 1) DNN compression through knowledge distillation to create compact models and 2) acceleration via early exits, allowing tasks to be completed once they reach a sufficient confidence level. Knowledge distillation exploits knowledge transfer to compress model by following a teacher-student paradigm, in which a compact model (student) obtains knowledge from a complex model (teacher) by learning the output class distributions of the teacher. Moreover, in this work, certain parts of the DNN are processed locally on end devices, and the remaining computations are sent to the edge server as needed. This is achieved through early exit mechanisms, where inference tasks exit the model once they reach a designated confidence level, thus not requiring full processing on either the local device or an edge server. This adaptive method balances the trade-off between minimizing latency and meeting accuracy requirements, making it ideal for the two types of

considered tasks: mission-critical applications, which demand low latency, and cost-sensitive tasks, which can tolerate slight accuracy reductions for reduced resource consumption.

2 Network Resource Allocation in IIoT

Resource allocation plays a pivotal role in literature in optimizing task processing and offloading in IIoT systems, where diverse devices and tasks require efficient distribution of different network resources (e.g., computing, bandwidth, and energy resources) to meet strict latency, accuracy, and energy constraints. In IIoT environments, network resources must be carefully allocated between local devices and edge servers to balance the workload and maximize system performance based on the dynamic network environment. In [22], a joint task offloading and resource allocation scheme is proposed in an end-edge–cloud architecture to minimize the long-term average system cost, including total system delay and economic cost affected by inference accuracy of the DNN models, while guaranteeing system stability and the accuracy requirements of the tasks at each time slot.

To achieve efficient DNN inference, the authors in [23] formulate a multi-dimensional computing resource management problem in a three-layered end-edge-cloud architecture, where at the beginning of each time slot, after making the offloading decision, the edge controller determines how much bandwidth and edge computing resources should be allocated to each task. The objective is to maximize the average inference accuracy while satisfying the strict delay requirements of inference tasks. In this work, the pre-trained DNN models are placed at the end device and edge server to

provide low-latency inference services. Considering the heterogeneity of industrial facilities and the limited resources of IIoT networks, the system provides each task with two dedicated DNN models with different accuracy levels, which could be selected to perform the corresponding inference tasks according to the available resources.

The authors in [24] study a two-timescale resource allocation problem in a 5g-empowered IIoT network, which aims to simultaneously minimize the long-term end-to-end delay and the grid energy cost, in which the optimization of energy management (the utilization of harvested energy and grid energy) is performed in a large time frame, while the optimization of transmission power allocation is performed in smaller time slots.

3 Learning-Assisted Approaches in IIoT

AI has become a critical enabler in edge-assisted IIoT environments, supporting efficient task offloading and resource allocation by combining the computation power of edge servers with real-time intelligent decision-making capabilities. This section reviews recent advancements in learning-assisted approaches for effective task offloading and resource allocation in IIoT systems.

3.1 Reinforcement Learning (RL)-Based Solutions

Reinforcement learning-based solutions have gained significant attention for optimizing task offloading and resource allocation in IIoT environments. By enabling IIoT devices and edge servers to learn optimal policies from interactions with their environment, RL-based approaches adapt to dynamic network conditions, varying task

requirements, and resource constraints. These solutions use RL algorithms to balance the trade-offs between local processing and edge offloading, ensuring low latency, high accuracy, and efficient resource utilization. In [23], the joint task assignment and DNN model placement problem are first transformed into a Markov Decision Process (MDP) to handle the complexity of continuous resource allocation under high-dimensional constraints. This transformation allows the problem to be addressed through sequential decision-making, where each time slot's resource allocation choices affect overall accuracy and delay performance. In the MDP framework, the system state includes task demands and available resources, while actions represent task assignment decisions, DNN model selection, and resource allocation adjustments. By casting the problem as an MDP, they enable the use of the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, which effectively learns policies for dynamically adjusting these decisions to maximize inference accuracy while meeting delay requirements under resource constraints.

The authors in [12] address the optimization of DNN inference acceleration by minimizing task delay and maximizing inference accuracy while managing computational resources. The problem is formulated as a decentralized, partially observable Markov decision process (Dec-POMDP). The authors employ a multi-agent reinforcement learning (MARL) algorithm to solve this. Each smart device acts as an agent that makes decisions based on local observations, dynamically selecting the optimal inference branch and task offloading scheme. The decentralized approach allows each device to operate independently, making decisions without requiring global information, thus enabling efficient and scalable resource utilization and task offloading in a distributed manner.

3.2 Joint Learning- and Modeling- Assisted Approaches

In [1], the authors aim to optimize resource allocation for collaborative DNN inference in IIoT networks by minimizing service delay while ensuring long-term accuracy requirements. They frame this as a constrained Markov Decision Process (CMDP), considering IIoT devices that generate tasks with accuracy demands and offload them to edge servers based on network conditions. To address the CMDP's complexity, the authors transform it into an MDP using Lyapunov optimization. After transformation, an optimization subroutine is embedded in the proposed algorithm to directly obtain optimal edge computing resource allocation. Afterward, a DRL approach is used to optimize decision-making in the resource allocation framework. Specifically, they implement an algorithm based on the Deep Deterministic Policy Gradient (DDPG) method to handle continuous action spaces, which is suitable for complex IIoT environments where decisions involve continuous adjustments, like sampling rate selection and resource allocation.

The authors in [25] employ an artificial bee colony algorithm to acquire the optimal task offloading decisions, which is a swarm optimization technique that simulates the multivariate, multimodal functions of bee foraging behavior while also avoiding local minima. In this algorithm, the location of the bee's food source represents a possible solution to the optimization problem, and the amount of nectar from the food source corresponds to the quality (fitness) of the associated solution. Upon acquiring the optimal offloading solution, a DDPG-based algorithm is used to provide the bandwidth and computation resource allocation decisions with the aim of minimizing the maintenance cost of the IIoT devices.

4 Research Issues

Despite the aforementioned efforts in the literature, the following research issues remain worthy of investigation:

1. Most studies on facility fault diagnosis in IIoT consider a networking architecture where IIoT devices are directly connected to base stations (BSs) to enable task offloading [26, 13]. To deal with an increasing number of IIoT sensors and further alleviate the communication burden between IIoT devices and the edge layer, a hierarchical networking architecture is desired where an intermediate layer of IIoT gateways (IGWs) with high processing capacities can be deployed in between devices and edge servers for data forwarding and data processing.

2. Most of the IIoT applications consider supporting either a single type of computing task [27] or a heterogeneous set of tasks with diverse requirements [18], where the task processing requirements are usually assumed stationary. However, for industrial facility fault diagnosis applications, the processing requirements (e.g., accuracy and delay) of each fault detection task vary over time to reflect the changing criticality level of the task. For instance, as the concentricity of the shaft in a washing machine increases, it becomes more susceptible to damage. Therefore, at different time instants, the criticality level of the same type of tasks for determining the machine's health condition can be different to ensure the machine stability [5].

3. To mitigate the computation burden on IIoT devices, various DNN compression and acceleration techniques, such as early exiting and knowledge distillation,

have been employed in other studies to offer a flexible trade-off between DNN processing delay and inference accuracy [21]. Among existing compression techniques, DNN model pruning, which removes a portion of least important model weights and connections, offers certain learning model size reduction, leading to small memory footprints and great flexibility in balancing inference accuracy with processing delay [28, 29]. Therefore, deploying pruned instances of DNNs on IIoT devices with limited computation capacity while maintaining full-weight models on edge servers with greater computing resources can enhance the overall task-processing performance. When the network load is light, offloading more tasks to the edge can be effective in maximizing the inference accuracy and communication resource utilization, whereas the pruned models on IIoT devices can be more leveraged to achieve low processing latency with acceptable accuracy reduction when the wireless channel conditions are poor and/or the network is congested. DNN pruning is specifically considered in this work due to its ability to reduce the model size and computational requirements while retaining acceptable accuracy levels, making it particularly suitable for resource-constrained IIoT devices. Unlike techniques such as DNN partitioning, which requires constant communication between local devices and edge servers for processing different portions of the network, pruning enables standalone operation by deploying lightweight models directly on local devices. This is especially advantageous in the considered fault detection scenario, where tasks with varying criticality levels and strict delay requirements must be processed efficiently, even under fluctuating network conditions. By allowing flexible trade-offs between accuracy and delay, DNN pruning enhances system adaptability and ensures

optimal resource utilization across dynamic IIoT environments.

4. How computing resources are allocated among tasks affects the overall service performance. As the generated tasks from different sensors have diverse delay requirements varying over time, it becomes crucial to dynamically allocate computing resources at both the local and edge sides in response to the varying network state and task requirements [17]. The time-varying accuracy and delay requirements attributed to each task's criticality level add complexity to the problem by introducing variability in both task prioritization and resource allocation. These constraints require the system to adaptively balance trade-offs between local processing and offloading decisions while maintaining optimal resource utilization under uncertain network conditions. Considering the communication resources allocated for task offloading, the overall system resource utilization needs to be maximized to obtain an optimal task processing policy.

Therefore, in this study, we consider a layered IIoT networking architecture, where IGWs interconnect different groups of sensors deployed on industrial washing machines with an edge server. Our objective is to maximize the overall system resource utilization while guaranteeing time-varying inference accuracy and delay requirements of generated tasks to enhance the edge computing performance. To capture the network dynamics and the impact of task offloading, model pruning, and computing resource allocation on the network-wide resource utilization, we formulate a stochastic optimization problem, which is then transformed as a Markov reward process (MRP) with per-time-slot constraints on processing accuracy and delay. Specifically, we bal-

ance the trade-off between local task processing and edge processing by taking into consideration resource utilization, task processing delay, and processing accuracy. The main technical issue is how to jointly determine an optimal task offloading, local processing model pruning, and computing resource allocation policy such that the overall bandwidth and computing resource utilization can be maximized while satisfying task processing accuracy and delay requirements in the long run. To deal with a high number of generated tasks and accommodate the time-varying nature of task requirements, we design a soft actor-critic (SAC)-based deep reinforcement learning (DRL) algorithm to learn a stationary policy by interacting with the network environment, where DNN models are customized to approximate the policy and value functions [30]. Specifically, our proposed SAC-based algorithm with experience replay [31] utilizes the stochastic policy gradient method to offer more comprehensive exploration and better adapts to the time-varying task requirements than the DDPG method [32].

5 Network Model

As shown in Fig. 1, we consider a two-layer network architecture with edge computing to support a fault detection service for industrial washing machines [5]. The first layer comprises N ($N \in \mathbb{Z}^+$) industrial washing machines situated in a factory environment for rail vehicle body cleaning, each equipped with one vibration sensor to detect the operating frequency of the machine for estimating its working status (i.e., normal or abnormal). Each vibration sensor is operated at a sampling rate of X kHz. All N sensors are partitioned into G ($G \in \mathbb{Z}^+$) groups according to their geographical proximity, and each group, indexed by g ($g \in \{1, 2, .., G\}$) contains M_g sensors all connected to IGW g through wired links [5], where $\sum_{g=1}^{G} M_g = N$. In the second layer, one LTE BS is deployed to provide a wide wireless communication coverage to all G IGWs, where the LTE machine-type (LTE-M) communication technology is employed for uplink data transmission from each IGW to the BS [33]. The BS is further connected through wired connections to an edge server with computing capacity for task processing, as shown in Fig. 1.

6 Computation Model

6.1 Task Model

As depicted in Fig. 2, the sensing data acquisition process on each vibration sensor includes data sampling and quantization of original vibration signals from the ac-

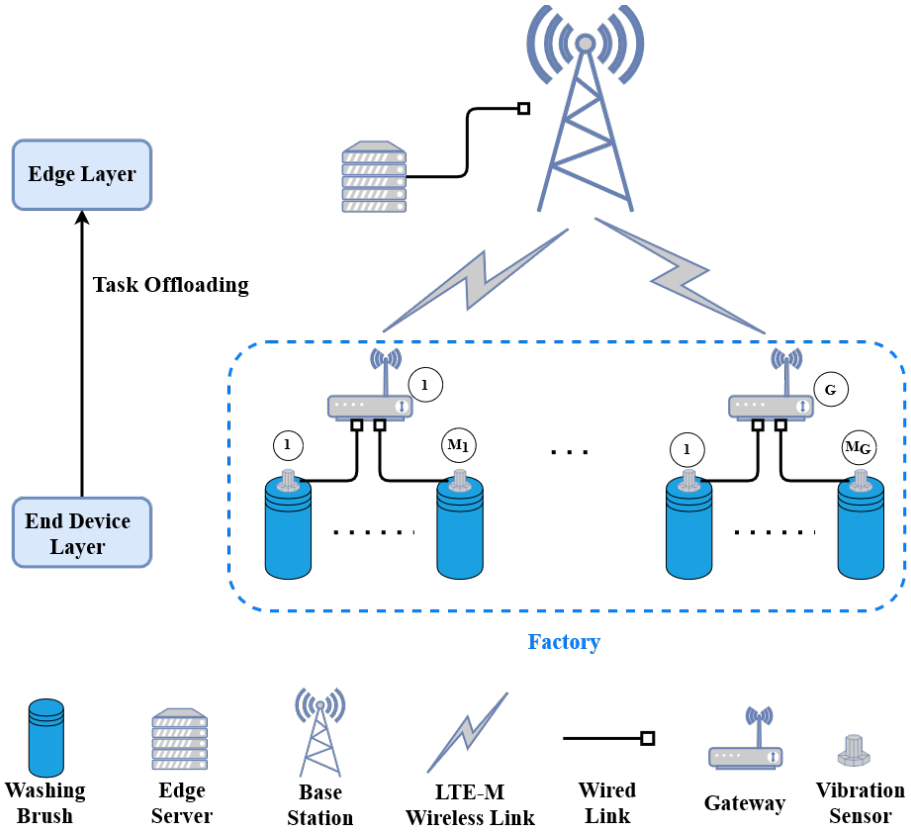


Figure 1: A two-layer network architecture with edge computing.

celerometer using an analog-to-digital converter (ADC)[1]. The digitalized signals are further transmitted through wired links, as shown in Fig. 1, to the IGW that the vibration sensor is associated with [34]. In the data link layer, the data stream from each sensor arrives at its connected gateway in fixed-size *computing tasks* to be processed for making fault detection decisions. During the network operation stage, time is partitioned into a sequence of time slots of fixed length Γ, each indexed by t ($t = 0, 1, 2...$). We assume that at the beginning of slot t, one task generated from one vibration sensor arrives at its connected gateway [5]. Therefore, at gateway g, a

[1]Note that the vibration sensors are assumed to maintain continuous functionality and no data loss occurs throughout the data acquisition process, as discussed in [5] and [34].

total of M_g tasks arrive at the beginning of each time slot. The length of each time slot is set as the maximum processing delay requirement among all tasks, denoted by $T^{\max}$. Each task is characterized by a triplet $\langle H, I, k^t_{m,g} \rangle$, where H represents the task size (in bits), I denotes the computation intensity needed to process one bit of task information (in FLoPs per bit), and $k^t_{m,g}$ indicates the criticality level of the task generated at gateway g from sensor m at time slot t. The criticality level ranges from 0 to 1, with a step increase of 0.2, and a larger $k^t_{m,g}$ reflects a higher criticality level [5]. Every task must meet a specified minimum processing accuracy and a determined maximum processing delay requirement, and tasks with higher criticality levels require a higher processing accuracy and a lower processing delay.

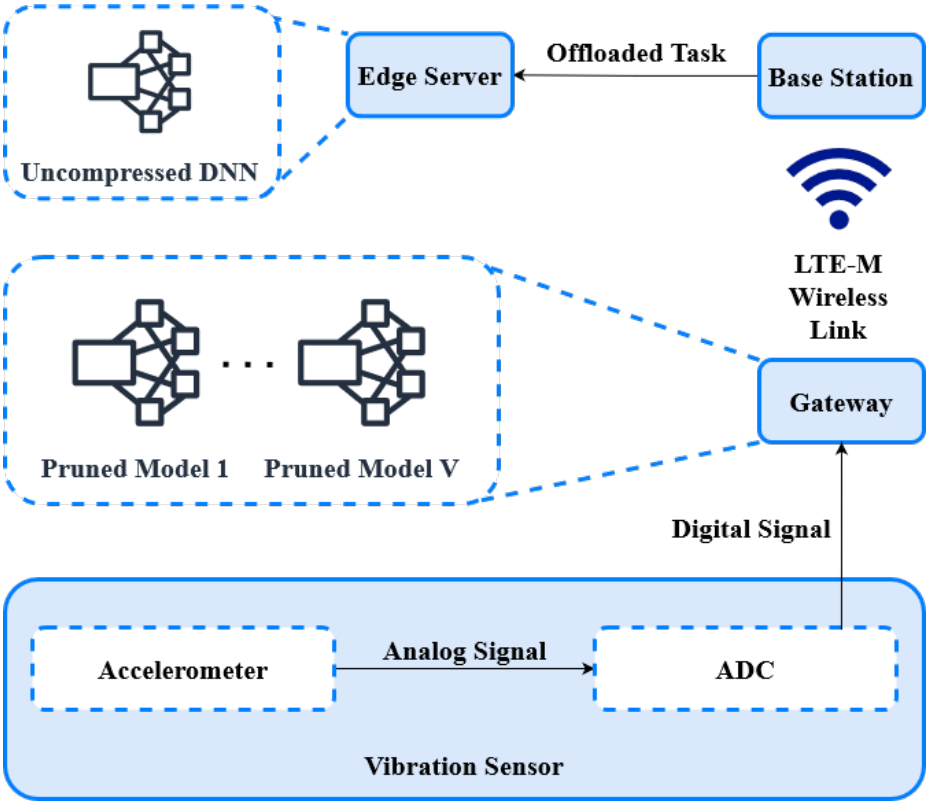


Figure 2: Data acquisition and task computing/offloading.

6.2 Convolutional Neural Network (CNN) Model Structuring and Deployment

To improve task processing efficiency, we consider deploying and structuring trained CNN models on each IoT gateway and the edge server connected to the BS to efficiently utilize the processing capacities of both entities. Specifically, we employ a CNN architecture, i.e., VGG-16 [35], which is utilized to diagnose facility fault type based on the collected dataset on bearing vibration signals [5]. VGG-16 consists of 13 convolutional layers and 3 fully connected layers. A well-trained and complete (full-weight) instance of VGG-16 is deployed on the edge server with sufficient computing resources for task inference, while V ($V \in \mathbb{Z}^+$) pruned instances of the model are deployed on each IGW for more efficient task computation at the price of reduced processing accuracy. Deploying pruned DNN instances on local devices provides multiple benefits, including reduced computational load, lower latency, and energy efficiency by minimizing the reliance on offloading. The availability of models with varying pruning rates allows dynamic adaptability to task requirements, enabling a trade-off between accuracy and delay based on task criticality. This approach enhances scalability and robustness to network variability and supports localized, real-time fault detection, which is crucial for time-sensitive industrial scenarios. These advantages collectively improve resource utilization and system performance in IIoT environments. Each instance on IGW g is characterized by a two-dimensional tuple $\langle A_v, p_v \rangle$, where $v \in \{1, 2, ..., V\}$, A_v is the average percentage of task inference accuracy achieved by pruned CNN instance v with respect to the accuracy of the complete CNN model, and $p_v \in [0, 1)$ is the model pruning rate of instance v. Here,

the L1-norm pruning technique is used [36], which involves removing a fraction (p_v) of weights with the lowest absolute weight magnitude in the complete CNN model. The L1-norm pruning is applied to both convolution and fully-connected layers. Each convolution layer consists of several kernels containing trainable parameters, and the L1 norm is computed for each kernel as the sum of the absolute values of its weights [29]. Then, we choose the p_v fraction of kernels with the smallest L1 norm and set the value of their weights to zero. Similarly, we set the p_v fraction of the weights connecting neurons in two consecutive fully connected layers to zero. As indicated in [29], p_v has a linear relationship with the number of FLoPs in both convolution and fully connected layers, and the number of FLoPs decreases by a factor of p_v after pruning. Therefore, the pruning creates a compressed network with reduced parameters to save inference operations and achieve high inference efficiency.

Following the pruning process, without subsequent retraining, the inference accuracy drops exponentially with the pruning rate [37, 29]. Therefore, each CNN instance is re-trained after pruning to mitigate accuracy loss. The pruning and retraining of the CNN instances happen offline before deployment. After retraining, the relation between accuracy and pruning rate is implicit, which depends on the specific CNN architecture and the dataset used to train the neural network [1, 29]. Therefore, we approximate the specific relation between p_v and A_v based on our employed experimental data, where a polynomial fitting function is used to obtain a closed-form relation [38]. Similar to [38], the least-square fitting error is minimized to get the optimal polynomial degree and fitting coefficients in (1).

$$A_v(p_v) = \sum_{i \in \mathbb{N}} a_i (p_v)^i \tag{1}$$

where each a_i denotes the optimal fitting coefficients. Based on the testing results, we choose the degree of 3 as the order of our polynomial fitting function to achieve a balance between fitting accuracy and computational complexity, as the fitting functions of higher orders exhibit comparable performance, shown in Fig. 3.

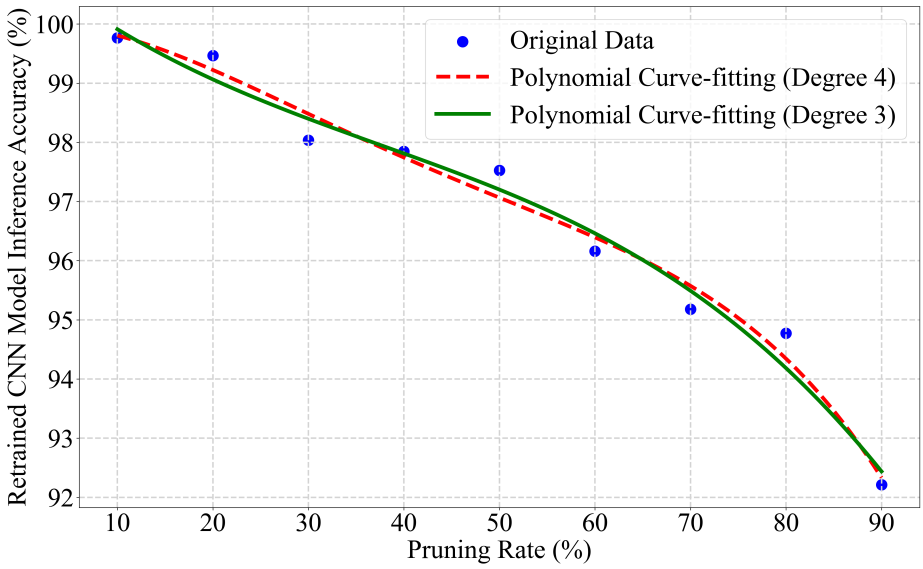


Figure 3: The relation between inference accuracy of a retrained CNN model and its pruning rate.

6.3 Processing Model

Based on the deployed CNN models, at time slot t, each task can be 1) locally processed through one pruned CNN instance at its arriving IGW or 2) offloaded to the edge for processing through the complete CNN model. Denote $o_{m,g}^t \in \{0, 1\}$ as the offloading decision variable for task m at IGW g in time slot t, where $o_{m,g}^t = 0$ indicates local processing and $o_{m,g}^t = 1$ indicates edge processing.

- *Local processing:* We utilize batch task processing to achieve fast inference [39], where tasks arriving at an IGW from different sensors in one time slot are fed into one of the deployed neural network instances as a batch of inputs instead of a single input, and the computations for each input are parallelized across different computation hardware threads. The computation capacity of the IGW is equally divided among the tasks chosen for local processing [1], and we assume the processing units used at the gateways are optimized for parallel processing, which enables efficient processing and fast inference time [39]. We consider that IGW g has a fixed processing capacity, denoted by C_g. The ratio of the computing capacity at IGW g dedicated to locally processing each task at time slot t is calculated as $\frac{1}{\sum_{m=1}^{M_g}(1-o_{m,g}^t)}$. Then, the local processing delay for each task m at IGW g in slot t is calculated as

$$L_{m,g}^t = (1 - o_{m,g}^t) \sum_{j=1}^{M_g} \frac{(1 - p_{v_g^t}) HI (1 - o_{j,g}^t)}{C_g} \tag{2}$$

where $p_{v_g^t}$ is the pruning rate associated with the choice of local inference model instance v_g^t at gateway g in time slot t.

- *Edge processing:* If a task is offloaded to the edge server through wireless communication between an IGW and the BS, it is processed by an uncompressed instance of CNN deployed at the edge. Tasks are transmitted in a sequential manner, and all offloaded tasks in one time slot need to be processed at the edge server within the slot duration to satisfy their individual maximum processing delay requirements. The tasks not satisfying their individual processing time requirements will be discarded. Therefore, to accommodate task offloading from multiple sensors at a time slot, we further partition each time slot t into

mini-slots of length τ [40, 41, 42]. Each mini-slot is indexed by i ($i = 1, 2, ..., q$), where $q \in \mathbb{N}$ and $q = \frac{\Gamma}{\tau}$. The duration of each mini-slot, τ, is set as the time it takes to transmit one task when the network is at its best condition using the whole system bandwidth provided, i.e., $\tau = \frac{H}{R^{\max}}$ where $R^{\max}$ is the maximum transmission rate between an IGW and the BS. Then, at each time slot, a number of mini-slots are allocated to offload each task generated at IGW g in a statistical multiplexing manner, where the synchronization of time slots and mini-slots among gateways is managed by the edge server [43] (Please see subsection 3.2.1 for further explanation). The tasks offloaded from different IGWs are processed at the edge in parallel, where the total computing capacity of the edge server, denoted as C_e, is dynamically divided and allocated among the arriving tasks. $c_g^t \in [0, 1]$ is the decision variable indicating the portion of the computing capacity at the edge dedicated to tasks from gateway g at time slot t. In this regard, the processing delay for task m offloaded from gateway g at time slot t is calculated as

$$E_{m,g}^t = \frac{o_{m,g}^t HI}{C_e(c_g^t + \epsilon)} \tag{3}$$

where $\epsilon \in (0, 1)$ is a small positive number used as a regularization parameter to avoid division by zero when $c_g^t = 0$. By adding ϵ, the division operation remains well-defined and helps to ensure numerical stability and avoid computational errors during optimization [44].

7 Communication Model

We consider an uplink wireless communication system from the IoT gateways to the BS for task offloading where the communication links are assumed non-line-of-sight (NLoS) as the BS is located outside the factory. Similar to [1], the uplink NLoS channel gain from gateway g to the BS, denoted by ψ_g^t at time slot t, is modeled as a three-state discrete-time Markov chain, where the channel gain within one time slot is assumed stable. The three channel states are named *Good*, *Normal*, and *Bad*, abbreviated as *G*, *N*, and *B*, respectively, indicating three categories of channel conditions ranging from a good state to a poor state. The value of channel gain at each state is acquired based on real-time measurements, and the one-step channel state transition probability matrix is given by [1]

$$P = \begin{bmatrix} P_{BB} & P_{BN} & P_{BG} \\ P_{NB} & P_{NN} & P_{NG} \\ P_{GB} & P_{GN} & P_{GG} \end{bmatrix} \tag{4}$$

where P_{xy} indicates the transition probability from state x to state y between consecutive time slots. Then, the uplink transmission rate for IGW g at time slot t is calculated as

$$R_g^t = \frac{M_g}{N} W \log_2 \left(1 + \frac{N z_g \psi_g^t}{M_g N_0 W}\right) \tag{5}$$

where N is the total number of sensors in the system, W is the total configured system bandwidth, z_g is the transmission power configured at IGW g for uplink task offloading, which is assumed to be fixed during the network operation stage, and N_0 is the Gaussian white noise power spectrum density.

As seen from (5), the total system bandwidth W is allocated among IGWs proportional to the number of sensors, M_g, associated with IGW g [1]. Considering that industrial sensors and gateways are relatively stationary with limited processing and energy capacities, the bandwidth re-allocation among industrial IoT gateways is typically conducted at a slow frequency (e.g., in the order of hours) to reduce the signaling overhead incurred in the re-allocation process [15]. Thus, we assume the bandwidth allocated to each gateway remains unchanged during each network operation stage [1]. Based on the above analysis, the offloading delay for task m arriving from IGW g at time slot t is given by

$$\Omega_{m,g}^t = \frac{o_{m,g}^t H}{R_g^t}. \tag{6}$$

Considering task processing results are usually small in size, the latency of transmitting the processing results back from the edge server to each IGW can be negligible [5, 45].

In IIoT scenarios where wireless communication resources for task offloading are often limited, task transmission latency can be longer than task inter-arrival time at an IGW [45], leading to task queuing before being offloaded. In the following, we present our modeling of task queuing at each IGW.

7.1 Queuing Model

At each IGW, a transmission queue is established to buffer tasks for offloading, which is designated to accommodate tasks arriving from all sensors connected to the IGW, as shown in Fig. 4. The tasks at each queue are prioritized based on their criticality levels, and the tasks with higher criticality levels are preemptively queued for trans-

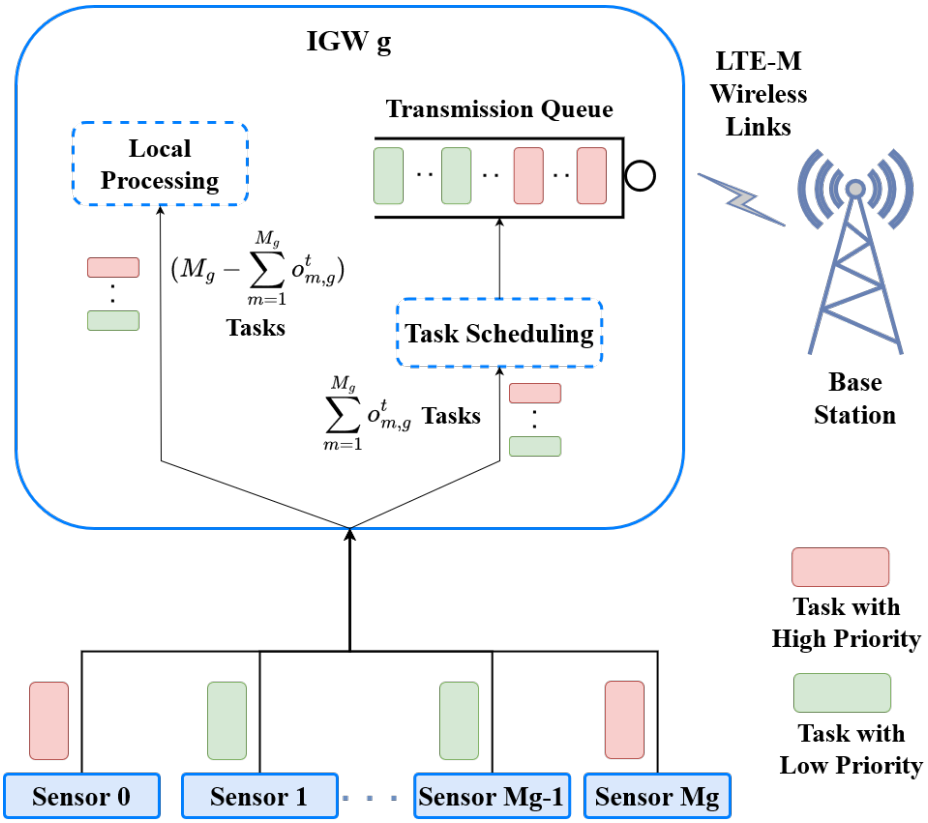


Figure 4: Local processing and transmission scheduling process at each gateway.

mission over those with lower levels. If several tasks have the same criticality level, the queuing among them will be determined randomly.

Based on the preceding discussions, the queuing delay for each task depends on the prioritized ordering of tasks in the transmission queue of a gateway. Denote δ_g^t ($\delta_g^t \in \{1, 2, ..., q\}$) as the number of mini-slots allocated to transmit one task from IGW g at time slot t, where $\delta_g^t = \lceil \frac{R^{\max}}{R_g^t} \rceil$. Note that δ_g^t is the same for all the tasks to be offloaded from gateway g at time slot t, though it may vary across different time slots. The time-varying nature of δ_g^t and the ordering of tasks in the transmission queues highlights the necessity of employing statistical multiplexing. Tasks unable to be accommodated for transmission or whose delay requirements are violated will

be dropped from the queue. Therefore, the transmission queuing delay for task m at IGW g is a summation of the transmission delay of the tasks queued before task m, given by

$$B^t_{m,g} = o^t_{m,g}\delta^t_g\tau\left[s^t_{m,g} + \frac{(b^t_{m,g}-1)}{2}\right] \tag{7}$$

where $s^t_{m,g}$ is the number of tasks with a higher criticality level than task m and $b^t_{m,g}$ is the number of tasks with the same criticality level as task m queued for offloading at IGW g, calculated, respectively, as

$$s^t_{m,g} = \sum_{j=1}^{M_g} h(j,m,g,t)o^t_{j,g} \tag{8}$$

and

$$b^t_{m,g} = \sum_{j=1}^{M_g} \mu(j,m,g,t)o^t_{j,g}. \tag{9}$$

In (8) and (9), $h(j,m,g,t)$ and $\mu(j,m,g,t)$ are two helper functions defined as

$$h(j,m,g,t) = \begin{cases} 1, & \text{if } k^t_{j,g} > k^t_{m,g} \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

and

$$\mu(j,m,g,t) = \begin{cases} 1, & \text{if } k^t_{m,g} = k^t_{j,g} \text{ and } j \neq m \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

After being transmitted to the BS, the tasks will arrive at the edge server. In the process of task offloading, the task processing delay at the edge server is much smaller than the task transmission delay [45]. Therefore, no queuing delay is considered for task processing at the edge server.

Problem Formulation

We present our research problem formulation on joint task offloading, pruned model selection, and computing resource allocation. Our objective is to maximize the long-term utilization of system radio bandwidth and computing resources while meeting the diverse inference accuracy and delay requirements of individual tasks, which change over time with the task criticality levels. We balance the trade-off between local task processing and edge processing by taking into consideration the resource utilization, the task processing delay, and task processing accuracy. Next, we determine, in every time slot t, the local computing resource utilization at any IGW g, the edge computing resource utilization, and the bandwidth utilization for processing tasks offloaded from IGW g, respectively.

8 Local Computing Resource Utilization

We define the local computing resource utilization of each IGW as the proportion of time spent in one slot processing the tasks at the IGW. As illustrated in Fig. 5(a), at each IGW, the tasks not offloaded are processed in parallel using batch processing, where the entire computing resources of the IGW are divided equally among the tasks (see section 3.3 for further details.) Consequently, by referring to (2), the local computing resource utilization for processing the tasks at gateway g in time slot t is calculated as the local processing delay of each task over the time slot duration Γ, given by

$$u_{l,g}^{t} = \sum_{m=1}^{M_g} \frac{(1 - p_{v_g^t}) H I (1 - o_{m,g}^{t})}{C_g \Gamma}. \tag{12}$$

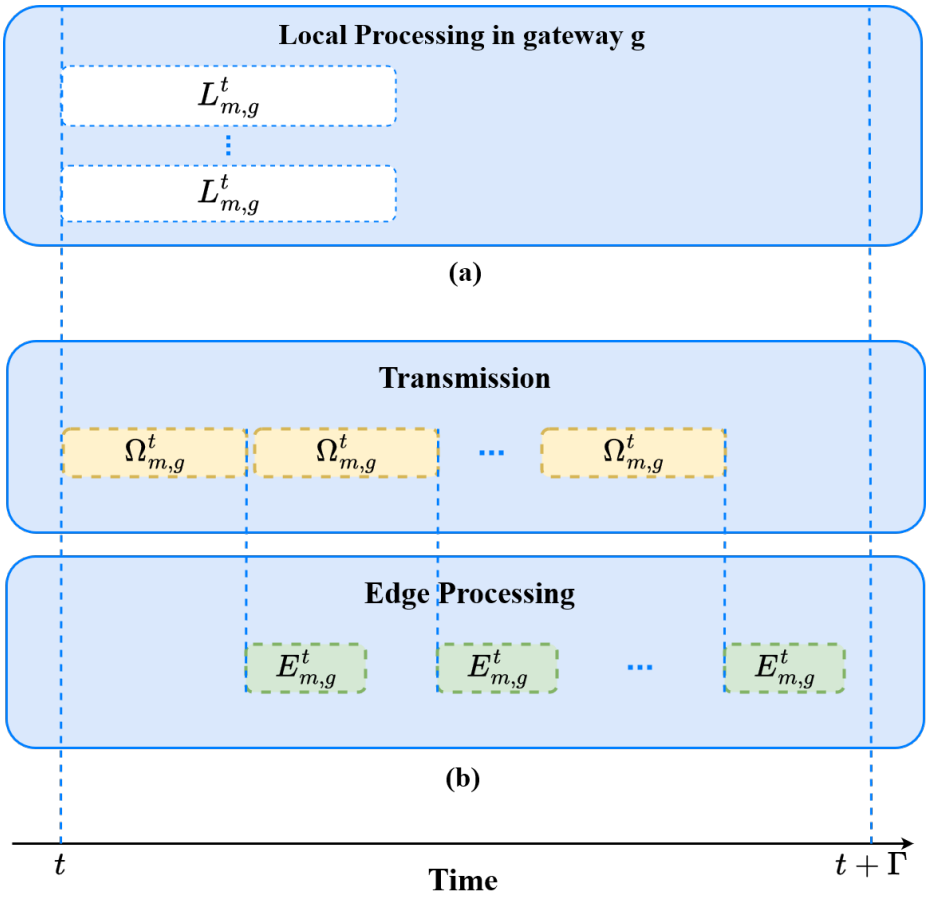


Figure 5: An illustrative example of (a) local computing resource utilization and (b) bandwidth and edge computing resource utilization within one time slot for an IGW.

9 System Bandwidth Utilization

At each IGW, the tasks chosen to be offloaded first enter the transmission queue and are then transmitted sequentially using the portion of the system bandwidth allocated to that IGW, as illustrated in Fig. 4. Specifically, at the beginning of any time slot t, the transmission of a task starts, followed by a sequence of tasks to be transmitted in the slot, as shown in Fig. 5(b). If a task cannot be transmitted within the duration of a time slot, it gets discarded. The overall utilization of bandwidth for IGW g at

time slot t is calculated as the duration of time used to transmit the offloaded tasks within the slot, given by

$$u_{b,g}^{t} = \sum_{m=1}^{M_g} \frac{\Omega_{m,g}^{t}}{\Gamma}. \tag{13}$$

10 Edge Computing Resource Utilization

After transmission, the offloaded tasks are processed at the edge server, where the total computing resources are allocated to process the tasks from different gateways. The tasks offloaded from any IGW g are sequentially processed according to the order of their arrivals. Therefore, the utilization of the computing resources on the edge server dedicated to IGW g is the summation of the edge processing delay for all the tasks offloaded from IGW g divided by the length of the time slot Γ, given by

$$u_{e,g}^{t} = \sum_{m=1}^{M_g} \frac{E_{m,g}^{t}}{\Gamma}. \tag{14}$$

Based on the preceding analysis, the summation of the local processing, communication, and edge processing utilization for all G IGWs at time slot t is represented as

$$u^{t} = \sum_{g=1}^{G} u_{l,g}^{t} + u_{e,g}^{t} + u_{b,g}^{t}. \tag{15}$$

Note that the presence of idle time in the local processing units, as observed in Figure 4.1 (a), is inherently accounted for in formulating the objective function, which aims to maximize overall resource utilization. Specifically, the local computing resource utilization at each gateway g in a given time slot t is defined as the proportion of total computing resources actively used for task processing relative to the available capacity. Therefore, idle time is effectively reduced, and thus, the aggregated utiliza-

tion, which sums local, bandwidth, and edge resource utilization, is also diminished. This mechanism ensures that the optimization process inherently penalizes idle local resources by prioritizing configurations (e.g., offloading decisions, pruned model selection) that better utilize the available computational capacity.

11 Proposed Joint Optimization Framework

Conducting more local processing with pruned learning models can increase task processing efficiency with a lower delay but a reduced processing accuracy, while a higher task inference accuracy and communication resource utilization can be achieved through edge processing at the cost of local resource underutilization and a longer latency. Therefore, the main research issue is to determine the optimal task offloading, pruned local model selection, and computing resource allocation policies such that the overall communication and computing resource utilization can be maximized in the long run with task processing accuracy and delay requirements satisfied. To this end, our problem is presented as a stochastic optimization formulation, given in (P1), where η is a large positive number, $T^{\min}$ and $T^{\max}$ represent the minimum and maximum task processing delay requirements, respectively, among all tasks, corresponding to the highest and lowest criticality levels of tasks, $A^{\max}$ and $A^{\min}$ denote the maximum and minimum task inference accuracy requirements, respectively, among all tasks, and $D^t_{m,g}$ represents the end-to-end delay for task m at gateway g in time slot t, calculated as

$$D^t_{m,g} = L^t_{m,g} + E^t_{m,g} + \Omega^t_{m,g} + B^t_{m,g}. \tag{16}$$

$$\textbf{(P1)}: \max_{o^t_{m,g}, c^t_g, v^t_g} \mathbb{E}\left\{\frac{1}{\eta}\sum_{t=1}^{\eta} u^t\right\}$$

$$s.t. \begin{cases} 0 \leq D^t_{m,g} \leq \left[T^{\max} - k^t_{m,g}(T^{\max} - T^{\min})\right], \forall m, g & \text{(17a)} \\ \left[A^{\min} + (A^{\max} - A^{\min})k^t_{m,g}\right](1 - o^t_{m,g}) \leq A_{v^t_g}(p_{v^t_g})(1 - o^t_{m,g}) \leq 1, \forall m, g & \text{(17b)} \\ \sum_{g=1}^{G}\left[c^t_g \prod_{m=1}^{M_g}(1 - o^t_{m,g})\right] = 0 & \text{(17c)} \\ \sum_{g=1}^{G} c^t_g = 1 & \text{(17d)} \\ c^t_g \in [0,1], \forall g & \text{(17e)} \\ E^t_{m,g} \leq \Omega^t_{m,g}, \forall m, g & \text{(17f)} \\ o^t_{m,g} \in \{0,1\}, \forall m, g & \text{(17g)} \\ p_{v^t_g} \in (0,1], \forall g & \text{(17h)} \end{cases}$$

The objective of (P1) is to maximize the stochastic average of the aggregated bandwidth and computing resource utilization for all the IGWs and the edge server, where $\mathbb{E}$ indicates the expectation operator, and the optimization variables are task offloading decision $o^t_{m,g}$, computing resource allocation decision c^t_g, and pruned local inference model selection v^t_g. Constraint (16a) indicates that the end-to-end delay of task m generated from gateway g cannot exceed the upper bound varying between $T^{\min}$ and $T^{\max}$ when $k^t_{m,g}$ takes values between 0 and 1. Similarly, constraint (16b) indicates that the inference accuracy of the chosen pruned local processing model v^t_g is limited by a lower bound changing in between $A^{\min}$ and $A^{\max}$ as $k^t_{m,g}$ varies. Constraint (16c) indicates that no edge computing capacity is dedicated to the gateways that do not have any offloading tasks, and constraint (16d) shows that the edge computing capacity is divided and allocated among the gateways for processing the offloaded

tasks. Constraint (16f) indicates that the edge processing delay of an offloaded task is smaller than or equal to its transmission delay to ensure task processing without queuing delay at the edge server. Any task offloaded from gateway g violating (16a), (16b), and/or (16f) is dropped from the edge server.

12 Problem Transformation

The problem (P1) is formulated in a centralized way where the edge server acts as the agent to make task offloading, local pruned model selection, and computing resource allocation decisions. Note that the uplink channel state information acquired from the model presented in (4) is assumed to be available to the edge server at the beginning of each time slot [1]. For the centralized agent to make decisions, each gateway updates with the edge server the task criticality levels at each time slot. The time used to transmit each task criticality level is usually small in size and is thus neglected [5]. In the proposed system, to capture the network state transitions and model the relation between states and policies, we describe the problem (P1) as a Markov reward process (MRP) formulation where the state transitions are independent of the actions taken [30]. The MRP is particularly valuable for simplifying and structuring complex decision-making scenarios that involve sequential decisions over time. It accounts for both immediate and future consequences, explicitly models the stochastic nature of environments, and incorporates uncertainty into the formulation. This approach provides a robust framework for understanding and optimizing decision-making processes in dynamic and uncertain contexts. The MRP formulation

is represented by a four-dimensional tuple at time slot t, which includes a set of network states $\mathcal{S}^t$, a set of actions $\mathcal{A}^t$, state transition probabilities, $\mathcal{P}(\mathcal{S}^{t+1}|\mathcal{S}^t)$, and a reward function, $\mathcal{R}(\mathcal{S}^t, \mathcal{A}^t)$, defined on states and actions. Specifically, to capture the dynamics of the system, $\mathcal{S}^t$ is designed to include the uplink wireless channel gains from all gateways to the base station and the criticality levels of all generated tasks at time slot t, denoted by

$$\mathcal{S}^t = \{\psi_g^t \mid \forall g\} \cup \{k_{m,g}^t \mid \forall m, g\}. \tag{1}$$

Furthermore, the action set $\mathcal{A}^t$ comprises the decision variables for task offloading, model pruning, and edge computing resource allocation, formally defined as

$$\mathcal{A}^t = \{o_{m,g}^t \mid \forall m, g\} \cup \{v_g^t \mid \forall g\} \cup \{c_g^t \mid \forall g\}. \tag{2}$$

The state transitions from t to $(t+1)$ include the updates on channel gains for all gateways and the criticality levels of the generated tasks, and thus, the state transition probability is given by

$$\begin{aligned} \mathcal{P}(\mathcal{S}^{t+1}|\ \mathcal{S}^t) &= \prod_{g=1}^{G} \mathcal{P}(\psi_g^{t+1}|\ \psi_g^t) \cdot \prod_{g=1}^{G} \prod_{m=1}^{M_g} \mathcal{P}(k_{m,g}^{t+1}|\ k_{m,g}^t) \\ &= \prod_{g=1}^{G} \mathcal{P}(\psi_g^{t+1}|\ \psi_g^t) \cdot \prod_{g=1}^{G} \prod_{m=1}^{M_g} \mathcal{P}(k_{m,g}^{t+1}). \end{aligned} \tag{3}$$

In (20), the first equality holds due to the independence of the channel gain values among different gateways and the independence of the channel gain values from the task criticality levels, and the second equality holds due to the criticality levels of tasks at one time slot are independent of the levels in the previous time slots, as they vary randomly over consecutive time slots [5]. Note that channel gain values evolve according to the transition probability matrix presented in (4). Given the balanced

sample distribution in the dataset, we assume that different criticality levels occur with equal probability. Consequently, at each time slot, the criticality level of each task is determined by sampling from a uniform distribution. This approach ensures a fair representation of all criticality levels throughout the analysis.

Solving the MRP problem is to determine a set of optimal policies (i.e., probabilities of choosing actions given network states), denoted by $\pi^*(\mathcal{A}|\mathcal{S})$, that maximizes the accumulated system reward over time, where $\mathcal{S}$ and $\mathcal{A}$ represent steady states and actions as t approaches infinity. Accordingly, a reward function is designed by considering the overall network resource utilization and the satisfaction of E2E delay and inference accuracy requirements at each time slot. Hence, we present the reward function as

$$r^t = u^t - \left[\frac{\sum_{g=1}^{G}\sum_{m=1}^{M_g}(1-\beta_{m,g}^t)}{N}\right] J \tag{4}$$

where $\beta_{m,g}^t$ is an indicator function, defined as

$$\beta_{m,g}^t = \begin{cases} 1, & \text{if } (16a), (16b), \text{and } (16f) \text{ hold} \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

In (21), J is a large positive number and is multiplied by the ratio of dropped tasks in time slot t. If the requirements in (16a), (16b), and/or (16f) are not satisfied, then a negative reward is received as a penalty, which changes according to the number of dropped tasks, and if all the requirements are satisfied, then the reward reflects the overall system resource utilization.

In (P1), the state and action dimensions are calculated as $N + G$ and $N + 2G$, respectively, where the action space contains a set of continuous variables c_g^t ranging between 0 and 1 and two sets of discrete variables. The action space size can be

estimated as $2^N V^G Z^G$, where Z is a large positive number estimating the number of values c_g^t can take when discretized, and the state space size of the problem is calculated as $\Psi^G K^N$, where Ψ and K are the numbers of configurable channel gain values and criticality levels, respectively. As the dimensions increase with the number of sensors N and the number of gateways G, both the state and action spaces grow exponentially, resulting in higher computational complexity. The conventional algorithms, such as Epsilon-Greedy [46] and Upper Confidence Bound (UCB) [47], used for solving MRP problems where the state transition probabilities are independent of the actions taken under the current states, may not be efficient in solving a complex problem with large action and state spaces where an optimal solution needs to be obtained at the beginning of each time slot[48].

13 SAC-Based Solution Design

Solving the transformed MRP problem with large state and action spaces and dynamic constraints necessitates the design of an advanced algorithm [49]. We propose to use a DRL-based approach to approximate the functional relation between each state-action pair and the corresponding reward using DNNs [10]. Particularly, DRL learns an effective policy through trial and error by interacting with the network environment [50]. In our formulated MRP problem, the task offloading and local pruned model selection are discrete decision variables, while the edge computing resource allocation decisions are continuous. Furthermore, since the state transition probability is independent of the actions taken, the state space needs to be thoroughly and efficiently explored to obtain optimal actions. To address these challenges, we build our algorithms based on SAC, which is an actor-critic learning framework used for accommodating continuous RL actions and offers the following advantages [51].

- **Enhanced Exploration:** SAC employs a stochastic policy, sampling actions according to a learned probability distribution, which inherently encourages exploration. The stochastic policy helps the learning agent thoroughly explore the high-dimensional action space with complex network dynamics, in contrast to the deterministic policy that relies on added noise for exploration.

- **Improved Sample Efficiency:** SAC is sample-efficient, achieving high performance with reduced environment interactions. This efficiency is attributed

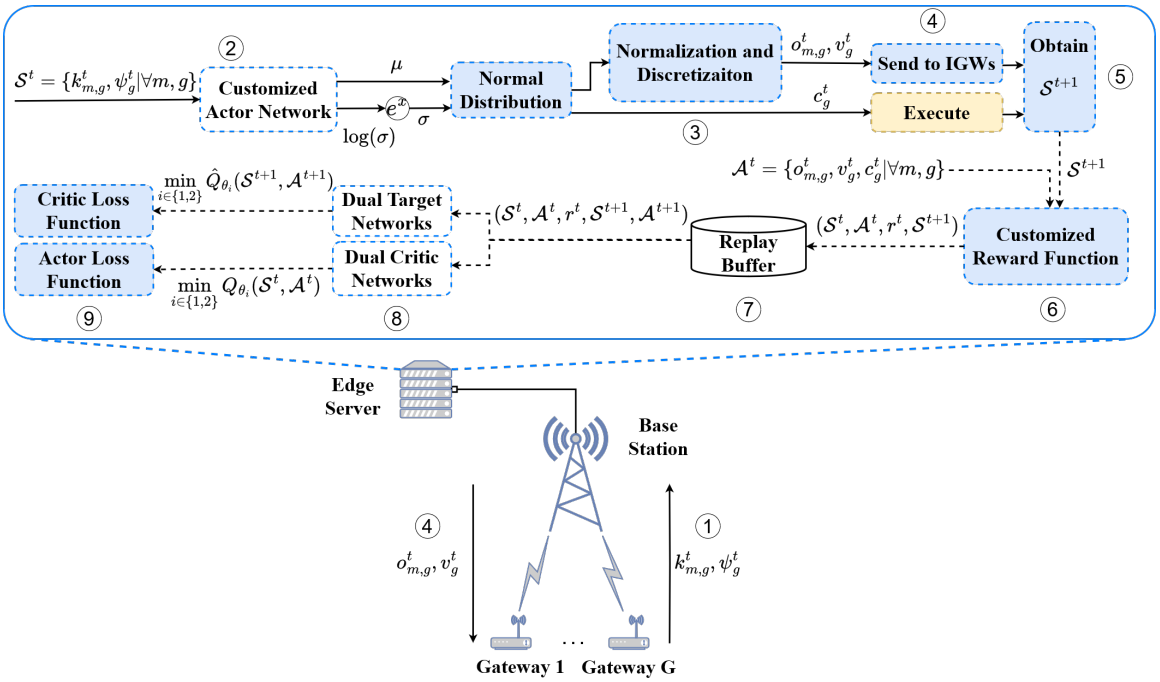


Figure 6: The proposed SAC-based solution framework.

to the improved exploration and stability provided by entropy regularization and dual Q-learning (i.e., using two critic networks to mitigate overestimation bias in estimating the value function).

Based on the aforementioned advantages, we propose an SAC-based joint task offloading, DNN pruning, and computing resource allocation (JOPA), which includes the design of the following three main functional components.

13.1 Customized Actor Network for Hybrid Actions

JOPA is built on the SAC algorithmic framework, tailored to address the formulated MRP with a hybrid action space. Different from the conventional SAC, which is

designed for continuous actions, our approach adapts SAC to handle a combination of continuous and discrete actions. We customize the actor network π_ϕ to generate decisions for task offloading, pruned model selection, and edge computing resource allocation. The actor network first produces mean μ and log standard deviation $\log(\sigma)$. Then, the log standard deviation is exponentiated to get standard deviation σ for sampling continuous actions from a normal distribution $\mathcal{N}(\mu, \sigma^2)$. Using log standard deviation ensures that the standard deviation obtained after exponentiation is always positive. Directly using standard deviation could lead to potential instability if small or negative values are produced, especially when training with gradients [31]. The samples are then processed to obtain the hybrid decision variables: 1) Binary task offloading decisions associated with N sensors are obtained by normalizing the N action variables to the range of $[0, 1]$ using the Sigmoid function and the rounding function consecutively; 2) The G categorical actions for pruned model selection corresponding to G gateways are similarly normalized and rounded to the range of $[0, V-1]$; 3) The G continuous action variables corresponding to the edge computing resource allocation decisions for tasks arrived from G gateways, respectively, are computed using the Softmax function, ensuring adherence to constraint (16d) in (P1).

13.2 Dual Critic Networks for Robust Policy Evaluation

In SAC, the Q-value of each state-action pair is estimated based on the maximum value over the set of possible actions. However, an estimated Q-value can be overly optimistic because the algorithm often selects the action with the highest Q-value estimate, which could be erroneously high due to noise or errors in the approxim-

ation. By incorporating two critic networks (Q_{θ_1} and Q_{θ_2}), our algorithm reduces overestimation bias, a key consideration when dealing with mixed actions for a more reliable assessment of the expected return (long-term accumulated reward). By using the minimum of the two estimates, SAC becomes more conservative in its Q-value approximation, reducing the chance of overestimation. The critic networks process state $\mathcal{S}^t$ and action $\mathcal{A}^t$, outputting an estimated return value for the state-action pair. This dual-network design enhances the robustness of our algorithm, preventing suboptimal convergence and ensuring that the solution space is thoroughly explored.

13.3 Stabilized Target Networks for Reliable Training

When updating Q-values, there is a risk of instability since the target values are based on the current frequently updated Q-values. This leads to high variance in target estimates, especially in the early stages of training when the agent is exploring. High variance can cause the learning process to become unstable or even diverge. To stabilize training, we employ two target networks, denoted by $\hat{\theta}_1$ and $\hat{\theta}_2$, to reduce variance during updates and minimize the chance of overestimation. Our approach initializes these networks as copies of the critic networks, ensuring consistency throughout training, given by:

$$\hat{\theta}_i \leftarrow \theta_i, i \in \{1, 2\}. \tag{6}$$

To mitigate the high variance issue and stabilize training, the algorithm uses the minimum of the two target values to update the critic networks, which reduces bias in the critic function update. Furthermore, the target networks are updated using a soft update rule, changing more slowly than the critic networks. This helps smooth

the target values and avoid instability caused by rapidly changing Q-values (See subsection 5.1.4 for more details).

13.4 Algorithm Design

Fig. 6 shows the key steps (Step 1 to Step 9) in our proposed SAC-based JOPA algorithmic framework. As shown in Algorithm 1, the JOPA is trained in a time-slotted manner. First, the learning agent at the edge server obtains experience by interacting with the environment. At time slot t, based on the current network state $\mathcal{S}^t$, the task offloading, pruned model selection, and edge computing resource allocation actions are determined using the actor network (Steps 1-3). Then, the edge computing resource allocation actions are executed on the edge server, while the pruned model selection and task offloading actions are transmitted to and are executed at the IGWs (Step 4). The corresponding reward r^t is calculated, and the next state $\mathcal{S}^{t+1}$ is observed from the environment to create the state transition tuple $(\mathcal{S}^t, \mathcal{A}^t, r^t, \mathcal{S}^{t+1})$ stored in the experience replay memory $\mathcal{D}$ for training the actor and critic networks (Steps 5-7). Once enough tuples are collected in the experience replay, a batch of transition tuples is randomly sampled from the experience replay memory to train the actor and critic networks, and update the target networks accordingly (Steps 8 and 9). We first update the critic networks by minimizing the mean squared loss function defined as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(\mathcal{S}^t, \mathcal{A}^t, r^t, \mathcal{S}^{t+1}) \sim \mathcal{D}} \left[Q_{\theta_i}(\mathcal{S}^t, \mathcal{A}^t) - y \right]^2, i \in \{1, 2\} \tag{7}$$

where y is the target value calculated using the target networks, given by

$$y = r^t + \gamma \min_{i \in \{1,2\}} \hat{Q}_{\theta_i}(\mathcal{S}^{t+1}, \mathcal{A}^{t+1}). \tag{8}$$

In (25), γ is the discount factor. When γ is closer to zero, more recent rewards are considered, undermining future rewards, while the agent values future rewards more when γ is set closer to 1. Next, we update the actor network by minimizing the loss calculated as:

$$\mathcal{L}(\phi) = \mathbb{E}_{\mathcal{S}^t \sim \mathcal{D}, \mathcal{A}^t \sim \pi_\phi} \left[\alpha \log \pi_\phi(\mathcal{A}^t | \mathcal{S}^t) - \min_{i \in \{1,2\}} Q_{\theta_i}(\mathcal{S}^t, \mathcal{A}^t) \right] \tag{9}$$

where $\alpha \log \pi_\phi(\mathcal{A}^t | \mathcal{S}^t)$ is the entropy term incorporated into the policy objective to ensure that the policy not only aims to maximize the expected reward but also maintains high entropy (a measure of randomness or unpredictability in the policy's action distribution). This approach helps prevent premature convergence to sub-optimal policies by encouraging continuous exploration of the action space. The entropy term is weighted by a temperature parameter α, which balances the trade-off between exploration and exploitation where higher α results in more stochasticity in the policy's action distribution and, therefore, more exploration. Note that the rounding function is not differentiable, and therefore, the actions before rounding are used to calculate the loss. The final step is to softly update the target networks by

$$\hat{\theta}_i \leftarrow \zeta \theta_i + (1 - \zeta) \hat{\theta}_i, \quad i \in \{1, 2\}, \tag{10}$$

where ζ is the learning rate parameter for the target networks. The detailed process of JOPA is listed in Algorithm 1, showing how the edge server as the learning agent interacts with the IGWs to conduct SAC-based training. After the training process is completed, the trained model is implemented in the edge server to execute the optimal online actions based on real-time network states.

The time complexity analysis of Algorithm 1 is provided in the following. At each time slot t, the state $\mathcal{S}^t$ is fed to the actor network, and the actions are obtained with

Algorithm 1: SAC-based algorithm for joint task offloading, DNN pruned model selection, and edge computing resource allocation (JOPA)

Initialize: network configuration and service parameters

Initialize: replay buffer $\mathcal{D}$, actor, critic, and target networks

for *time slot* $t \in \{1, 2, ..., \eta\}$ **do**

  Observe state $\mathcal{S}^t$ and obtain mean and log standard deviation from the actor network;

  Exponentiate log standard deviation to derive standard deviation σ;

  Using activation functions, obtain the actions $\mathcal{A}^t = \{c_g^t, o_{m,g}^t, v_g^t | \forall m, g\}$;

  for *each IGW* $g \in \{1, 2, ..., G\}$ **do**

    Execute c_g^t on the edge server and send v_g^t to IGW g for execution;

    for *each task* $m \in \{1, 2, ..., M_g\}$ **do**

      Send $o_{m,g}^t$ to IGW g for execution;

  Observe reward r^t and the next state $\mathcal{S}^{t+1}$;

  Store transition $(\mathcal{S}^t, \mathcal{A}^t, r^t, \mathcal{S}^{t+1})$ in $\mathcal{D}$;

  if *size of* $\mathcal{D} \geq$ *batch size* **then**

    Sample a batch of transitions from $\mathcal{D}$;

    Compute target values using (25);

    Update critic networks by minimizing the loss in (24);

    Update the actor network by minimizing the policy loss defined in (26); Soft update target networks using (27);

the time complexity of $O(N + G)$. Based on the number of actions, the execution of the actions take $O(N + G)$ time. Considering that the actor network has an input

size of $(N+G)$ and an output size of $(N+2G)$, updating happens in $O(N+G)$ time. Likewise, training the critic and target networks happens in $O(N+G)$ time since the input size is $(2N+2G)$ and the output size is 1. As $N = \sum_{g=1}^{G} M_g$ and it takes η time slots to solve the problem, therefore the time complexity for Algorithm 1 in terms of its input variables is $O(\eta G M_g^{\max})$, where $M_g^{\max} = \max_g \{M_g\}$.

Simulation Results

Computer simulations are conducted to demonstrate the performance of the proposed JOPA algorithm and the advantages over existing schemes.

14 Simulation Setup

We consider a smart factory environment in our simulation, featuring 5 lanes of washing brushes, each connected to one IGW. To evaluate the scalability of the proposed scheme, we consider five scenarios with 100, 125, 150, 175, and 200 vibration sensors, where the sensors installed on the washing brushes in each lane detect vibration signals, which are then digitized. The digitized data can be processed either locally on the connected IGW or be offloaded to an edge server connected to a BS outside the factory for further processing. Each IGW connects to the BS through the LTE Cat-M2 technology, with the uplink transmission power set as 1 W and the computing power providing 768 GFLoPS/s processing rate. The total available spectrum bandwidth for uplink transmissions from the IGWs to the BS is set as 5MHz [2, 15]. The three channel conditions **Good (G)**, **Normal (N)**, and **Bad (B)** correspond to channel gains 6×10^{-13}, 4×10^{-13}, and 2×10^{-13}, respectively, with the transition probability matrix set as [1]:

$$P = \begin{bmatrix} P_{BB} & P_{BN} & P_{BG} \\ P_{NB} & P_{NN} & P_{NG} \\ P_{GB} & P_{GN} & P_{GG} \end{bmatrix} = \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.7 & 0.3 \end{bmatrix}. \tag{11}$$

The duration of a time slot in the simulation is set to 1s, and, consequently, $R^{\max}$ is calculated as 5.68 Mbit/s, with q and τ values being 347 and 0.0028, respectively. The

bearing vibration signal is collected at a 4 kHz sampling rate with 16-bit quantization. The task data is the digitized 1s vibration signal sampled during the previous time slot, so the task data size is the data volume of a 1s signal [2, 15] , which is calculated as the product of the raw sampling rate and the quantization bits of the signal, resulting in a task data size of 32 kb. The edge server connected to the BS is simulated by an NVIDIA RTX 3070 GPU with 20.31 TFLoPs/s computing power for parallel edge processing. To perform fault diagnosis tasks, the edge server is equipped with a full-weight, well-trained VGG-16 model [35], and each IGW is equipped with 6 pruned models. The corresponding accuracy and pruning rate for each model can be found in Table 2, and the optimal polynomial fitting function is given by

$$A_v(p_v) = -1.729 \times 10^{-5}(p_v)^3 + 0.001953(p_v)^2 - 0.1313(p_v) + 101. \tag{12}$$

Considering that VGG-16 requires 10 GFLOPS for inference, the task processing intensity is 3.12×10^6. According to (3), a task must be dropped if $o^t_{m,g} = 1$ and c^t_g is zero. Therefore, we set ϵ such that when c^t_g is zero, the edge processing delay is equal to the length of a time slot, resulting in a value of 5×10^{-13} (See section 3.2 for more details.)

Algorithm 1 is implemented using Python 3.10 with PyTorch 2.1.0 and CUDA for parallel DNN training on GPUs. The DNNs forming the SAC-based module deployed on the edge server consist of 3 fully-connected hidden layers, each with 1024 neurons. Based on the formulation in (P1), the accuracy lower-bound and delay upper-bound associated with each criticality level are detailed in Table 3. Other important simulation parameters and training hyper-parameters are summarized in Table 4. The performance of the proposed scheme is evaluated and compared with a

Table 2: Inference accuracy and pruning rate of pruned models deployed on IGWs.

Pruning Rate	**Inference Accuracy**
0.1	99.76
0.3	98.03
0.5	97.52
0.7	95.17
0.8	94.77
0.9	92.21

version without pruned model selection for local processing ($V = 1$) and an accuracy-guaranteed collaborative DNN inference scheme [1]. The performance evaluation and comparison focus on network resource utilization, satisfaction with time-varying QoS requirements (delay and accuracy), and task-dropping ratio.

15 Performance Evaluation

To evaluate the convergence of JOPA in solving (P1), we train the DNNs for the two instances of the proposed scheme, i.e., $V = 6$ and $V = 1$, across the five scenarios mentioned in section 6.1. We utilize the Optuna package for hyperparameter tuning, running 100 experiments with different hyperparameters for each scenario to obtain the best results. In each experiment, we train the DNNs for 10,000 time slots (i.e., $\eta = 10,000$) in an online manner by interacting with the environment. The training results for both instances of the proposed scheme are illustrated in Fig 7. The training results show that the rewards for both instances fluctuate at the beginning

Table 3: Accuracy and delay requirements of tasks associated with their criticality levels

Criticality level	**Accuracy lower-bound**	**Delay upper-bound**
0.0	0.920 ($A^{\min}$)	1.0s ($T^{\max}$)
0.2	0.934	0.82s
0.4	0.948	0.64s
0.6	0.962	0.46s
0.8	0.976	0.28s
1.0	0.990 ($A^{\max}$)	0.1s ($T^{\min}$)

of the training due to exploration. However, they gradually stabilize as the training progresses. It is also evident that as the total number of sensors increases, the time to converge for both scheme instances also increases. When $V = 1$, the algorithm generally converges more quickly than the case of $V = 6$ because the action space size is significantly reduced by a factor of V^G. In comparison, when $V = 6$, the algorithm achieves a higher reward, indicating a higher overall network resource utilization and a lower task-dropping rate at the end of the training, which benefits from the flexibility of having the pruned model selection for local processing.

To further validate the performance gap of JOPA, we compare the rewards of the proposed scheme after convergence with that of an optimal solution obtained by exhaustive search where the computing resource allocation action space is discretized into 20 values. All combinations of action variables are tested to obtain the optimal solution, and the action achieving the highest reward is selected as optimal. We

Table 4: Simulation parameters and training hyper-parameters [1, 2]

Parameter	**Value**
Noise power spectrum density (N_0) [2]	$10^{-18}W/Hz$
Temperature parameter (α)	10^{-5}
Maximum delay requirement ($T^{\max}$)	1s
Target learning rate (ζ)	10^{-5}
Minimum delay requirement ($T^{\min}$)	0.1s
Discounting factor (γ)	0.99
Maximum accuracy requirement ($A^{\max}$)	0.99
Batch size	512
Minimum accuracy requirement ($A^{\min}$)	0.92
Penalty coefficient (J)	99
Replay buffer size	10,000

consider a light-loaded network scenario with 3 gateways and 6 sensors for tractability to obtain the optimal solution within a reasonable time. As shown in Fig. 8, JOPA achieves a near-optimal solution with a small performance gap. The reward fluctuations of JOPA are due to the sampled actions from a continuous space for the computing resource allocation.

16 Performance Comparison

We compare the performance of JOPA with two benchmark schemes: 1) the proposed scheme without pruned model selection (JOPAV1) and 2) an accuracy-guaranteed

delay minimization (AGDM) scheme with static task requirements [1]. Both AGDM and JOPAV1 utilize a mid-pruned model, characterized by an accuracy of 95.17% and a pruning rate of 0.7, for local processing. In the AGDM scheme, the task criticality levels for each IGW are initialized randomly but remain unchanged.

We compare the performance of the three schemes in terms of network resource utilization, adaptability to network load conditions, dynamic delay/accuracy requirements, and trade-off between QoS satisfaction and resource utilization.

- **Network resource utilization:** Figs. 9(a) and 9(b) show that the JOPA consistently achieves the highest mean bandwidth and edge computing resource utilization, followed by JOPAV1 and AGDM. This is because the JOPA has consistently higher task offloading rates, shown in Fig. 9(e), and lower task dropping rates, shown in Fig. 9(f), as the network load increases. Unlike JOPA and JOPAV1, despite an increase in network load and offloading rate, AGDM only has a minimal increase in bandwidth and edge utilization. This is due to its highest dropping rate, which reduces its effective utilization of bandwidth and edge computing resources. In a high network load condition, by offloading more tasks while maintaining lower dropping rates, JOPA also performs better than the JOPAV1 in edge and bandwidth utilization. In In Figs. 9(a) and 9(b), the differences among the three schemes are initially negligible but becomes more notable as the network load (N) increases. In Fig. 9(c), AGDM prioritizes local task processing to minimize the overall processing latency, showing consistently high local processing resource utilization over different settings.

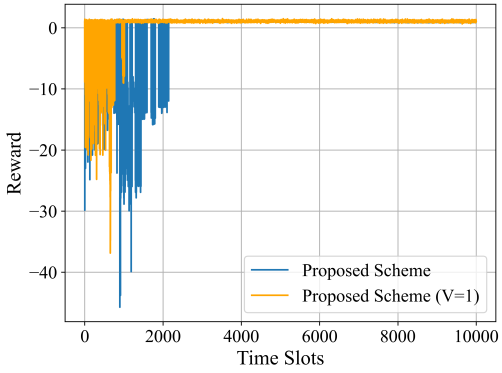
The JOPA achieves higher local processing resource utilization when the network load is low due to its flexibility in selecting less pruned models to achieve a better time utilization of local resources and higher task processing accuracy. As the network load increases, JOPA tends to offload more tasks to the edge, as shown in Fig. 9(e) thereby reducing the local side of the resource utilization. On the other hand, JOPAV1 shows consistently high local resource utilization over different network load conditions, balancing a trade-off between local processing and edge utilization. Fig. 9(d) provides a comparison of overall resource utilization among the three schemes, by aggregating local computing, edge computing, and bandwidth utilizations. JOPA consistently achieves the highest overall resource utilization, followed by JOPAV1 and AGDM. While all three schemes perform similarly under low network load conditions, the differences become more obvious as the numbers of gateways and sensors increase. Especially under a medium or high network load, the JOPA's strategy of balancing task offloading with local processing achieves higher overall utilization than the other two schemes.

- **Adaptability to network loads and dynamic QoS requirements:** Fig. 9(f) shows that JOPA effectively satisfies the accuracy and delay requirements with no task dropping under the network capacity. When the network load (N) keeps increasing, the task dropping happens due to the violation of accuracy and delay constraints as a result of exceeding the network capacity. However, the JOPA maintains the lowest task-dropping rate (below the target task-dropping rate limit of 1% [52]), leveraging its adaptability to time-varying delay and ac-

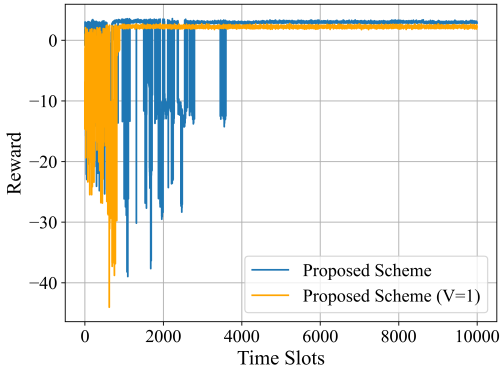
curacy requirements and its flexibility of pruned local processing model selection to balance the trade-off between accuracy and delay. In contrast, without the flexibility of local model selection, the JOPAV1 exceeds the tolerable limit when N increases beyond 175. The adaptability of the proposed JOPA to the dynamic delay and accuracy requirements is demonstrated in Figs. 10(a)-(d), where the JOPA outperforms the AGDM in adaptively satisfying the dynamically changing QoS requirements over time, leading to a much reduced task dropping rate shown in Fig. 9(f). The AGDM exhibits a much higher task-dropping rate without adapting to the dynamic delay and accuracy requirements.

- **Trade-off between QoS satisfaction and resource utilization:** A balanced trade-off between resource utilization and QoS satisfaction (i.e., delay and accuracy) is achieved, as shown in Figs. 11(a) and 11(b). While achieving the highest overall resource utilization and the lowest task-dropping rate, JOPA achieves slightly higher task processing accuracy than JOPV1 and AGDM, at the cost of sacrificing some delay performance, whereas the AGDM aims at achieving the minimum average E2E task processing delay with certain accuracy guarantee. For JOPA and JOPAV1, as the network load increases, a significant portion of tasks are offloaded to the edge server to maintain a consistently high average accuracy using the full-weight model while achieving high bandwidth and edge computing resource utilization. The JOPA and JOPAV1 consistently achieves higher average accuracy than AGDM, as shown in Figs. 11(a) and 11(b) by offloading more tasks to the edge server and maintaining a low task dropping rate with the satisfaction of the varied strict accuracy and

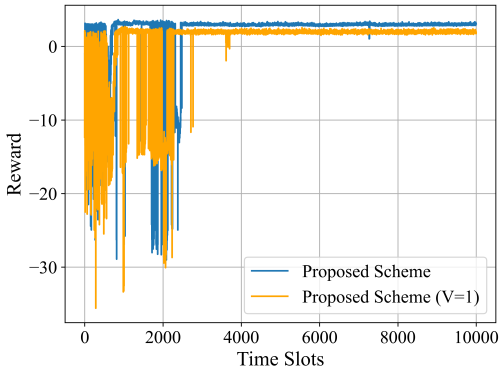
delay requirements corresponding to different criticality levels, listed in Table 3.



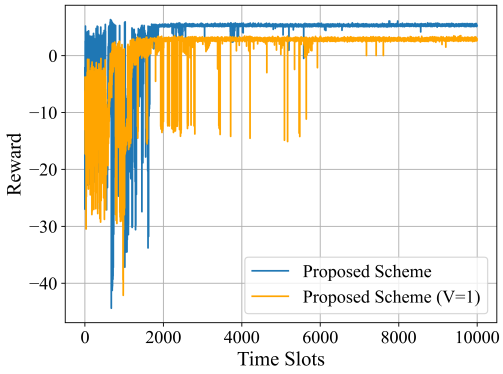
(a) G=5, N=100



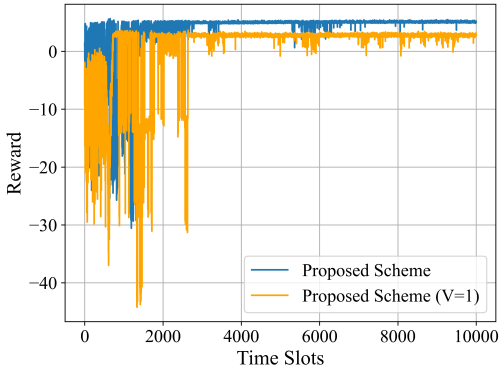
(b) G=5, N=125



(c) G=5, N=150



(d) G=5, N=175



(e) G=5, N=200

Figure 7: Training rewards for the proposed scheme for two instances, $V = 1$ and $V = 6$.

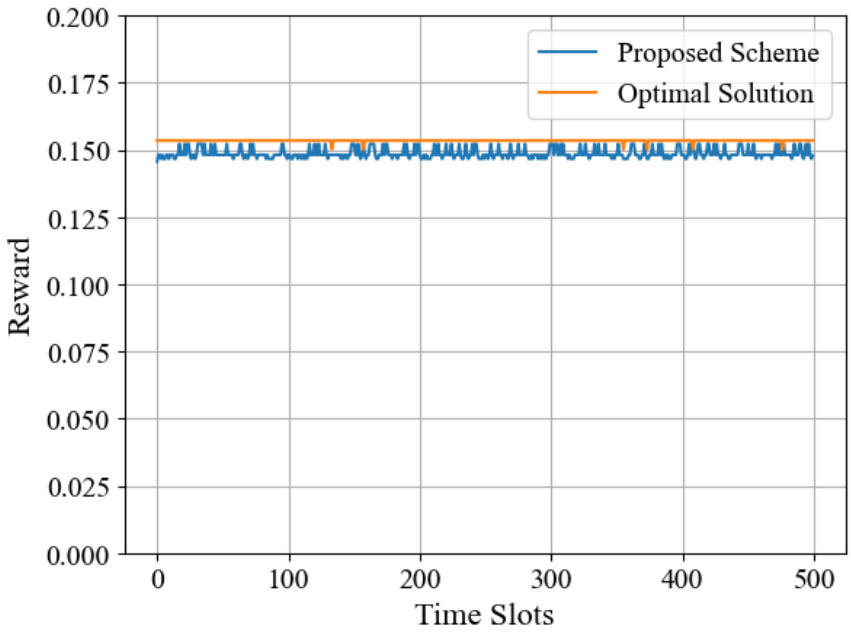


Figure 8: The reward comparison between the proposed scheme (after convergence) and the optimal solution.

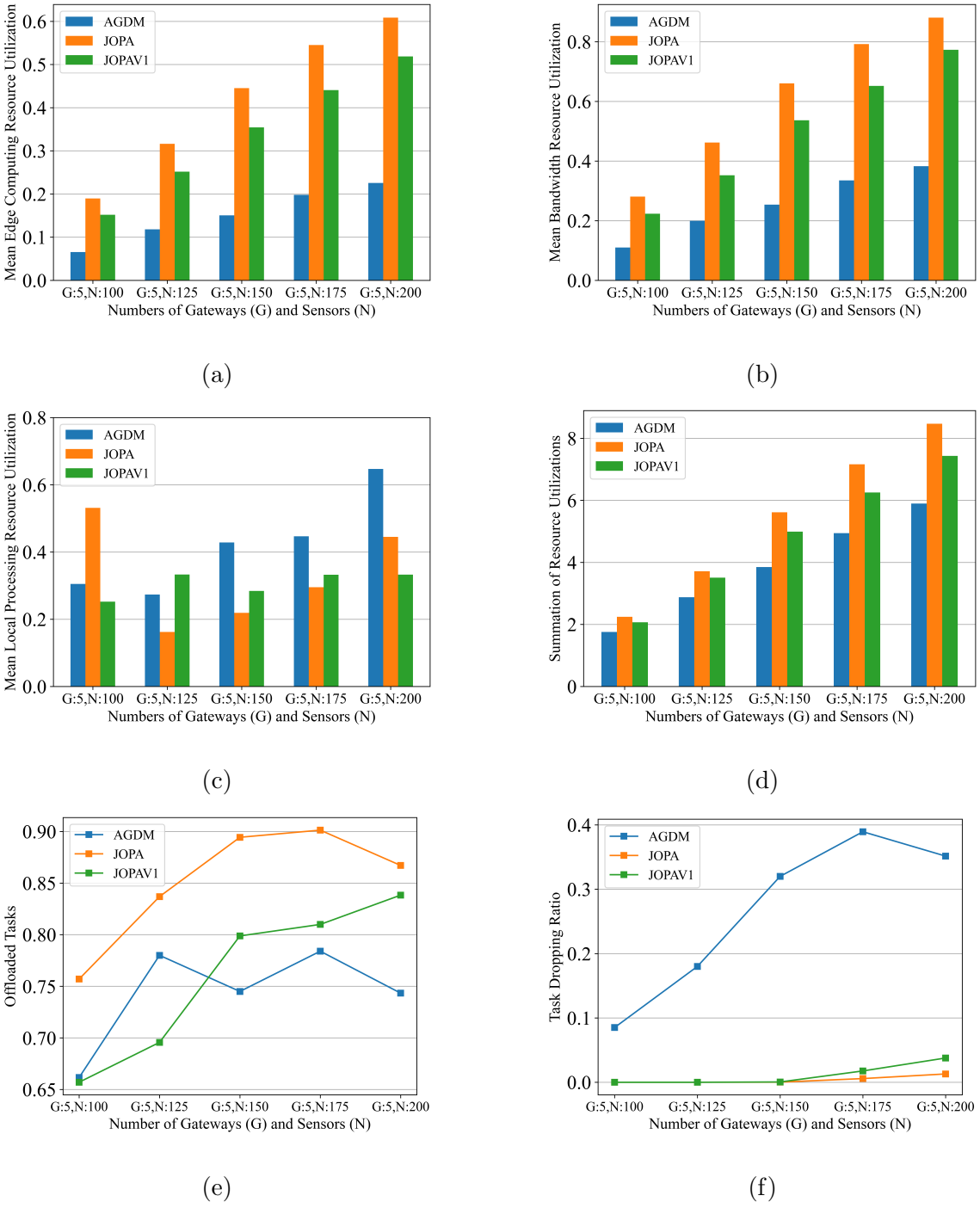


Figure 9: Comparison between JOPA, JOPA, and AGDM in terms of (a) bandwidth, (b) edge utilization, (c) local utilization, (d) overall utilization, (e) offloading ratio, and (f) task dropping ratio

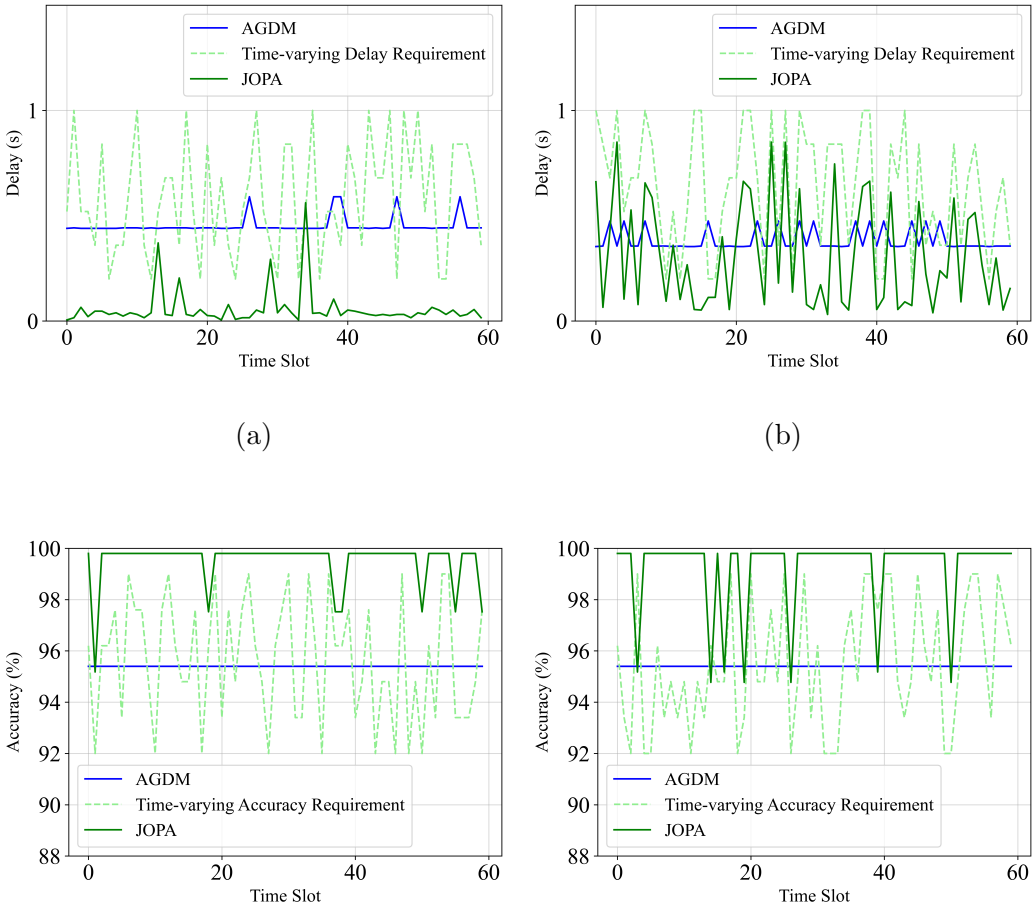
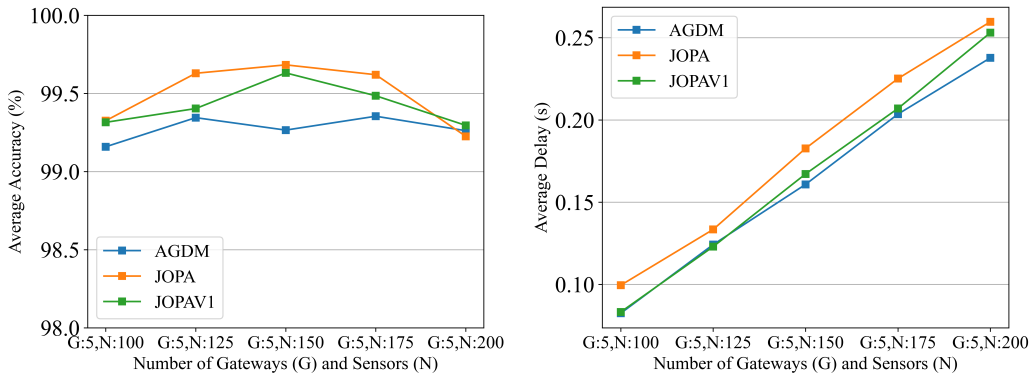


Figure 10: Comparison of the adaptability between JOPA and AGDM to delay requirements in (a) and (b), and accuracy requirements in (c) and (d).



(a) (b)

Figure 11: (a) Average inference accuracy, (b) E2E delay

17 Concluding Remarks

In this thesis, we have investigated joint task offloading, DNN model pruning, and computing resource allocation under a layered IIoT networking architecture to support diverse and dynamic QoS requirements of a fault detection service for industrial washing machines. We have formulated the problem as a stochastic optimization problem to maximize the overall radio bandwidth and computing resource utilization on the IGWs and the edge server while guaranteeing the per-slot time-varying E2E delay and inference accuracy requirements. To capture the network state transitions and the relations between states and policies, our problem is transformed as an MRP formulation which has large state and action spaces growing with the numbers of IGWs and IIoT sensors. To solve the MRP problem efficiently, we have developed a DRL-based solution, i.e., the JOPA algorithm, where the SAC algorithmic framework is customized to thoroughly explore the state and action spaces to obtain an improved solution. In the designed SAC framework: 1) The actor network is customized to support a mix of discrete and continuous actions. 2) Dual critic networks are employed to minimize the chance of Q-value overestimation. 3) Dual target networks are leveraged to enhance the training stability. Extensive simulations have been conducted to evaluate the performance of the JOPA algorithm and its advantages over two benchmark schemes. It is demonstrated that our proposed solution achieves superior performance in terms of maximizing the network resource utilization, satisfying the dynamic QoS requirements, and adapting to the varying network load. The proposed

scheme provides an efficient and robust solution framework for tackling a complex multi-dimensional resource allocation problem in an AI and edge computing-assisted IIoT environment.

18 Future Research

The research presented in this thesis opens several directions for future work, which can enhance the understanding of IIoT systems.

- In our current experiment, the available radio bandwidth is uniformly and statically allocated to individual IGWs. A valuable direction for future research could be to explore dynamic radio bandwidth allocation strategies based on network conditions or service demands. Evaluating such approaches requires optimization in the long run, which could provide deeper insights into the performance and robustness of IIoT systems.

- This thesis focused on a single task type with fixed task size and computation intensity. However, real-world IIoT environments contain tasks that may vary significantly in size, complexity, and resource needs. Future research could examine the system's adaptability to handling heterogeneous task types and explore how diverse computation loads affect task offloading and resource allocation strategies.

References

[1] Wen Wu, Peng Yang, Weiting Zhang, Conghao Zhou, and Xuemin Shen. Accuracy-guaranteed collaborative DNN inference in industrial IoT via deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 17(7):4988–4998, 2021.

[2] Shaobo Mao, Man Hon Cheung, and Vincent W. S. Wong. Joint energy allocation for sensing and transmission in rechargeable wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 63(6):2862–2875, 2014.

[3] Alp Bayar, Umut Şener, Kerem Kayabay, and P. Erhan Eren. Edge computing applications in industrial IoT: A literature review. In José Ángel Bañares, Jörn Altmann, Orna Agmon Ben-Yehuda, Karim Djemame, Vlado Stankovski, and Bruno Tuffin, editors, *Economics of Grids, Clouds, Systems, and Services*, pages 124–131, Cham, 2023. Springer Nature Switzerland.

[4] Kesavan Gunasekaran, V. Vinoth Kumar, A. C. Kaladevi, T. R. Mahesh, C. Rohith Bhat, and Krishnamoorthy Venkatesan. Smart decision-making and communication strategy in industrial internet of things. *IEEE Access*, 11:28222–28235, 2023.

[5] Weiting Zhang, Dong Yang, Youzhi Xu, Xuefeng Huang, Jun Zhang, and Mikael Gidlund. DeepHealth: A self-attention based method for instant intelligent predictive maintenance in industrial internet of things. *IEEE Transactions on Industrial Informatics*, 17(8):5461–5473, 2021.

[6] Nipun Setia. The blockchain-powered edge computing platform for developing smart internet of things (IoT) applications. In *2023 2nd International Conference on Futuristic Technologies (INCOFT)*, pages 1–6, 2023.

[7] David Hästbacka, Jari Halme, Laurentiu Barna, Henrikki Hoikka, Henri Pettinen, Martin Larrañaga, Mikael Björkbom, Heikki Mesiä, Antti Jaatinen, and Marko Elo. Dynamic edge and cloud service integration for industrial IoT and production monitoring applications of industrial cyber-physical systems. *IEEE Transactions on Industrial Informatics*, 18(1):498–508, 2022.

[8] Tie Qiu, Jiancheng Chi, Xiaobo Zhou, Zhaolong Ning, Mohammed Atiquzzaman, and Dapeng Oliver Wu. Edge computing in industrial internet of things: Architecture, advances and challenges. *IEEE Communications Surveys & Tutorials*, 22(4):2462–2488, 2020.

[9] Megha Sharma, Abhinav Tomar, and Abhishek Hazra. Edge computing for industry 5.0: Fundamental, applications, and research challenges. *IEEE Internet of Things Journal*, 11(11):19070–19093, 2024.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] Enrico Zio. Prognostics and health management methods for reliability prediction and predictive maintenance. *IEEE Transactions on Reliability*, 73(1):41–41, 2024.

[12] Chongwu Dong, Muhammad Shafiq, Maryam M. Al Dabel, Yanbin Sun, and Zhihong Tian. DNN inference acceleration for smart devices in industry 5.0 by decentralized deep reinforcement learning. *IEEE Transactions on Consumer Electronics*, 70(1):1519–1530, 2024.

[13] Samira Chouikhi, Moez Esseghir, and Leila Merghem-Boulahia. Computation offloading for industrial internet of things: A cooperative approach. pages 626–631, 2023.

[14] Shunpu Tang, Lunyuan Chen, Ke He, Junjuan Xia, Lisheng Fan, and Arumugam Nallanathan. Computational intelligence and deep learning for next-generation edge-enabled industrial IoT. *IEEE Transactions on Network Science and Engineering*, 10(5):2881–2893, 2023.

[15] R. Sultan, A. Refaey, and W. Hamouda. Resource allocation in CAT-M and LTE-A coexistence: A joint contention bandwidth optimization scheme. In *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), London, ON, Canada*, pages 1–6, 2020.

[16] Xiangjie Kong, Yuhan Wu, Hui Wang, and Feng Xia. Edge computing for internet of everything: A survey. *IEEE Internet of Things Journal*, 9(23):23472–23485, 2022.

[17] Siqi Zhang, Na Yi, and Yi Ma. A survey of computation offloading with task types. *IEEE Transactions on Intelligent Transportation Systems*, 25(8):8313–8333, 2024.

[18] Chengfang Ling, Kai Peng, Shangguang Wang, Xiaolong Xu, and Victor C. M. Leung. A multi-agent DRL-based computation offloading and resource allocation method with attention mechanism in MEC-enabled iiot. *IEEE Transactions on Services Computing*, pages 1–15, 2024.

[19] Xiaojie Zhang, Motahare Mounesan, and Saptarshi Debroy. EFFECT-DNN: Energy-efficient edge framework for real-time DNN inference. In *2023 IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 10–20, 2023.

[20] Chongwu Dong, Muhammad Shafiq, Maryam M. Al Dabel, Yanbin Sun, and Zhihong Tian. DNN inference acceleration for smart devices in industry 5.0 by decentralized deep reinforcement learning. *IEEE Transactions on Consumer Electronics*, 70(1):1519–1530, 2024.

[21] Weiwei Fang, Feng Xue, Yi Ding, Naixue Xiong, and Victor C. M. Leung. EdgeKE: An on-demand deep learning IoT system for cognitive big data on industrial edge devices. *IEEE Transactions on Industrial Informatics*, 17(9):6144–6152, 2021.

[22] Wenhao Fan, Shenmeng Li, Jie Liu, Yi Su, Fan Wu, and Yuan'An Liu. Joint task offloading and resource allocation for accuracy-aware machine-learning-based IIoT applications. *IEEE Internet of Things Journal*, 10(4):3305–3321, 2023.

[23] Weiting Zhang, Dong Yang, Haixia Peng, Wen Wu, Wei Quan, Hongke Zhang, and Xuemin Shen. Deep reinforcement learning based resource management for

DNN inference in industrial IoT. *IEEE Transactions on Vehicular Technology*, 70(8):7605–7618, 2021.

[24] Yanhua He, Yun Ren, Zhenyu Zhou, Shahid Mumtaz, Saba Al-Rubaye, Antonios Tsourdos, and Octavia A. Dobre. Two-timescale resource allocation for automated networks in IIoT. *IEEE Transactions on Wireless Communications*, 21(10):7881–7896, 2022.

[25] Bo Zhang and Chenghao Wang. Deep reinforcement learning-based predictive maintenance task offloading and resource allocation. In *IEEE 23rd International Conference on Communication Technology (ICCT)*, pages 659–664, 2023.

[26] Sixian Qin, Yingyang Chen, Shuai Wang, Zhixuan Xie, Miaowen Wen, and Derrick Wing Kwan Ng. Integrating edge intelligence and industrial IoT via learning-communication balancing power allocation. In *IEEE International Conference on Communications*, pages 861–866, 2024.

[27] Abhijeet Mahapatra, Santosh K. Majhi, Kaushik Mishra, Rosy Pradhan, D. Chandrasekhar Rao, and Sandeep K. Panda. An energy-aware task offloading and load balancing for latency-sensitive IoT applications in the fog-cloud continuum. *IEEE Access*, 12:14334–14349, 2024.

[28] Yifan Chen, Zhuoquan Yu, Christine Mwase, Yi Jin, Xin Hu, Lirong Zheng, and Zhuo Zou. Self-aware collaborative edge inference with embedded devices for task-oriented IIoT. In *IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, pages 1–5, 2023.

[29] S. Vadera and S. Ameen. Methods for pruning deep neural networks. *IEEE Access*, 10:63280–63300, 2022.

[30] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[31] F. Zhang, G. Han, L. Liu, M. Martínez-García, and Y. Peng. Deep reinforcement learning based cooperative partial task offloading and resource allocation for IIoT applications. *IEEE Transactions on Network Science and Engineering*, 10(5):2991–3006, September-October 2023.

[32] T. Lillicrap et al. Continuous control with deep reinforcement learning. In *Proc. Int. Conf. Learn*, San Juan, Puerto Rico, 2016. Representations.

[33] N. H. Mahmood, N. Marchenko, M. Gidlund, and P. Popovski. *Wireless Networks and Industrial IoT*. Springer, New York, NY, USA, 2020.

[34] X. Wang, S. Lu, W. Huang, Q. Wang, S. Zhang, and M. Xia. Efficient data reduction at the edge of industrial internet of things for PMSM bearing fault diagnosis. *IEEE Transactions on Instrumentation and Measurement*, 70:1–12, 2021.

[35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint, 2014.

[36] X. Liu, W. Xia, and Z. Fan. A deep neural network pruning method based on gradient L1-norm. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, China, 2020, pp, 2070-2074. Chengdu.

[37] W. Kang, D. Kim, and J. Park. DMS: Dynamic model scaling for quality-aware deep learning inference in mobile and embedded devices. *IEEE Access*, 7:68048–16805, 2019.

[38] Z. Chen, Z. Chen, J. Lin, S. Liu, and W. Li. Deep neural network acceleration based on low-rank approximated channel pruning. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(4):1232–1244, April 2020.

[39] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo. Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism. *IEEE Transactions on Mobile Computing*, 22(5):3017–3030, May 2023.

[40] Y. Bian, Y. Sun, M. Zhai, W. Wu, Z. Wang, and J. Zeng. Dependency-aware task scheduling and offloading scheme based on graph neural network for MEC-assisted network. In *2023 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pages 1–6, August 2023.

[41] Tiantian Yang, Rong Chai, and Liping Zhang. Latency optimization-based joint task offloading and scheduling for multi-user MEC system. In *29th Wireless and Optical Communications Conference (WOCC)*, pages 1–6, 2020.

[42] Q. Ye, W. Shi, K. Qu, H. He, W. Zhuang, and X. Shen. Joint RAN slicing and computation offloading for autonomous vehicular networks: A learning-assisted hierarchical approach. *IEEE Open Journal of Vehicular Technology*, 2:272–288, 2021.

[43] F. Kelly. Notes on effective bandwidths. *Stochastic networks: theory and applications*, 4:141–168, 1996.

[44] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint, 2014.

[45] L. Zeng, E. Li, Z. Zhou, and X. Chen. Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things. *IEEE Network*, 33(5):96–103, 2019.

[46] Will Dabney, Georg Ostrovski, and André Barreto. Temporally-extended ϵ-greedy exploration. *arXiv preprint arXiv:2006.01782*, 2020.

[47] Peter Auer. Using upper confidence bounds for online learning. In *Proceedings 41st annual symposium on foundations of computer science*, pages 270–279. IEEE, 2000.

[48] H. Tran-Dang, K. H. Kwon, and D. S. Kim. Bandit learning-based distributed computation in fog computing networks: A survey. *IEEE Access*, 2023.

[49] Francesco Pase, Marco Giordani, Giampaolo Cuozzo, Sara Cavallero, Joseph Eichinger, Roberto Verdone, and Michele Zorzi. Distributed resource allocation for urllc in iiot scenarios: A multi-armed bandit approach. In *IEEE Globecom Workshops (GC Wkshps)*, pages 383–388, 2022.

[50] Xiaolan Liu, Jiadong Yu, Jian Wang, and Yue Gao. Resource allocation with edge computing in IoT networks via machine learning. *IEEE Internet of Things Journal*, 7(4):3415–3426, 2020.

[51] Weiting Zhang, Dong Yang, Wen Wu, Haixia Peng, Ning Zhang, Hongke Zhang, and Xuemin Shen. Optimizing federated learning in distributed industrial IoT: A multi-agent approach. *IEEE Journal on Selected Areas in Communications*, 39(12):3688–3703, 2021.

[52] Saúl Langarica, Christian Rüffelmacher, and Felipe Núñez. An industrial internet application for real-time fault diagnosis in industrial motors. *IEEE Transactions on Automation Science and Engineering*, 17(1):284–295, 2020.