From Edge to Equipment: Design and Implementation of a Machine-Learning-Enabled Smart Manufacturing System

by

Hung Nguyen

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 6, 2023

Keywords: edge computing, smart manufacturing, computation offloading, computing performance, machine learning, secs gem

Abstract

In smart manufacturing, data management systems are built with a multi-layer architecture, in which the most significant layers are the edge and cloud layers. The edge layer, not surprisingly, renders support to data analysis that genuinely demands low latency. Cloud platforms store vast amounts of data while performing extensive computations such as machine learning and big data analysis. This type of data management system has a limitation rooted in the fact that all data ought to be transferred from the equipment layer to the edge layer in order to perform thorough data analyses. Even worse, data transferring adds delays to a computation process in smart manufacturing. In the first part of the dissertation studies, we investigate an offloading strategy to shift a selection of computation tasks toward the equipment layer. Our computation offloading mechanism opts for smart manufacturing tasks that are not only light weighted but also do not require saving or archiving at the edge/cloud. We demonstrate that an edge layer is able to judiciously offload computing tasks to an equipment layer, thereby curtailing latency and slashing the amount of transferred data during smart manufacturing. Our experimental results confirm that the proposed offloading strategy offers the capability for data analysis computing in real-time at the equipment level - an array of smart devices are slated to speed up the data analysis process in semiconductor manufacturing. With collected data, we apply the empirical results as training and testing data to construct a machine learning model that recommends whether it is advantageous to offload computation from the edge layer to the equipment layer based on the current system status.

In the second part of the dissertation, we elaborate on a novel scheduler – a scheduling algorithm that allocates edge computing resources with awareness of workload at the equipment layer. Our edge scheduling algorithm is adroit at determining the most appropriate scenarios to offload computing tasks from edge to equipment, thereby maximizing throughput while meeting the priority requirements of the tasks. A limitation of current research on edge scheduling is that available resources from the equipment layer were not used to achieve maximum throughput. The main difference between our scheduling algorithm and the other edge scheduling techniques is that we use available resources at the equipment layer. Other State-of-the-Art scheduler algorithms are not considering using resources at the equipment level. By using additional resources at the equipment level, our experimental results shows that the total computation time has been shortened by 27.75% and the throughput has increased by 38.45% comparing to Hybrid Computing Solution or HCS scheduling performance, a State-of-the-Art scheduling algorithm.

Moreover, to enable offloading computation tasks from the edge layer to the equipment layer, the edge layer ought to be able to assign specific computation tasks to the equipment. In semiconductor manufacturing, the host computer located at the edge layer communicates to the equipment through SECS/GEM communication protocol. As the last piece in this dissertation, we design an advanced protocol on the SECS/GEM interface to facilitate the transfer of computational tasks from the edge to the equipment. Current research on Equipment level Fault Detection and Classification (FDC) suggested to build a software module at the equipment layer to perform computation. The limitation of this technique is that it requires software modification every time the computation logic changes. Furthermore, this technique is not flexible to allow the equipment to perform any other computation tasks besides FDC. With the new protocol in place, the host has the capability to dynamically assign data analysis tasks to the equipment. Additionally, the protocol also offers a mechanism for the equipment to report back the analysis results to the host.

Acknowledgments

I would like to express my deepest appreciation to Dr. Xiao Qin from Auburn University and Dr. Yi Zhou from Columbus State University for the guidance and support throughout my research at Auburn University. Dr. Qin has provided me invaluable direction for my research. Dr. Zhou spent countless meetings to share his knowledge and expertise with me. I am also thankful to my Ph.D. committee, Dr. Shubhra (Santu) Karmaker, Dr. Jakita Thomas, Dr. Alvin Lim, and my University Reader, Dr. Eric Wetzel, for all the guidance and outstanding feedbacks.

I am also grateful to the executive team at Yield Engineering Systems, Inc. for the support during my research progress.

Finally, but the most importantly, I appreciate for the support from my wife, Loan, and my children, Michelle and Jenny, for unconditional love and support.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

APC	Advanced Process Control
CEID	Collection Event ID
FDC	Fault Detection and Classification
GEM	The Generic Model for Communications and Control of Manufacturing Equipment
HCS	Hybrid Computing Solution
KNN	K-Nearest Neighbors
LDA	Linear Discriminant Analysis
SECS	SEMI Equipment Communication Standard
SEMI	Semiconductor Equipment and Materials International
SPC	Statistical Process Control
SVM	Support Vector Machine
VID	Variable ID

Chapter 1

Introduction

1.1 Background

With the fast development of information technology, cloud computing, machine learning, natural language processing, artificial intelligence, modern production and manufacturing of the semiconductor industry have relied on smart manufacturing systems to achieve high productivity, cost reduction, efficiency in production control, and early detection of production issues [13, 26, 29, 30, 34]. In semiconductor manufacturing, monitoring and analyzing process data is critical and indispensable to bolster productivity [32, 69]. Data analysis techniques such as machine learning, deep learning, statistical process control (SPC), fault detection and classification (FDC), and advanced process control (APC) are deployed for the purpose of defect prediction, maintenance recommendations, and resource allocation [4, 8, 11, 22, 41, 59].

In the process of semiconductor manufacturing, production data from semiconductor equipment is collected and transferred to an edge layer and a cloud layer for further analysis and storage. Real-time computing in smart manufacturing is performed at the edge layer [21, 58]. If production data ought to be stored or archived, data will be transferred and inserted into cloud databases [38, 78].

Most data management systems for manufacturing are built with two computing layers: the cloud layer and edge layer [35, 40, 61]. The cloud layer, in a growing number of cases, are applied to store a large amount of data and to support complicated computing tasks that require abundant computing resources. These computing tasks include, but are not limited to, artificial intelligence algorithms, big data applications, machine learning algorithms, and deep learning networks [5, 12, 19, 49]. The cloud layer, with its rich resources, can be beneficial for big data tasks as well as long-term data storage [53, 23, 14]. The edge platform, which is closer to the devices and equipment, is used for low-latency computing tasks that are critical for the early detection of fault processes [65]. One example is using the edge platform to perform and orchestrate data analyses. A holistic combination of cloud and edge computing in data management systems helps to curb the latency in real-time data analysis computation. In a wide range of these systems, data analysis computing is pushed from the cloud level to the edge level with the hope to reduce the latency of data transferring and analysis computing [25, 33, 45, 46, 67].

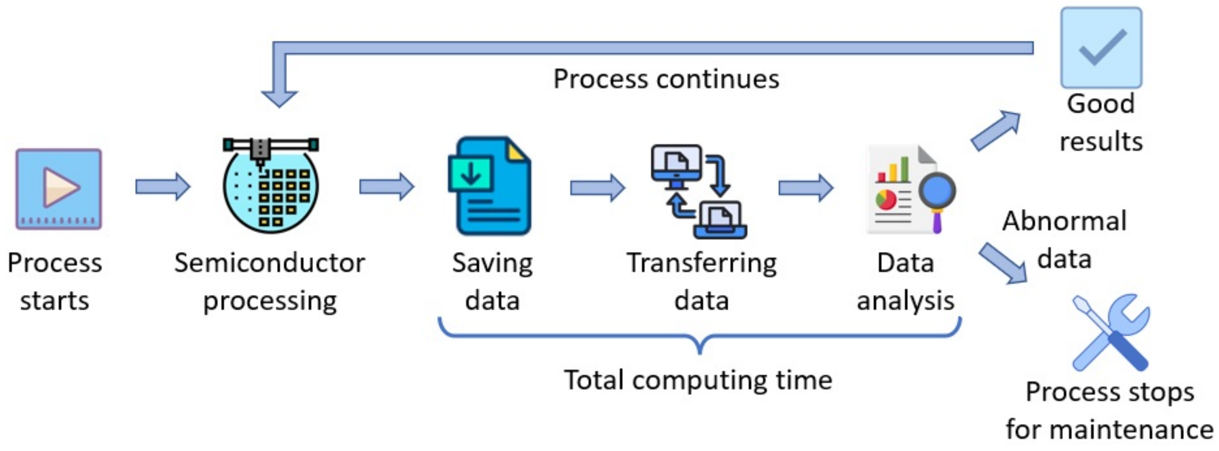


Figure 1.1: An Example of Computing Time in Semiconductor Processing

1.2 Motivations and Research

1.2.1 Why this dissertation research is important?

We can see from Fig. 1.1 that data analytic computing tasks are frequently and repeatedly performed during semiconductor manufacturing. The computing tasks may be simple tasks such as process data monitoring, statistical process control, resource allocation, or complex algorithms for early failure prediction and other purposes [16, 32]. Data analysis in smart manufacturing, especially in semiconductor manufacturing, is vital to achieve high product quality [2, 30]. Reducing data analysis computation latency is desirable for increasing production throughput [9, 45].

As chip size is reduced to the single-digit nanometer level, the amount of processing data that is needed for analysis significantly increases [7, 17]. Performing data analysis at the edge layer or the cloud layer requires a vast amount of data to be transferred from equipment [65, 66]. Accordingly, the latency of data analysis and the computing process is enlarged. Network bandwidth, of course, is likely to be overloaded and saturated by delivering too much data from thousands of equipment in the factory to the edge layer [45, 67]. Furthermore, handling all types of computing tasks at the edge layer inevitably faces the grand challenge of system complexity: it is obliged to perform data management and analysis on all different data types from heterogeneous equipment types in semiconductor processing [47, 60]. These problems motivate us to propose an offloading method, for smart manufacturing in the semiconductor industry, to judiciously shift a selection of computing tasks from the edge layer to the equipment layer.

1.2.2 Dissertation Statement

The primary goal of this dissertation study is to reduce the latency of computation tasks in semiconductor smart manufacturing environment when the amount of data increases from 2000 data points to 50000 data points per wafer. We aim to solve this issue with three main tasks: analyzing the performance of offloading computation tasks from the edge layer to the equipment layer, designing a scheduling algorithm to support offloading, and proposing a protocol to provide a mechanism for dynamically assigning computation tasks to the equipment.

1.2.3 Research Questions

To achieve the aforementioned overarching goal, we will address the four following intriguing research questions through the dissertation.

- Research Question 1: What is the key performance discrepancy between edge-based computing and equipment-based computing?

- Research Question 2: How to determine the tasks that should be offloaded from the edge layer to the equipment layer to enhance the performance of smart manufacturing systems?
- Research Question 3: How to incorporate task-offloading into edge resource scheduling?
- Research Question 4: How to apply offloading tasks to semiconductor manufacturing environments?

1.3 Benefits and Contributions

In this dissertation, we propose a judicious method of selecting computing tasks that can be offloaded to the equipment level to achieve the lowest computing latency for smart manufacturing. With this approach in place, some computing tasks are performed at the edge level while the other selected tasks are running at the equipment layer where the data is originated.

Our empirical study in the arena of smart manufacturing in the semiconductor industry demonstrates three-fold strengths of our computation offloading mechanism. An immediate benefit of our new design is a reduction in the overhead of handling data for computing tasks at the edge level, including storing data into temporary storage at the edge for computation. Another advantage is curbing computing latency for smart manufacturing tasks: our offloading mechanism enables smart manufacturing systems to detect defective materials early in a manufacturing process. As a third good point, our offloading mechanism plays a vital role in smart manufacturing because each piece of equipment produces a different type of data, including the format of data and the meaning of data [66]. Our offloading method reduces the development time to build a data management system that needs to handle a wide variety of data types and all computing tasks at the edge layer. The aforementioned new benefits offered by our offloading mechanism are tabulated below.

- *Benefit 1.* Reducing the overhead of handling data for computing tasks at the edge level.
- *Benefit 2.* Curbing computing latency for smart manufacturing tasks.
- *Benefit 3.* Shortening the development time spent in building a data management system.

We systematically conduct a performance comparison on edge-layer-based computing against its equipment-layer-based counterpart. The comparison study weighs in multiple factors, including algorithmic complexity, data size, CPU utilization, and computer type - edge servers or equipment nodes. On the basis of this empirical study, we propose an algorithm to select tasks that can be offloaded to the equipment layer, aiming to immensely shorten processing latency.

Moreover, we propose a resource scheduling algorithm that takes into account the total resources from both the edge and the equipment. In doing so, this novel resource scheduling algorithm can offload computing tasks to the equipment layer.

Furthermore, to enable offloading computing tasks from the edge layer to the equipment layer in semiconductor manufacturing, we proposed a protocol for the edge to send requests to the equipment. In semiconductor industry, SECS/GEM interface has become a dominant method for the host computer on the edge layer to communicate with the equipment by sending commands and requesting data and status [44, 81]. Therefore, we build our protocol based on the format of SECS/GEM interface. Powered by this protocol, the host computer on the edge can define computing tasks on the equipment. The host can also define the events to trigger computing tasks to be executed. Additionally, the host can request computing results from the equipment.

The contributions of this research are summarized as follows.

- *Contribution 1.* We conduct a performance comparison between edge-based computing and equipment-based computing.

- *Contribution 2.* We propose a computation offloading mechanism to cut back the latency of smart manufacturing tasks.

- *Contribution 3.* We propose a machine learning model to generate computation offloading (yes/no) recommendations.

- *Contribution 4.* We propose a resource scheduling that enables offloading computing tasks from the edge layer to the equipment layer.

- *Contribution 5.* We propose a communication protocol for the edge layer to communicate with the equipment layer to define computing tasks, trigger computing tasks, and collect computing results.

1.4 A Roadmap

Chapter 2 reviews related works about edge computing for the smart manufacturing environment. Chapter 3 explains our proposed system for offloading computing tasks from the edge layer to the equipment layer. The section 3.1 describes the model we use for calculating total computation time at the edge layer and the equipment layer. After that, in section 3.2 we describe the system settings we used for experiments and the factors that can affect the experimental results. Then the results of our experiments are reviewed and discussed in section 3.3. Based on the results, we propose a selection method for offloading computing tasks to the equipment level. This section also introduces several machine learning models that we used for evaluating our experimental results. Section 3.4 explains in detail the evaluation of experimental results with various machine learning models and selects the best model for offloading decisions. In chapter 4, we propose an algorithm for our novel resource scheduler that can be applied to a pool of resources from both the edge layer and the equipment layer. We also run experiments and compare our results with a State-of-the-Art scheduling from [33]. Chapter 5 discusses in detail our new protocol for communication between the edge layer and the equipment layer. The new protocol is designed to allow the host computer on the edge layer to dynamically assign computing tasks to the equipment. Chapter 6 gives conclusions and discusses additional research directions on offloading computing tasks from the edge layer to the equipment layer.

Chapter 2

Related Works

This chapter presents research that is related to our studies. The chapter is organized as follows. Section 2.1 discusses previous research on smart manufacturing systems based on edge and cloud layers. Section 2.2 presents research related to offloading computation to equipment. Finally, section 2.3 walks through research on resource scheduling on the edge layer.

2.1 Smart Manufacturing Systems with Cloud-Edge

Edge computing has evolved as an important technique for enterprise data management systems [6, 39, 50, 79]. Data generated in manufacturing has increased in both volume and complexity [2, 42, 51]. Building an effective system architecture for data management and analysis is vital for smart manufacturing, especially in time-sensitive environments such as semiconductor processing [18, 47].

Umpteen attempts have been made to design system architectures to slash the computing time for real-time applications [15, 48, 72, 75]. Proposed system architectures tried to reduce latency in data analysis computing by creating multi-layer systems which collaborate between the cloud and edge layers [24, 52]. Latency is one of the most important metrics for computing performance evaluation. The computing performance can be achieved by offloading computing tasks from the cloud to the edge where it is closer to the data [20, 57].

In [33], the authors proposed a hybrid computing framework to support various real-time requirements in smart manufacturing supported by edge computing. This hybrid architecture system includes the following elements: cloud server, device computing layer, software defined network layer, and edge computing layer. In this architecture, the device computing layer is responsible for driving mechanical devices. On the other hand, the edge servers are used for real-time computing tasks in the edge layer since this layer is close to the device layer where the data is produced. In the cloud layer, the servers are used for computationally intensive tasks such as AI and machine learning. Lastly, the software-defined network is used for coordination among different computing layers for resource scheduling to achieve low latency.

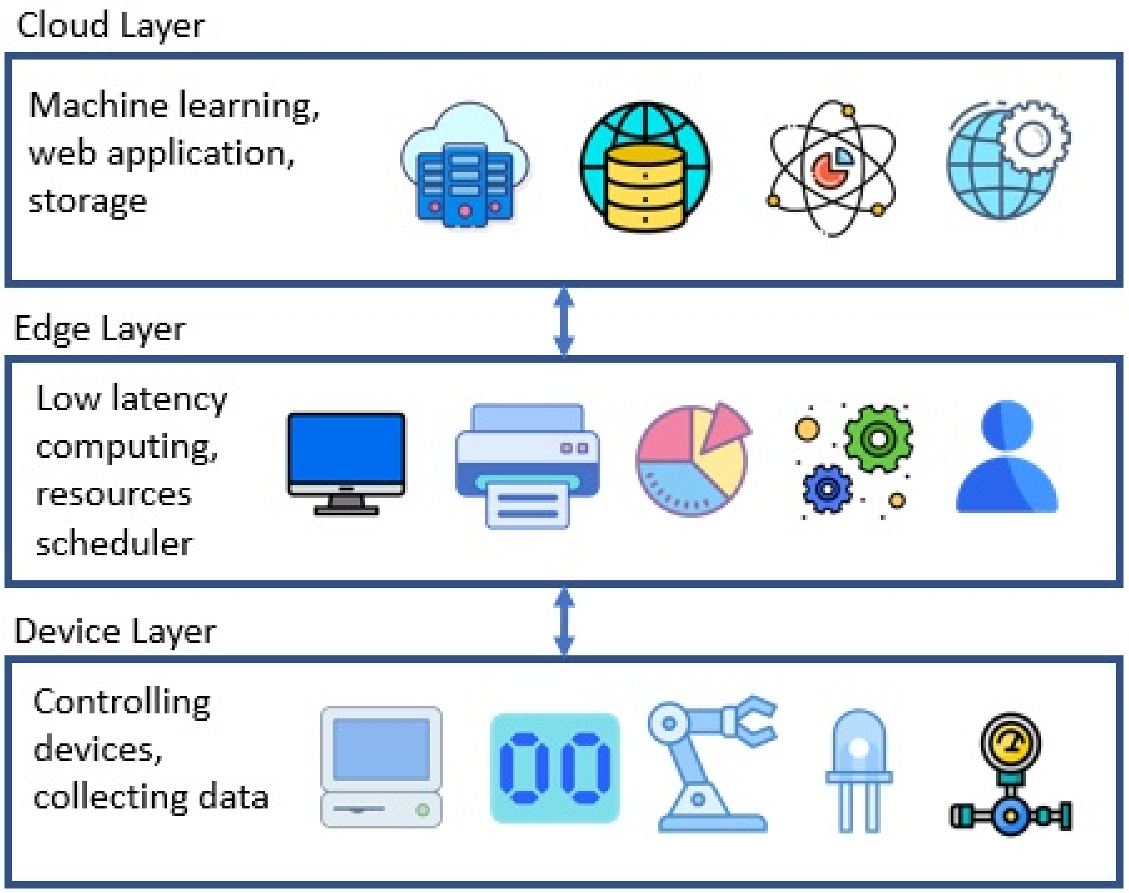

Cloud Layer
Machine learning, web application, storage
Edge Layer
Low latency computing, resources scheduler
Device Layer
Controlling devices, collecting data

Figure 2.1: A Typical Edge-Cloud System Architecture

The authors in [71] proposed an edge-supported cloud computing platform specifically for smart manufacturing. This platform includes three modules: the edge production module, the edge metrology module, and the cloud module. The edge metrology module is used for quality control, while the edge production module aids in continuous process control. The cloud is used for AI-assisted decision-making to support smart manufacturing. The edge computing layer combines the edge metrology module and the edge production model. In this platform, defective chips can be detected early in the edge computing layer leading to improvement in product quality and reduction in production costs.

In [51], the authors introduced a hierarchical architecture that includes the cloud, the fog, and the edge. The cloud is utilized for big data analysis and storage in this architecture. It also supports large-scale collaboration among the layers within the architecture. The fog computing layer is considered as an extension of cloud computing and provides computing services closer to the devices. Edge computing also supports computation like fog computing, but it is closer to data sources. In this architecture, fog computing utilizes interconnection capabilities among nodes, and edge computing is performed in isolated edge nodes.

The authors in [62] proposed using cloud only, without the edge layer, to support data analytic in semiconductor manufacturing. In this design, data from the equipment layer is transferred through the cloud gateway to the cloud infrastructure, beyond the facility's firewall. A software application will be installed on the cloud to perform data storage and analysis. Fault detection and classification (FDC) is performed at the cloud layer. The software on the cloud will send notification back to the system if it detects any issues.

Similarly, multiple cloud-edge systems have been proposed in [3, 31, 40, 54, 74, 64, 73]. The overall architecture of those systems can be summarized as shown in Fig. 1.

There are four common points in the proposed cloud-edge architecture.

1) Moving time-sensitive computing tasks from the cloud layer to the edge layer in order to achieve low latency and early detection of defects.

2) The physical device layer produces and sends the data to the edge layer for computing. This layer includes machines, sensors, cameras, motors, industrial PCs, and other mechanical and electrical devices.

3) An algorithm exists to coordinate and schedule computing tasks in the edge layer.

4) The systems were built to support smart manufacturing.

The above architecture faces the issue of longer time requirement to transfer data from the device layer to the edge layer because the amount of data produced by the device layer is increasing significantly. Although computing tasks have been pushed down from the cloud to the edge, which is closer to the data, latency in computing at the edge layer increases when the amount of data increases. Furthermore, the local bandwidth from the device layer to the edge layer will become a bottleneck with the increase in the amount of data that needs to be transferred from equipment to the edge layer.

2.2 Offloading Computation Tasks

Some significant efforts have been made to perform data analysis at the equipment level. In [28], the authors introduced the Equipment level Fault Detection and Classification (FDC) System to perform optimization and anomaly detection of semiconductor equipment. In this design, the equipment transfers some data to the Equipment level FDC System locally instead of the host. The Equipment Level FDC System can perform data analysis in a shorter time since it does not add data transfer latency into the analysis process and can react faster to any issues in material processing.

In [36], the authors proposed a system that can be mainly used in mobile communication. The system includes three layers: cloud computing layer, edge computing layer, and end device layer. In this system, the edge layer includes edge servers and edge devices. Edge servers are deployed on the edge network and act as bridges between edge devices and the cloud. The end device layer includes mobile servers and mobile devices. Mobile servers provide computing services for the assign tasks. Mobile devices are end devices that generate tasks and can choose to process tasks locally or offload the tasks to the edge servers for computing or scheduling to other mobile servers for computing.

In an effort to reduce computation latency, other studies have been conducted to try to offload computation tasks from the edge to other available resources such as local servers, other mobile devices, or nearby smart vehicles on freeway [27, 68, 77].

2.3 Resources Scheduling

With the rapid development of cloud-edge systems, scheduling resources to perform computing tasks became a critical process for efficiently managing system resources. In [33], the authors proposed a scheduling algorithm to select an edge server that can satisfy the condition that the queued time is less than the required time. If there are multiple edge servers that satisfy the condition, the algorithm will randomly choose an edge server to perform the task. In [1], the authors proposed a hybrid workflow scheduling on cooperative edge cloud computing. This cooperative edge cloud computing includes three layers. Layer 1 is IoT devices. Layer 2 is cooperative edge nodes which consists of edge devices and edge data centers. Layer 3 is multi-cloud Services. Layer 3 consists of powerful resources to carry out extensive computing tasks such as machine learning, business intelligence, and interactive visualization. In this cooperative edge-cloud computing system, the scheduling method estimates and plans resources based on QoA (Quality of Service) parameters and is responsible for selecting optimal virtual machines for task execution. There are many research on scheduling for cloud-edge or cloud-fog-edge systems [10, 37, 43, 63, 70, 76], task scheduling on the edge and fog layer has been developed and discussed to support smart manufacturing.

Chapter 3

From Edge to Equipment: Design and Implementation of a Machine-Learning-Enabled Smart Manufacturing System

This chapter presents our studies on offloading computation tasks from the edge layer to the equipment layer and is organized as follows. Section 3.1 presents how we model computation time at the edge layer and the equipment layer. Next, section 3.2 explains the setups for our experiments, followed by experimental results in section 3.3. Section 3.4 discusses in detail how we apply machine learning models based on our collected data. The summary of this chapter is outlined in section 3.5.

3.1 Modeling Computation Time at the Edge and Equipment Layers

We develop a simple yet effective computation time model with respect to two cases, which facilities making decisions of offloading computation from the edge layer to the equipment layer.

3.1.1 Computing Time on the Edge Layer

In this scenario, data analytic is carried out at the edge layer. To this end, characteristics data of each processed material (e.g., wafer) and the information of equipment statuses must be transferred to the edge layer for computation. One common method in the semiconductor industry for transferring data from the equipment layer to the edge layer is file transfer. More specifically, the control system running on the equipment computer first writes processed data to a file during material processing. Next, after the material processing is finished, the control system sends the data file to the edge. Then, the edge performs analytical computations to check the system status and determine if the product quality is normal for the process to continue. The computing time on the edge layer can be expressed as:

$$T_{Ed\_total} = T_{Eq\_write} + T_{transfer} + T_{Ed\_read} + T_{Ed\_comp}, \tag{3.1}$$

where T_{Ed_total} denotes the total time to perform computations on the edge server; T_{Eq_write} denotes the total time to write all process data or test data for material to data files; $T_{transfer}$ denotes the total time to transfer data files for material from the equipment layer to the edge layer; T_{Ed_read} represents the time for the edge server to read material data files; and T_{Ed_comp} is the time for the edge server to perform computing tasks on the data set.

Assuming there are N process steps in total, and process step i^{th} has M_i data files, the total time for an equipment computer to write all process data or test data to data files is computed as follows:

$$T_{Eq\_write} = \sum_{i=1}^{N} \sum_{j=1}^{M_i} t_{Eq\_write,i,j}, \tag{3.2}$$

where $t_{Eq\_write,i,j}$ denotes the time to write data to output data file j^{th} at process step i^{th}.

The total time to transfer data files from the equipment layer to the edge layer is calculated as:

$$T_{transfer} = \sum_{k=1}^{K} t_{transfer,k}, \tag{3.3}$$

where K represents the number of files to be transferred, and $t_{transfer,k}$ denotes the time for transferring file k^{th}.

And K can be computed as:

$$K = \sum_{i=1}^{N} M_i, \tag{3.4}$$

where M_i is the number of data files that are produced at process step i^{th}.

3.1.2 Computing Time on the Equipment layer

In the scenario of performing analytical computing at the equipment layer, there is no need to transfer data files from the equipment layer to the edge layer. In particular, data produced at the equipment layer is saved to local data files. After the material processing is completed, the control system running on the equipment computer reads the data file to perform analytical computations. Then, based on the calculation result, the control system determines whether the semiconductor manufacturing system proceeds to the next process. Accordingly, the computing time on the equipment layer can be formed as:

$$T_{Eq\_total} = T_{Eq\_write} + T_{Eq\_read} + T_{Eq\_comp}, \tag{3.5}$$

where T_{Eq_total} denotes the total time to perform computations on the equipment layer; T_{Eq_write} denotes the total time for an equipment computer to write all process data or test data for material processing to data files, which is calculated according to Equation (3.2); T_{Eq_read} is the time for the equipment computer to read the local data files; and T_{Eq_comp} represents the time for the equipment computer to perform computing tasks on the data set.

3.1.3 Machine Learning Models

After collecting experimental data, we fit our data set into machine learning models to evaluate the model that can be a good fit for computation offload recommendation.

Logistic Regression Classification Model

The logistic function has the following form:

$$p(x) = \frac{1}{1 + e^{-(x-\mu)/s}} \tag{3.6}$$

Where μ is a location parameter (p(μ) = 1/2) and s is a scale parameter.

Our data set will have multiple features, p_1, p_2, ..., p_n, the logistic function can be rewritten as follows:

$$p([x_1, x_2, ..., x_n]) = \frac{1}{1 + e^{-([x_1,x_2,...,x_n]-\mu)/s}} \tag{3.7}$$

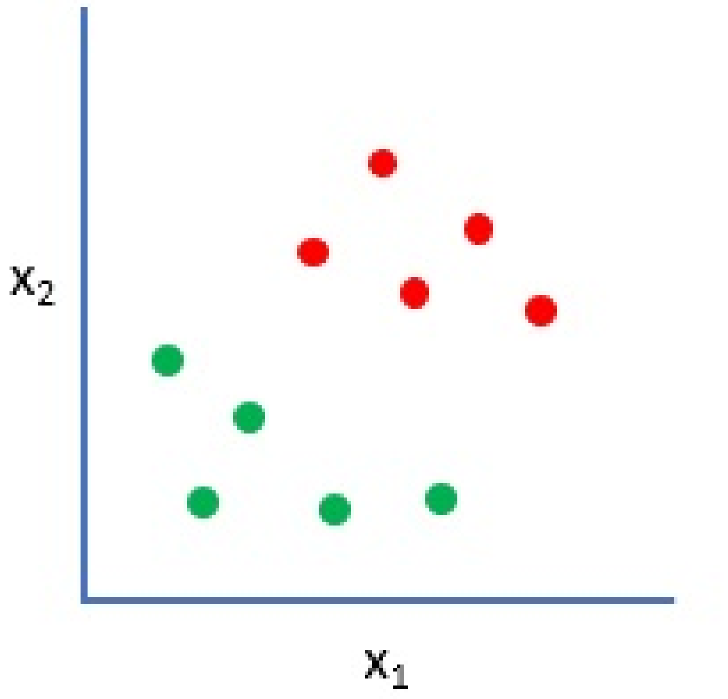


Figure 3.1: Dimensionality Reduction in LDA: Data set with 2 features

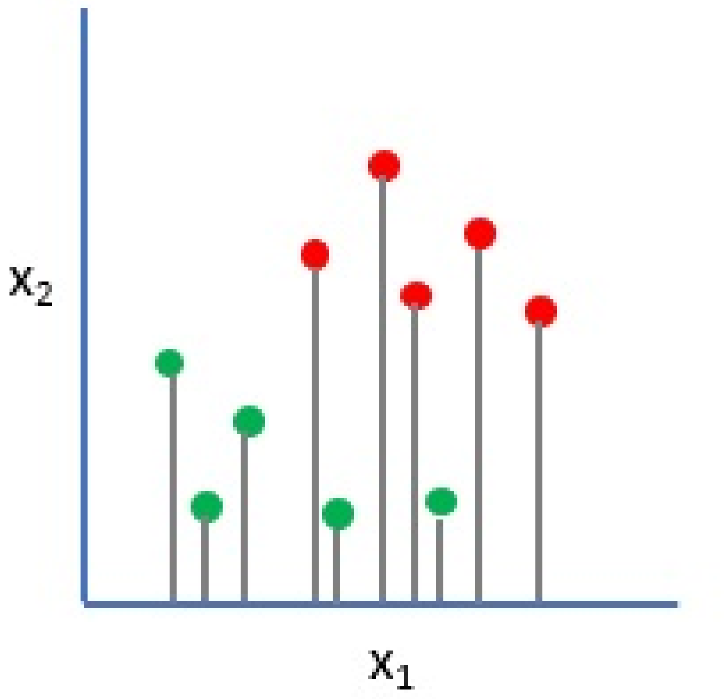


Figure 3.2: Dimensionality Reduction in LDA: Project data to X Axis

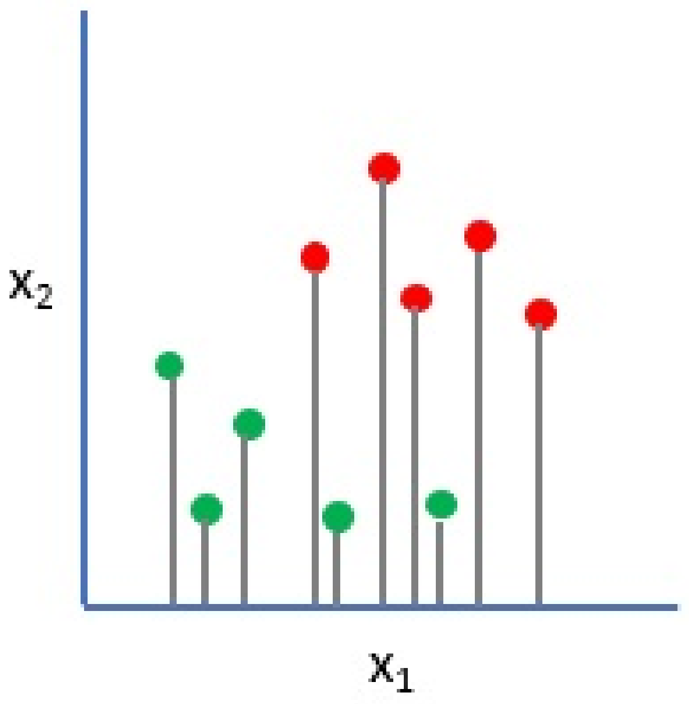


Figure 3.3: Dimensionality Reduction in LDA: Project to X Axis Result

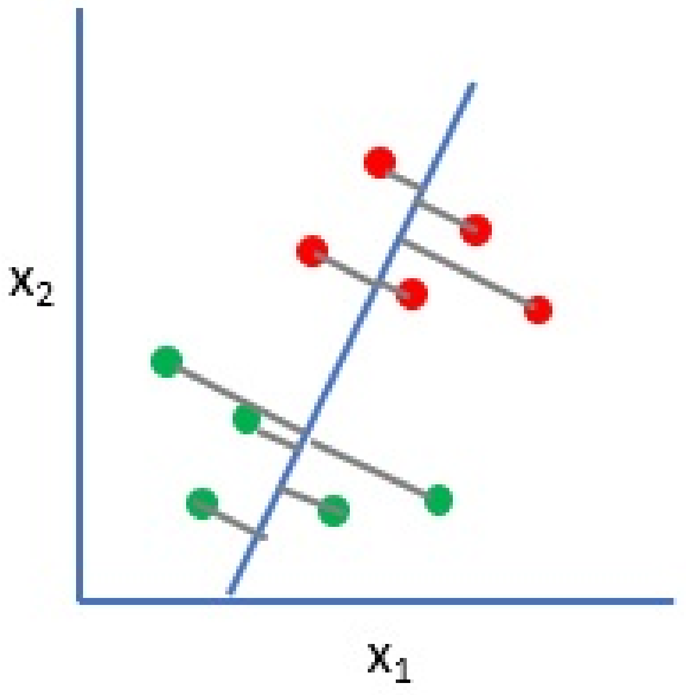


Figure 3.4: Dimensionality Reduction in LDA: Project data to New Axis

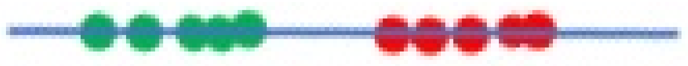


Figure 3.5: Dimensionality Reduction in LDA: Project to New Axis Result

Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a dimensionality reduction technique for supervised classification problems. Dimensionality reduction is a process of reducing the number of features in a data set by removing redundant and dependent features. Dimensionality reduction is important for better understanding and presentation of a data set. Dimensionality reduction also enhances the performance of classification process. Figure 3.1 shows a data set with 2 features x_1 and x_2 and 2 classes green and red. Figure 3.2 shows a projection of this data set to x_1 axis. This method doesn't clearly distinguish data into two classes as shown in Figure 3.3. This approach ignores useful information from feature x_2. Figure 3.4 shows that LDA finds a new axis for projecting data set and achieves a good result as shown in Figure 3.5.

LDA algorithm performs classification by maximizing the separability between classes, i.e. the distance between the mean of different classes, and minimizing the variation within each class.

Suppose the data set has C classes, μ_i is the mean vector of class i, M_i is the number of samples within class i, i = 1,2,3...,C , M is the total number of samples in the data set, and μ is the mean of the entire data set.

$$M = \sum_{i=1}^{C} M_i \tag{3.8}$$

$$\mu = \frac{1}{C}\sum_{i=1}^{C}\mu_i \tag{3.9}$$

The separability between classes, or between-class matrix is calculated as in Equation 3.10.

$$S_b = \sum_{i=1}^{C}(\mu_i - \mu)(\mu_i - \mu)^T \tag{3.10}$$

The within-class matrix is calculated as shown in Function 3.11.

$$S_w = \sum_{i=1}^{C}\sum_{j=1}^{M_i}(x_{i,j} - \mu_i)(x_{i,j} - \mu_i)^T \tag{3.11}$$

K-Nearest Neighbors (KNN)

K-Nearest neighbors algorithm is a "lazy learning" model due to the fact that this model only stores training data. There is no actual training stage in this model. When a prediction needs to be performed, the model will use stored training data for computing k-nearest neighbors around the point being classified. K and distance metrics will need to be defined for computing k-nearest neighbors. Then the majority class of k-nearest neighbors will be used to determine the class of the query point. The followings are the most commonly used distance metrics.

- Euclidean distance is calculated as below:

$$d(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.12}$$

- Manhattan distance is calculated as below:

$$d(X,Y) = \sum_{i=1}^{n}|x_i - y_i| \tag{3.13}$$

- Minkowski distance is calculated as below:

$$d(X,Y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{(1/p)} \tag{3.14}$$

- Hamming distance is used when comparing two binary strings with the same length. The Hamming distance between two binary strings is the number of bit positions where the bits are different.

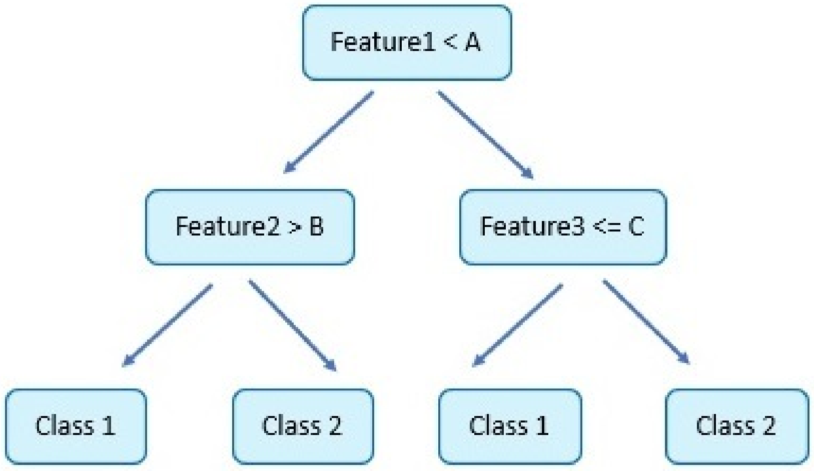


Figure 3.6: An Example of Decision Tree Classifier

Decision Tree Classifier

Decision tree classifier is a supervisor machine learning algorithm that simply makes a classification decision based on a set of rules on features of input data. During training stage, a decision tree is generated from training data. Figure 3.6 shows an example of a decision tree classifier with 3 features and 2 classes.

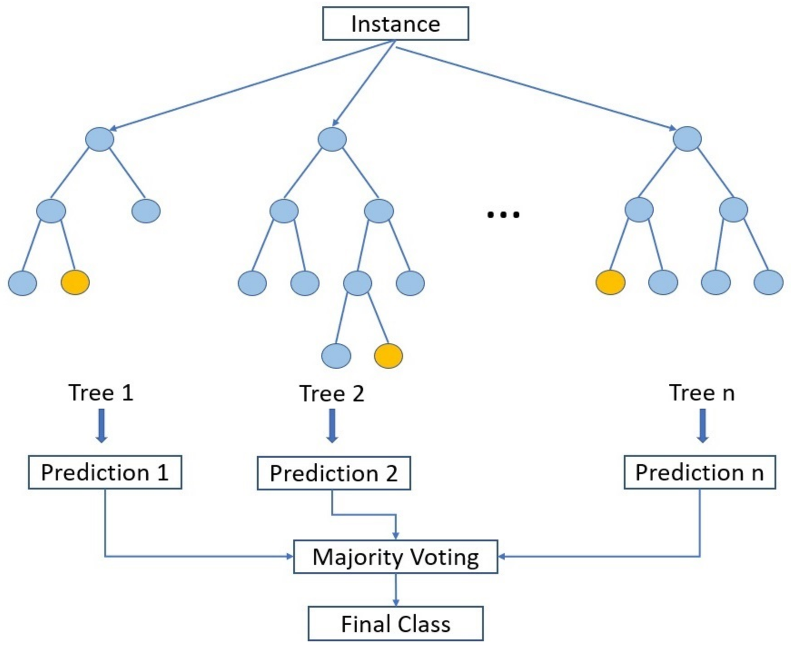


Figure 3.7: Random Forest Concept

Random Forest

Random forest is a supervisor machine learning algorithm that can be used for both classification and regression problems. Random forest algorithm contains multiple decision trees. Each decision tree takes a subset of a given data set and produces a predicted output. The random forest algorithm takes the predicted output from each decision tree and votes for majority to produce a final prediction. The concept of random forest algorithm is illustrated in Figure 3.7.

Support Vector Machines

Support vector machines (SVM) is a machine learning algorithm that is a supervised machine learning algorithm used for classification and/or regression. SVM is used to find a hyperplane that classifies the type of data. When the number of features is 2, the hyperplane is a line. When the number of features is 3, the hyperplane is a plane. Figure 3.8 illustrates hyperplanes in SVM. The objective of SVM is finding a hyperplane such that the margin is maximized. With a large margin, future data points can be classified with more accuracy. Figure 3.9 illustrates small vs. large margin in SVM.

3.2 Experimental Setup

We implemented the proposed method of offloading computation from the edge layer to the equipment layer. We conducted a series of experiments with different factors that can have bearing on the performance of computation at the equipment layer. Then we compared the performance at the equipment layer with that at the edge layer.

To carry out experiments, we set up a system that includes a server for edge computing and a computer for equipment computing. The edge computer has an AMD Threadripper Pro 3975WX 3.5 GHz 32 cores processor, 512 GB memory, 2TB SSD NVMe for the operating system, and 2TB SSD SATA for data. The operating system on the edge server is Ubuntu 20.04. The equipment computer is a regular computer with Windows 10 Professional, Intel i7 8 cores 2.80 GHz CPU, 16GB memory, and 1TB hard drive.

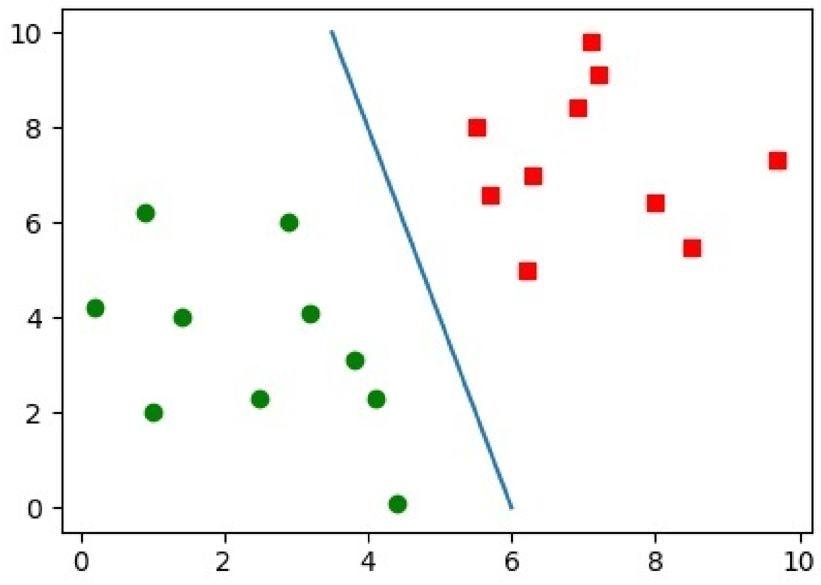
The experiments aim to determine the factors or combination of thereof that can deliver better performance at the equipment layer.

- Data size: the number of data points which are produced for each material (for example, wafer) during processing. The data size we used for experiments range from 2000 data points to 50000 data points.

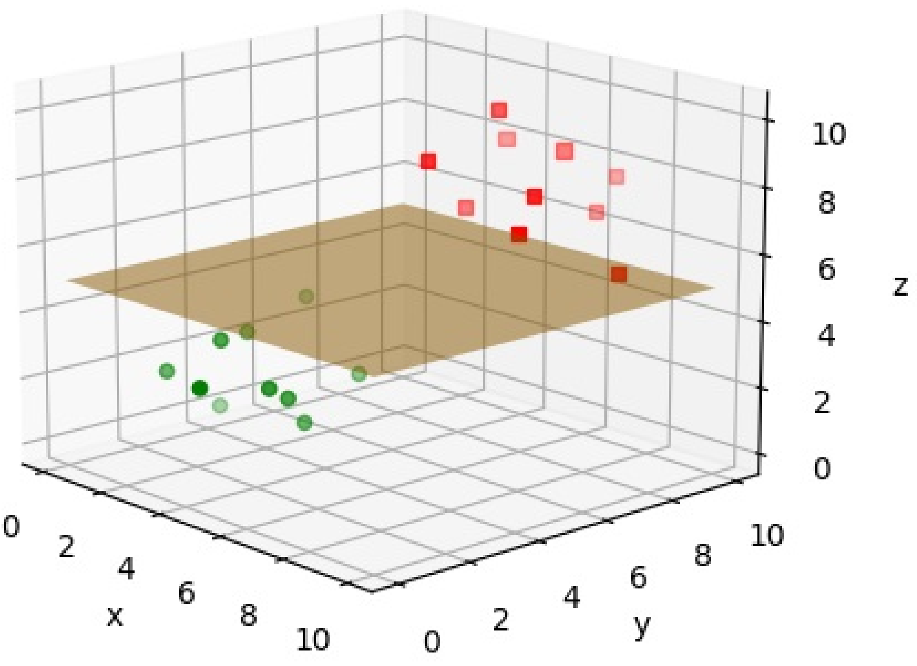
- CPU utilization: the current utilization of the CPU during analytic computation (excluding the utilization that is used for performing analytic computation).

- Algorithm complexity: We conducted the experiments with different algorithmic complexities: O(n), O(nlogn), and O(n^2)

- Edge or Equipment: We performed experiments on the edge server and equipment computer and compared the results.

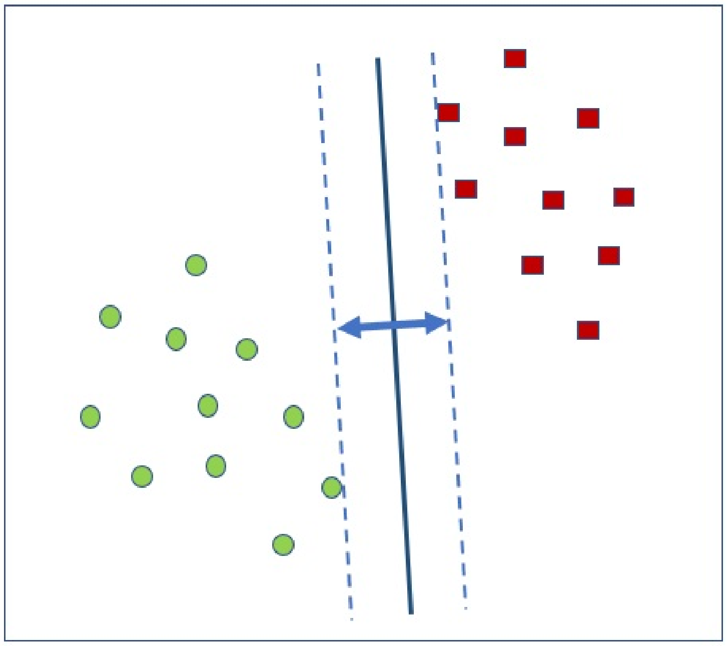


(a) SVM Hyperplane With 2 Features

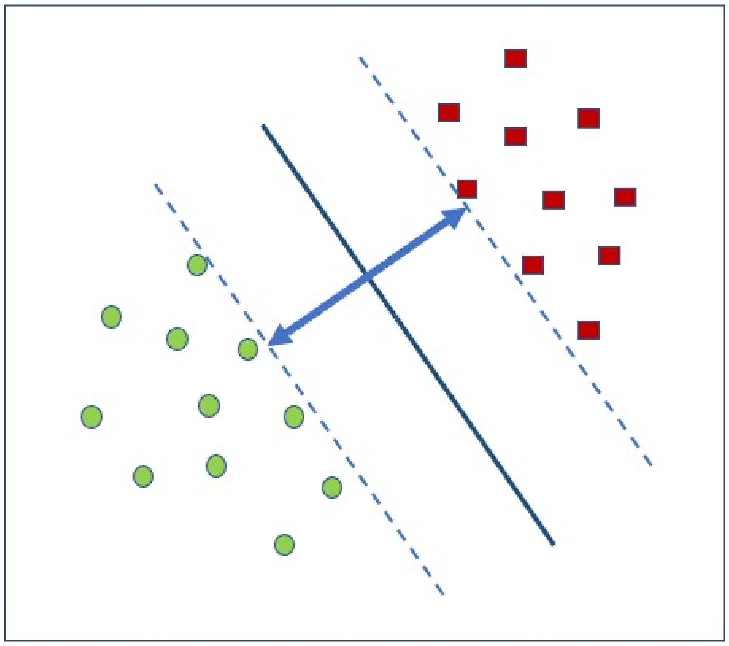


(b) SVM Hyperplane With 3 Features

Figure 3.8: SVM Hyperplane



(a) Small Margin



(b) Large Margin

Figure 3.9: SVM Margin

For each algorithmic complexity, we measured computing time with different data sizes, from 2000 data points to 50000 data points. The experiments were conducted on the edge server and on the equipment computer. For the equipment computer, we measured computing time with different percentages of CPU utilization. Total computing time is measured for each test case.

3.3 Experimental Results

We conduct extensive experiments to evaluate the performance and effectiveness of our offloading strategy. In this part of the study, we shed light on the impacts of numerous factors on computing performance. In particular, we first investigate the impacts of the algorithmic complexity of computing tasks. Then, we examine the influence of the CPU utilization of equipment computers, followed by the examination of the performance factor of data size[80]. Finally, we present a computing offload algorithm that takes into factors algorithmic complexity, CPU utilization, memory usage to make offloading decisions, thereby minimizing computing time.

3.3.1 Algorithmic Complexity of Computing Tasks

In the first group of experiments, we investigate the impacts of the time complexity (i.e., Big O) of computing tasks on computing time. We study three computational tasks with different time complexities, namely, O(n), O(nlogn), and O(n^2). We compare the computing time of the three computational tasks running on the edge layer against that on the equipment layer. To thoroughly study the equipment performance, we vary the CPU utilization of the equipment computer when performing the computational tasks.

Computational Tasks with O(n)

Fig. 3.10 shows the computing time for O(n) tasks running on the device computer and the edge server with respect to different numbers of data points. It indicates that performing

O(n) computing tasks on the equipment computer saves more time than performing the tasks on the edge server. These results are expected. The reason is that O(n) computing tasks can be completed in a very short time. For example, with 2000 data points, O(n) computing tasks can be done in 0.0214 seconds on the equipment computer even when the CPU utilization is as high as 99%. In contrast, data needs to be transferred from the equipment layer to the edge layer first in order to perform computing tasks on the edge layer. In this case, data transfer contributes most of the latency in edge computing due to the short computation time of O(n) tasks.

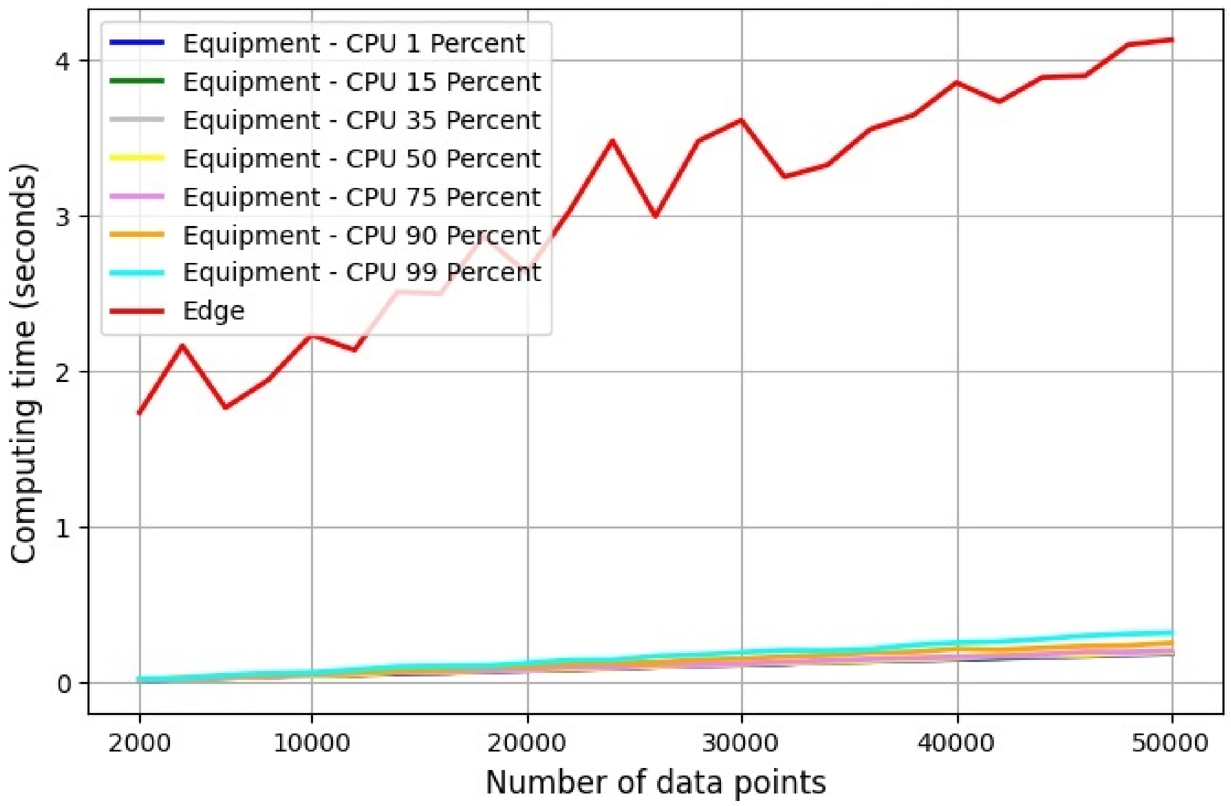


Figure 3.10: O(n) Performance

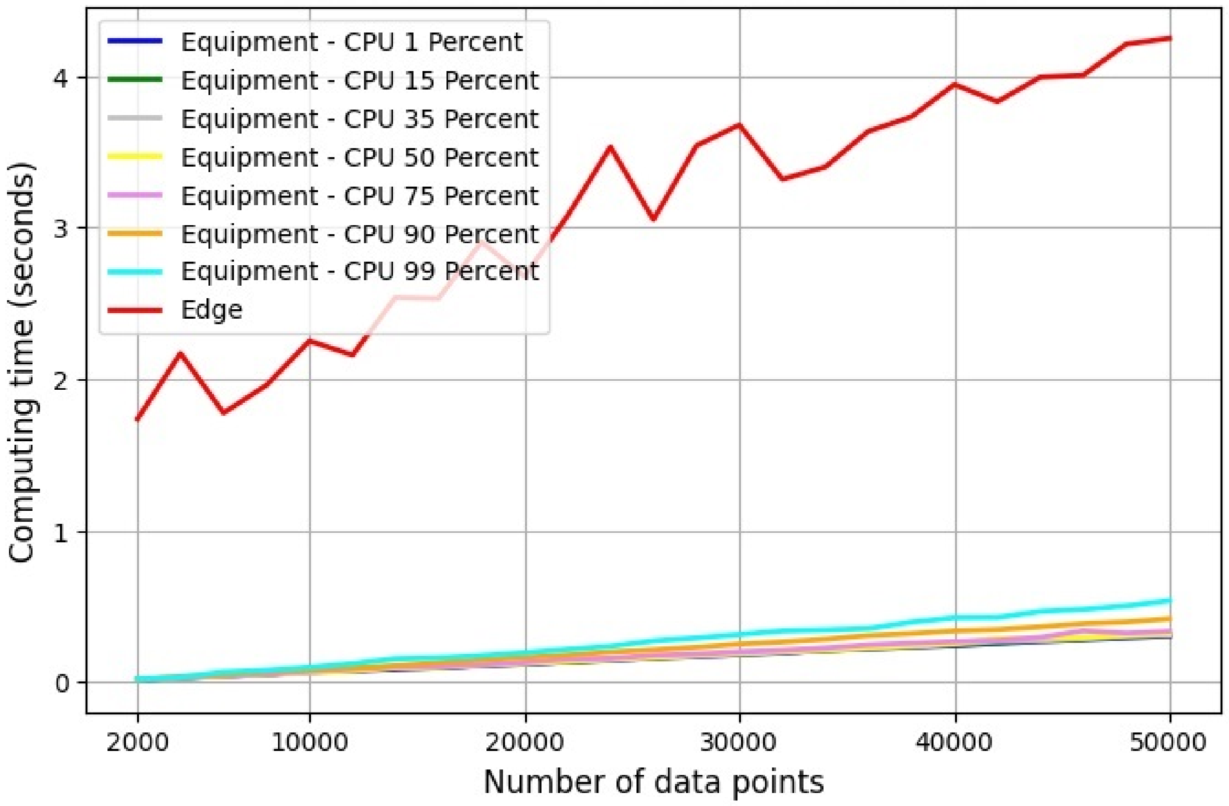


Figure 3.11: O(nlogn) Performance

Computational Tasks with O(nlogn)

The experimental results of the O(nlogn) case plotted in Fig. 3.11 are similar to the results illustrated in Fig. 3.10. That is, the computing time of the O(nlogn) computational tasks running on the equipment computer is shorter than that running on the edge server. Although the computing time of the O(nlogn) computational tasks increases slightly compared to the O(n) tasks, data transfer from the equipment layer to the edge layer still dominates the total computing time. For example, even when the CPU utilization is 99%, the computing time is still under one second.

Computational Tasks with O(n^2)

However, when it comes to the computational tasks with the time complexity of O(n^2), the edge server outperforms the equipment computer in terms of computing time (see Fig.

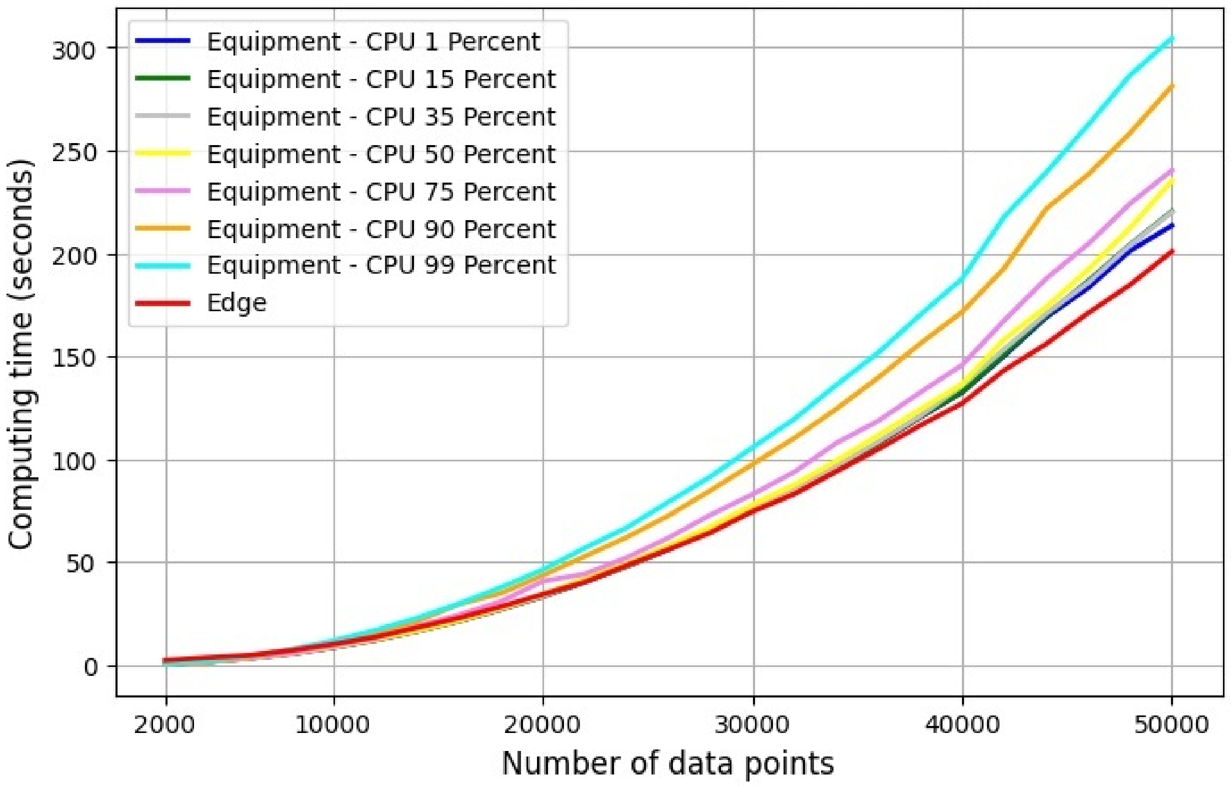


Figure 3.12: O(n^2) Performance

3.12). There exists two main reasons behind this observation. First, it takes much longer time for O(n^2) computational tasks to be completed. Second, when performing O(n^2) tasks, the number of data points plays a significant role in computing time. Specifically, the computing time increases exponentially as data size increases. Unlike the O(n) and O(nlogn) cases above, data transfer from the equipment layer to the edge layer in this case becomes an insignificant factor in computing time.

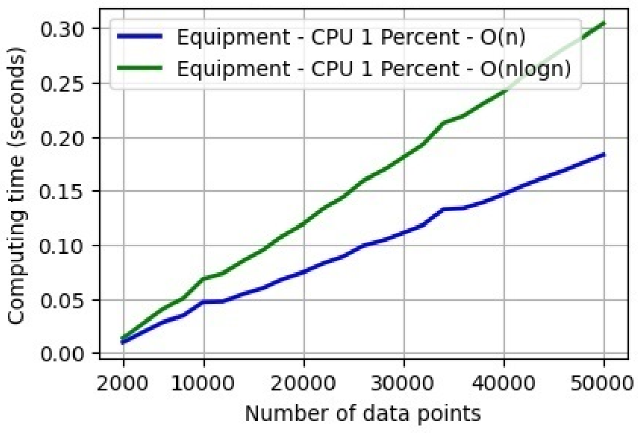
Performance Comparison Between Tasks of O(n), O(nlogn), and O(n^2)

To investigate the impacts of CPU utilization on computing time, we compare the computing performance between tasks of O(n), O(nlogn), and O(n^2) on the equipment computer. Fig. 3.13 shows the comparison in computing time between O(n) and O(nlogn) tasks, O(n) and O(n^2) tasks, and O(nlogn) and O(n^2) tasks, running on the equipment computer with its CPU utilization set to 1%. It demonstrates that the computing time of both O(n) and O(nlogn) tasks goes up linearly as the data size increases. However, the total computing time is still very small. On the other hand, the computing time of O(n^2) tasks grows exponentially as the data size increases. We can see the same trends when CPU utilization is set 50%, 90%, and 99% in Fig. 3.14, Fig. 3.15, and Fig. 3.16 respectively.

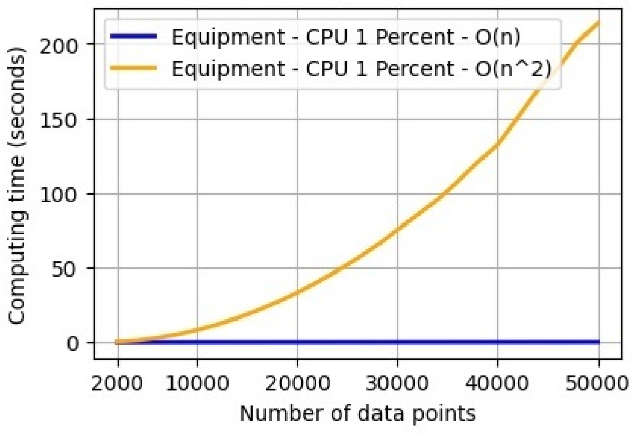
3.3.2 CPU Utilization

In this set of experiments, we evaluate the impacts of CPU utilization on computing time. We measure the computing time of O(n), O(nlogn) and O(n^2) tasks under different CPU utilization. We set the number of data points at 10000, 30000 and 50000.

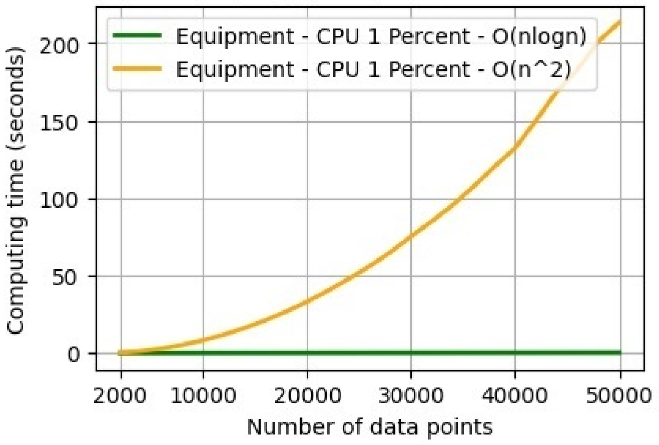
Fig. 3.17 plots the computing time of O(n), O(nlogn) and O(n^2) tasks with respect to different CPU utilization. It shows that when the CPU is busy for other tasks, it takes a longer time for the computational tasks to complete. And in the case of 10000 data points shown in Fig. 3.17a, the computing time increases exponentially after CPU utilization becomes higher than 50%. We also see similar trends in the 30000 and 50000 data point cases (see Fig. 3.17b and Fig. 3.17c). That is, the computing time rises sharply when the CPU utilization reaches 75% in both Fig. 3.17b and 3.17c. These ascending trends in computing time are expected and exist for two main reasons. First, when the CPU utilization of the equipment computer is high, the CPU must switch to other tasks and cannot focus on the computational tasks, thereby taking a longer time for the CPU to perform the computational tasks. Second, with more data points fed to the CPU, the size of the data becomes larger, and thus, leading to a longer time to process the data. In short, CPU utilization has a significant impact on determining whether to offload computing tasks from the edge layer to the equipment layer.



(a) Performance O(n) vs O(nlogn)

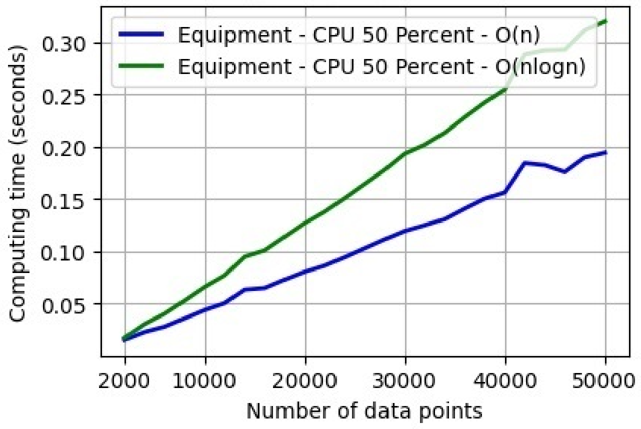


(b) Performance O(n) vs O(n^2)

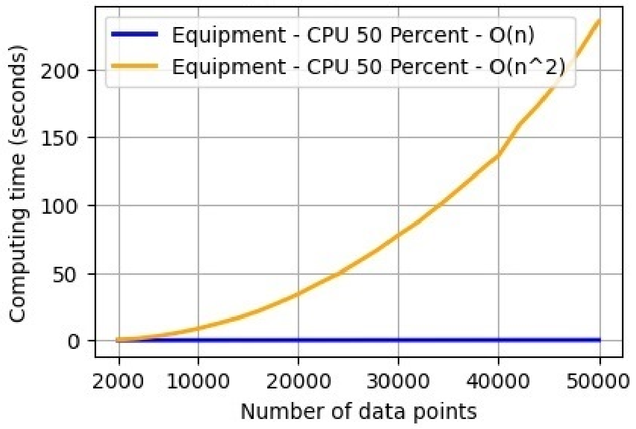


(c) Performance O(nlogn) vs O(n^2)

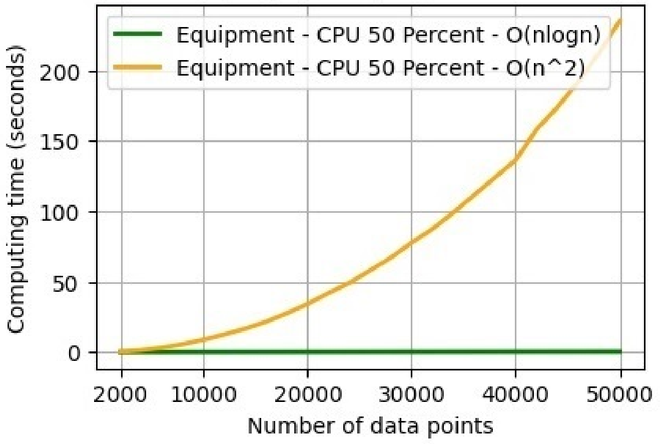
Figure 3.13: The impact of algorithmic complexity on performance of equipment computer with 1% CPU utilization



(a) Performance O(n) vs O(nlogn)

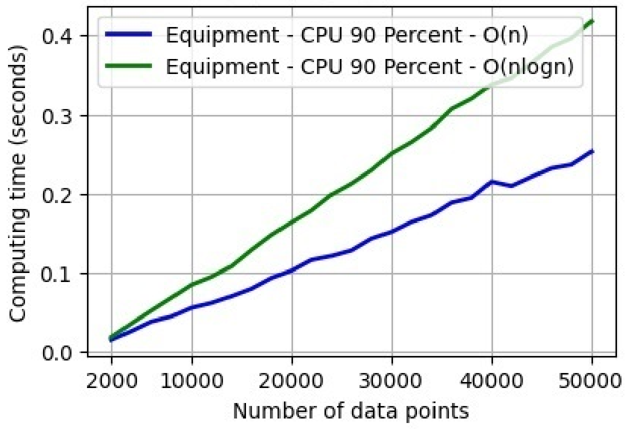


(b) Performance O(n) vs O(n^2)

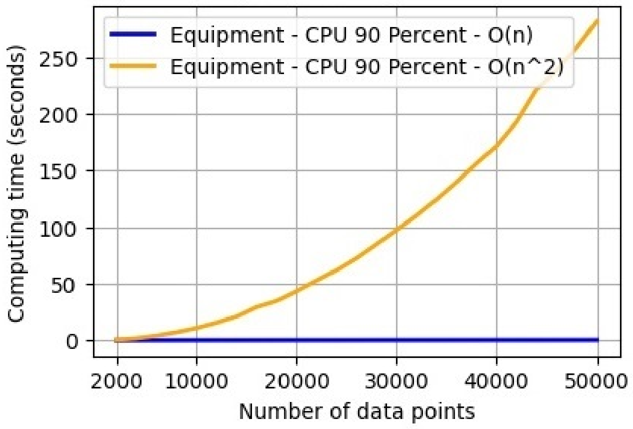


(c) Performance O(nlogn) vs O(n^2)

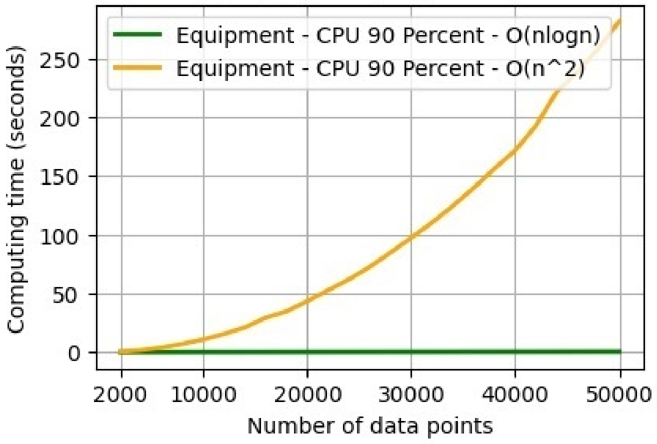
Figure 3.14: The impact of algorithmic complexity on performance of equipment computer with 50% CPU utilization



(a) Performance O(n) vs O(nlogn)

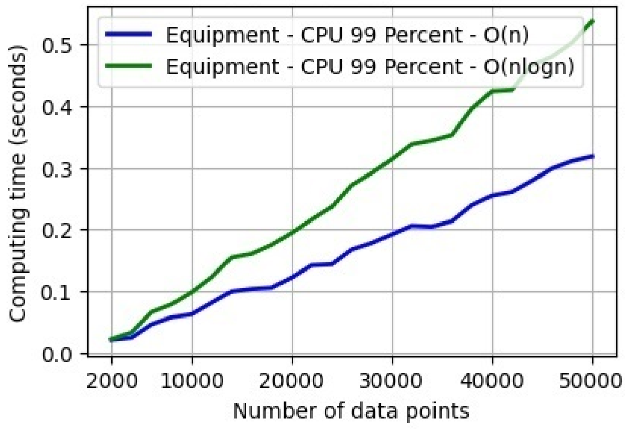


(b) Performance O(n) vs O(n^2)

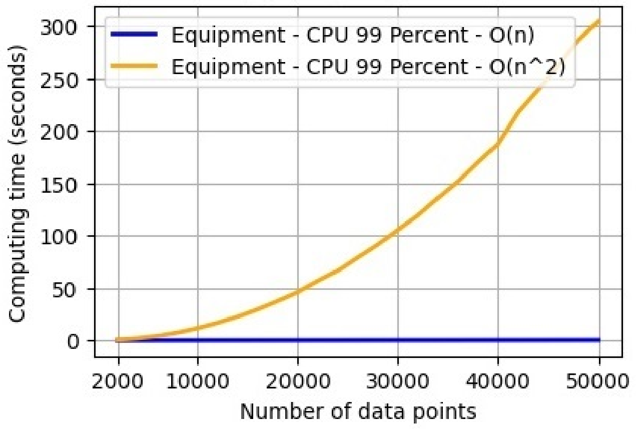


(c) Performance O(nlogn) vs O(n^2)

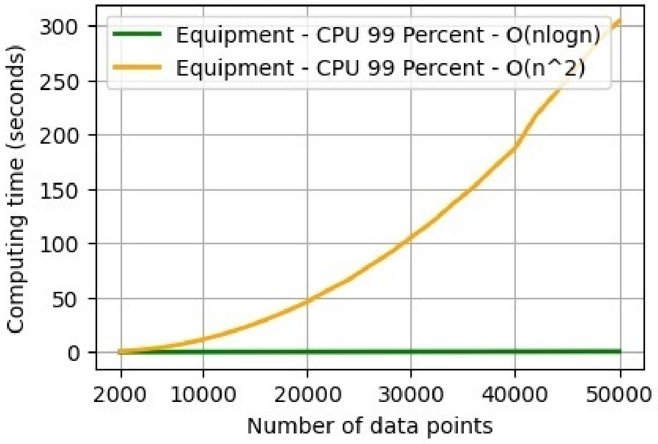
Figure 3.15: The impact of algorithmic complexity on performance of equipment computer with 90% CPU utilization



(a) Performance O(n) vs O(nlogn)



(b) Performance O(n) vs O(n^2)



(c) Performance O(nlogn) vs O(n^2)

Figure 3.16: The impact of algorithmic complexity on performance of equipment computer with 99% CPU utilization

3.3.3 Data Size

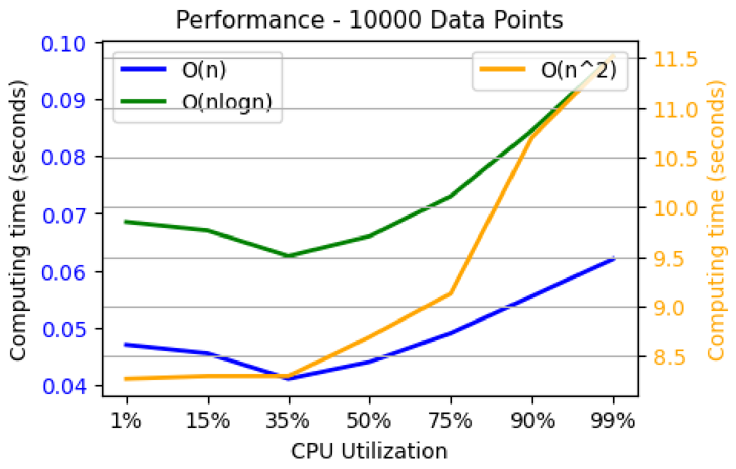
Now we are in the position of investigating the impacts of data size on the computing time. Again, we measure the computing times of O(n), O(nlogn) and O(n^2) tasks with respect to the number of data points. We carry out the experiments under four CPU utilization conditions of the equipment computer, ranging from 1% to 50%, 90%, and 99%. The experimental results are plotted in Fig. 3.13, Fig. 3.14, Fig. 3.15 and Fig. 3.16, respectively.

We can observe from Fig. 3.13 - Fig. 3.16 that regardless of the time complexity, the computing time grows as the data size increases. And among these three computational tasks of different complexities, the curves of O(n) and O(nlogn) tasks grow relatively linearly. In contrast, the computing time of O(n^2) task soar exponentially. For instance, Fig. 3.15 shows that the curves of O(n) and O(nlogn) tasks both ascend relatively linearly, with the O(nlogn) curve higher than the O(n) curve. However, compared with the curve of O(n^2) task, O(n) and O(nlogn) tasks exhibit flat curves, being much smaller than that of O(n^2) task. These phenomena exist because the increase in the number of data points cause the CPU to take a longer time to process.

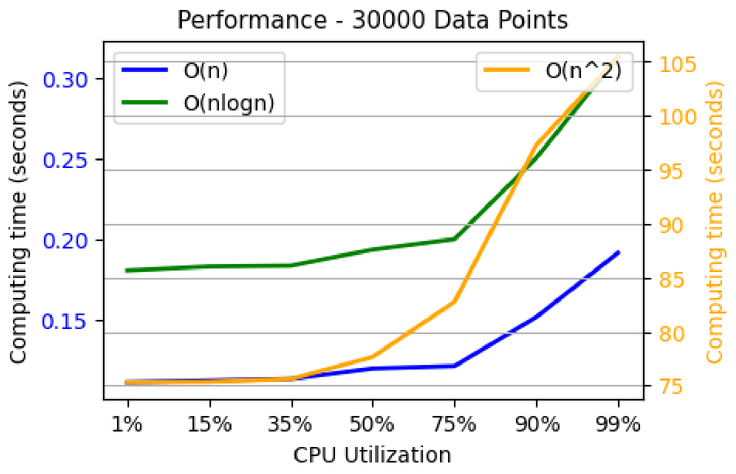
In a nutshell, the number of data points serves as a non-negligible factor in computing time. The larger the number of data points, the longer the computing time. The data used in this section can be found in Appendix A.

3.3.4 Algorithm Design

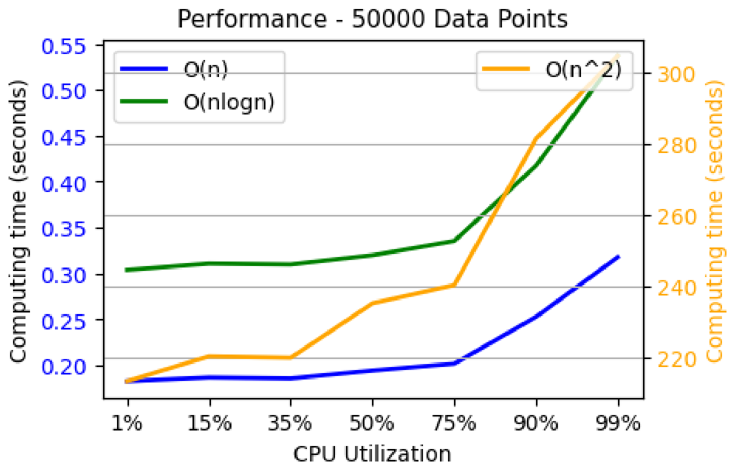
In this part of the study, we propose our computing offload algorithm that facilitates making decisions on when to offload heavy tasks from the edge layer to the equipment layer. In order for the equipment computer to perform its functionalities, such as controlling the equipment and producing data, it is vital to preserve certain computing resources (e.g., CPU time and memory space) and avoid pushing the equipment computer to its maximum limits. Our observations and analysis above reveal that the most significant factors that affect computing time are the time complexity of computational tasks and the CPU utilization of the



(a) Performance - 10000 Data Points



(b) Performance - 30000 Data Points



(c) Performance - 50000 Data Points

Figure 3.17: The impact of CPU utilization on performance of equipment computer

Algorithm 1 Computing Offload Algorithm

```
Input:
CPU_Threshold
RAM_Threshold
Current_CPU_Utilization
Current_RAM_Utilization
 1: if $Current\_CPU\_Utilization > CPU\_Threshold$ then
 2:     Computing on edge layer
 3:     Return
 4: end if
 5:
 6: if $Current\_RAM\_Utilization > RAM\_Threshold$ then
 7:     Computing on edge layer
 8:     Return
 9: end if
10:
11: if $Complexity >= O(n^2)$ then
12:     Computing on edge layer
13: else
14:     Computing on equipment layer
15: end if
```

equipment computer. For example, our experimental results in Section 3.3.1, Section 3.3.2, and Section 3.3.3 indicate that O(n^2) tasks perform better at the edge layer, whereas O(n) and O(nlogn) tasks perform better at the equipment layer.

Taking into account the three significant factors: CPU utilization, memory usage, and time complexity, we propose an algorithm to determine whether to offload a specific computation task to the equipment layer or perform the task on the edge layer, aiming to minimize computing time.

The computing offload algorithm outlined in Algorithm 1 first checks the current CPU utilization of the equipment computer; if it is greater than a pre-defined CPU threshold, the computational task will be performed on the edge layer (see Lines 1-3 in Algorithm 1). Next, the computing offload algorithm examines the current memory usage; if it is greater than a pre-defined memory threshold, the computational task will be performed on the edge layer (see Lines 6-9 in Algorithm 1). Lastly, the computing offload algorithm looks into the time complexity of the computational task; if the time complexity is bigger than $O(n^2)$, the computational task will be executed on the edge layer, otherwise, the computational task will be offloaded and performed on the equipment computer (see Lines 11-15 in Algorithm 1).

3.4 Using machine learning to determine whether to offload computation from the edge layer to the equipment layer

3.4.1 Purpose of the Evaluation With Machine Learning Models

The purpose of using machine learning models is to evaluate the validity our recommendation to offload computation tasks from the edge to the equipment layer. The machine learning models can also help to determine when it is efficient to offload computation. Lastly, after training and testing machine learning algorithms, we can choose the best machine learning model and compare it with Algorithm 1. Figure 3.18 shows a generic machine learning training and testing flow.

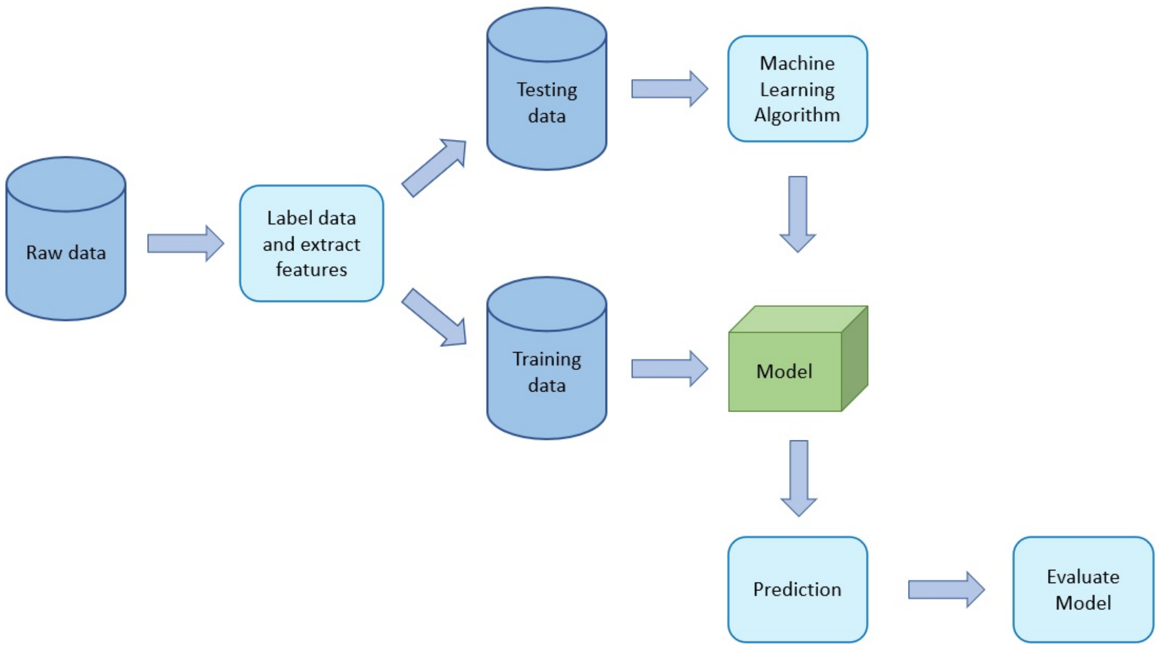


Figure 3.18: Machine Learning Flow

3.4.2 Experimental Setup

As discussed in section 3.3, the factors that affect the computation total time are data size, CPU utilization, and algorithmic complexity of the computation task. Obviously, CPU speed also affects the computing time. The list of factors includes:

- Data size (number of data points)
- CPU utilization (percentage)
- CPU speed (GHz, we have only one CPU speed due to limited resource, but we put it here for heterogeneous platform)
- Algorithmic complexity (O(n), O(nlogn), or O(n^2)

Table 3.1: Important Factors in Computation Time

Data Size	**CPU Utilization**	**CPU Speed**	**Algorithmic Complexity**
200	1	2.8	O(n)
400	1	2.8	O(n)
...	...	...	...
50000	1	2.8	O(n)
200	1	2.8	O(nlogn)
400	1	2.8	O(nlogn)
...	...	...	...
50000	1	2.8	O(n^2)
200	1	2.8	O(n^2)
400	1	2.8	O(n^2)
...	...	...	...
50000	1	2.8	O(n^2)

Continued on next page

Table 3.1 – continued from previous page

Data Size	CPU Utilization	CPU Speed	Algorithmic Complexity
200	15	2.8	O(n)
400	15	2.8	O(n)
...	...	...	...
50000	15	2.8	O(n)
200	15	2.8	O(nlogn)
400	15	2.8	O(nlogn)
...	...	...	...
50000	15	2.8	O(nlogn)
200	15	2.8	O(n^2)
400	15	2.8	O(n^2)
...	...	...	...
50000	15	2.8	O(n^2)
...	...	...	...
...	...	...	...
...	...	...	...
200	99	2.8	O(n)
400	99	2.8	O(n)
...	...	...	...
50000	99	2.8	O(n)
200	99	2.8	O(nlogn)
400	99	2.8	O(nlogn)
...	...	...	...
50000	99	2.8	O(nlogn)
200	99	2.8	O(n^2)

Continued on next page

Table 3.1 – continued from previous page

Data Size	**CPU Utilization**	**CPU Speed**	**Algorithmic Complexity**
400	99	2.8	O(n^2)
...	...	...	...
48000	99	2.8	O(n^2)
50000	99	2.8	O(n^2)

Table 3.1 shows details of the factors in calculating total computation time. We use those factors as features to feed into machine learning models that determine whether to offload computing tasks from the edge layer to the equipment layer. Data size, CPU utilization, and CPU speed are numerical values which can be used as features. The algorithmic complexity feature has three values, O(n), O(nlogn), and O(n^2). We use one-hot encoding method to convert the feature Algorithmic Complexity into three features: O(n), O(nlogn), and O(n^2). After converting, our feature list includes:

- Data size (number of data points)
- CPU utilization (percentage)
- CPU speed (GHz)
- O(n): 1 or 0. If O(n) is 1, O(nlogn) and O(n^2) must be zero.
- O(nlogn): 1 or 0. If O(nlogn) is 1, O(n) and O(n^2) must be zero.
- O(n^2): 1 or 0. If O(n^2) is 1, O(n) and O(nlogn) must be zero.

We create an Offload label to indicate whether the computation task should be offloaded from the edge layer to the equipment layer. Offload can be labeled as 1 or 0. If Offload is 1, it is efficient to offload computation task from the edge to the equipment. Otherwise, the value of Offload is 0. We use total computing time data that we collect in our experiments to label data. If total computing time on the equipment is less than total computing time on the edge server, Offload is labeled as 1. If total computing time on the equipment is greater than or equal to total computing time on the edge server, it is not necessary to offload computing task to the equipment, Offload is labeled as 0. Table 3.2 shows data format we use for training machine learning models.

Table 3.2: Important Factors in Computation Time

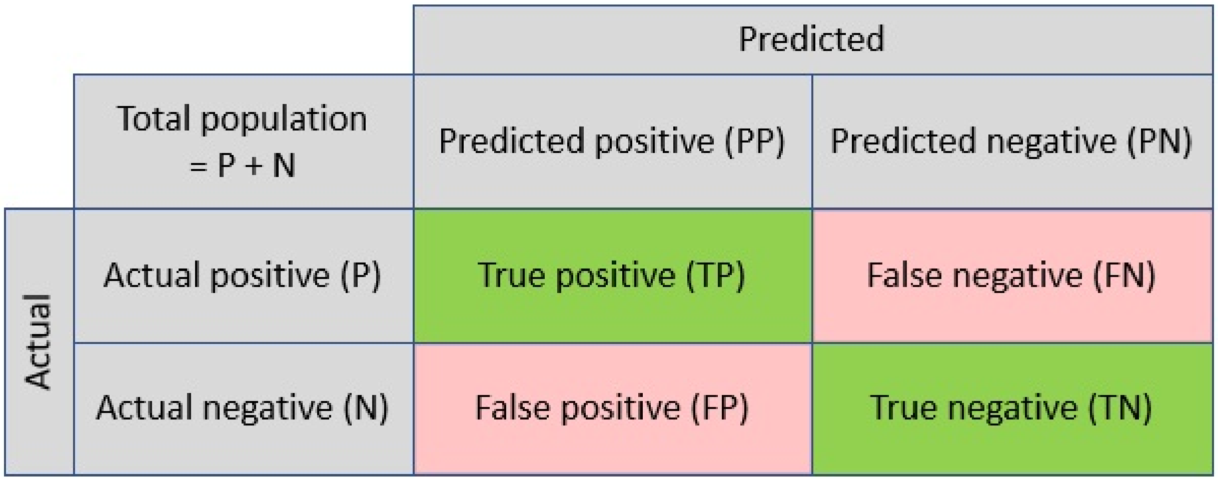
Data Size	**CPU Utilization**	**CPU Speed**	**O(n)**	**O(nlogn)**	**O(n^2)**	**Offload**
200	1	2.8	1	0	0	1
400	1	2.8	1	0	0	1
...	...	...	...	...	...	...
50000	1	2.8	1	0	0	1
200	1	2.8	0	1	0	1
400	1	2.8	0	1	0	1
...	...	...	...	...	...	...
50000	1	2.8	0	1	0	1
200	1	2.8	0	0	1	1
400	1	2.8	0	0	1	1
...	...	...	...	...	...	...
50000	1	2.8	0	0	1	0
200	15	2.8	1	0	0	1
400	15	2.8	1	0	0	1
...	...	...	...	...	...	...
50000	15	2.8	1	0	0	1
200	15	2.8	0	1	0	1
400	15	2.8	0	1	0	1
...	...	...	...	...	...	...

Continued on next page

Table 3.2 – continued from previous page

Data Size	**CPU Utilization**	**CPU Speed**	**O(n)**	**O(nlogn)**	**O(n^2)**	**Offload**
50000	15	2.8	0	1	0	1
200	15	2.8	0	0	1	1
400	15	2.8	0	0	1	1
...	...	...	...	...	...	...
50000	15	2.8	0	0	1	0
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
200	99	2.8	1	0	0	1
400	99	2.8	1	0	0	1
...	...	...	...	...	...	...
50000	99	2.8	1	0	0	1
200	99	2.8	0	1	0	1
400	99	2.8	0	1	0	1
...	...	...	...	...	...	...
50000	99	2.8	0	1	0	1
200	99	2.8	0	0	1	1
400	99	2.8	0	0	1	1
...	...	...	...	...	...	...
48000	99	2.8	0	0	1	0
50000	99	2.8	0	0	1	0

We use Stratified K-Fold function in sklearn python library to split the data set into training and testing data. We split the data set 10 times, each time the Stratified K-Fold function creates a different split into training and testing data sets. The reason we use Stratified K-Fold is to preserve the percentage of samples for each class in the Offload column. In other words, the ratio between offload and non-offload cases in the training data is the same as that of testing data. For each split of data, we apply machine learning algorithms on the training data set to build a model for each algorithm. Then we use the models to predict the outcome of the testing data set.



		Predicted	
	Total population = P + N	Predicted positive (PP)	Predicted negative (PN)
Actual	Actual positive (P)	True positive (TP)	False negative (FN)
	Actual negative (N)	False positive (FP)	True negative (TN)

Figure 3.19: Generic confusion matrix

3.4.3 Evaluation Method

We conduct training and testing of our data set on various machine learning algorithms, including logistic regression, linear discriminant analysis, k-nearest neighbors classifier, decision tree classifier, random forest, and support vector machines. It is worth noting that since we have only one CPU speed in our dataset, it is not necessary to include CPU speed in the feature list for this evaluation. For heterogeneous platforms with different types of computers, CPU speeds should be included in the feature list.

For each machine learning algorithm, we feed the training data and build a model. We use the model to predict the outcome of the testing data. Then we evaluate the performance of the machine learning model by comparing the prediction with the actual labels of the testing data set. To evaluate the performance of a machine learning model, we collect the following parameters for each run and calculate the average values:

- Confusion matrix
  - True positive
  - True negative
  - False positive
  - False negative
- Precision
- Recall
- Accuracy score

A confusion matrix is a table that is used to define the performance of a classification algorithm. A generic confusion matrix is shown in Figure 3.19. The main components of the confusion matrix are true positive (TP), true negative (TN), false positive (FP), false negative (FN).

The accuracy score is calculated as follows:

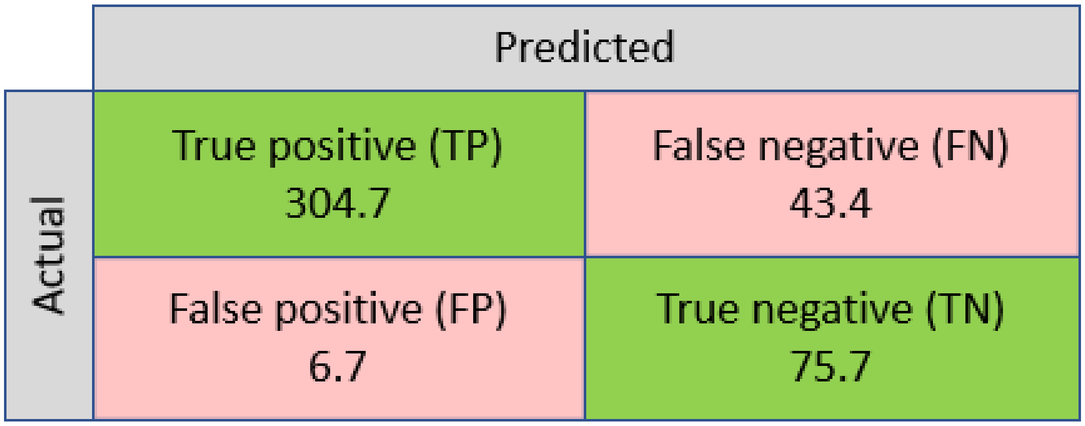
$$Accuracy\ score = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.15}$$

The precision value is calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3.16}$$

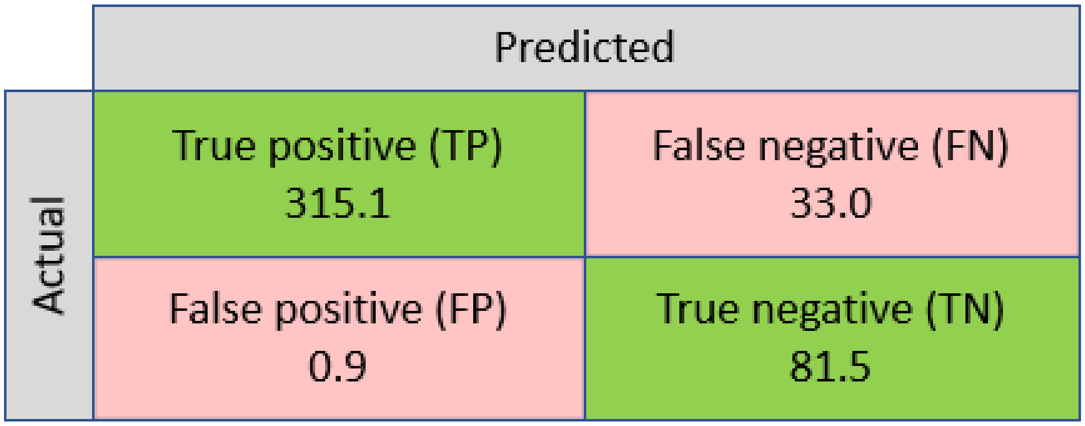
The recall value is calculated as follows:

$$Recall = \frac{TP}{TP + FN} \tag{3.17}$$



	Predicted	
Actual	True positive (TP) 304.7	False negative (FN) 43.4
	False positive (FP) 6.7	True negative (TN) 75.7

Figure 3.20: Confusion Matrix from Logistic Regression Model



	Predicted	
Actual	True positive (TP) 315.1	False negative (FN) 33.0
	False positive (FP) 0.9	True negative (TN) 81.5

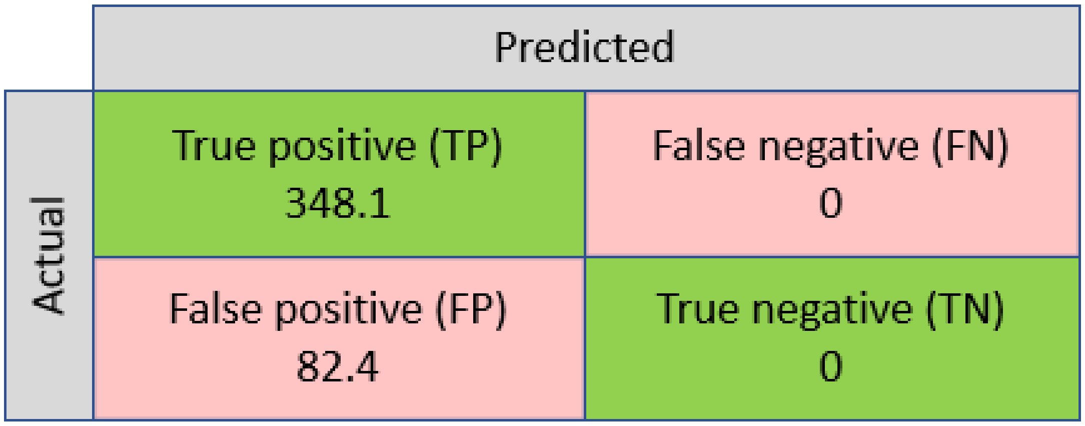
Figure 3.21: Confusion Matrix from Linear Discriminant Analysis Model

3.4.4 Observation

Logistic Regression

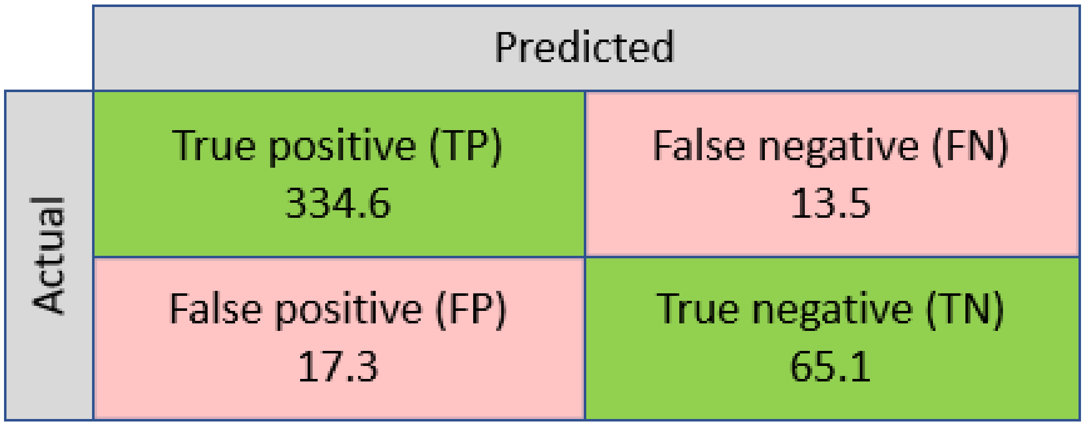
Figure 3.20 shows the confusion matrix when testing our data set with logistic regression model. From the confusion matrix, we calculate the following measures:

- Precision = 0.9811
- Recall = 0.8753
- Accuracy score = 0.8837



	Predicted	
Actual	True positive (TP) 348.1	False negative (FN) 0
	False positive (FP) 82.4	True negative (TN) 0

Figure 3.22: Confusion Matrix from K-Nearest Neighbors Classifier Model



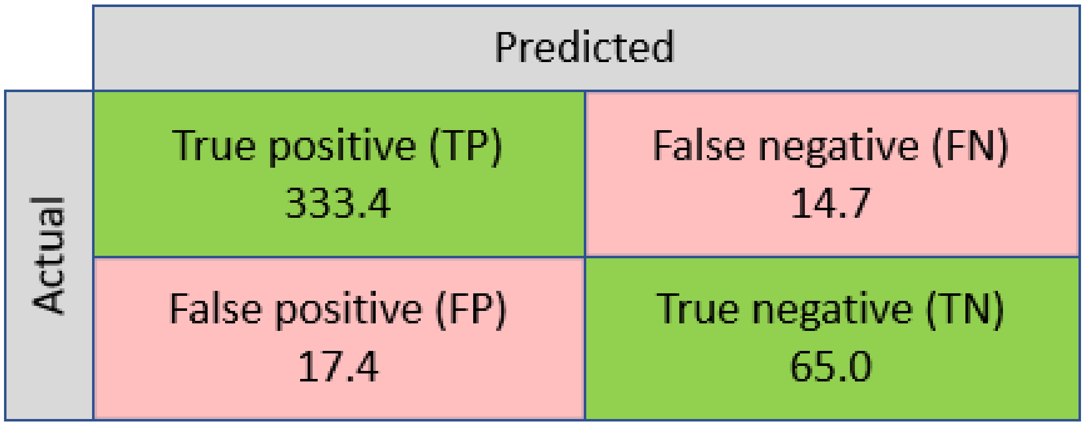
	Predicted	
Actual	True positive (TP) 334.6	False negative (FN) 13.5
	False positive (FP) 17.3	True negative (TN) 65.1

Figure 3.23: Confusion Matrix from Decision Tree Classifier Model

Linear Discriminant Analysis

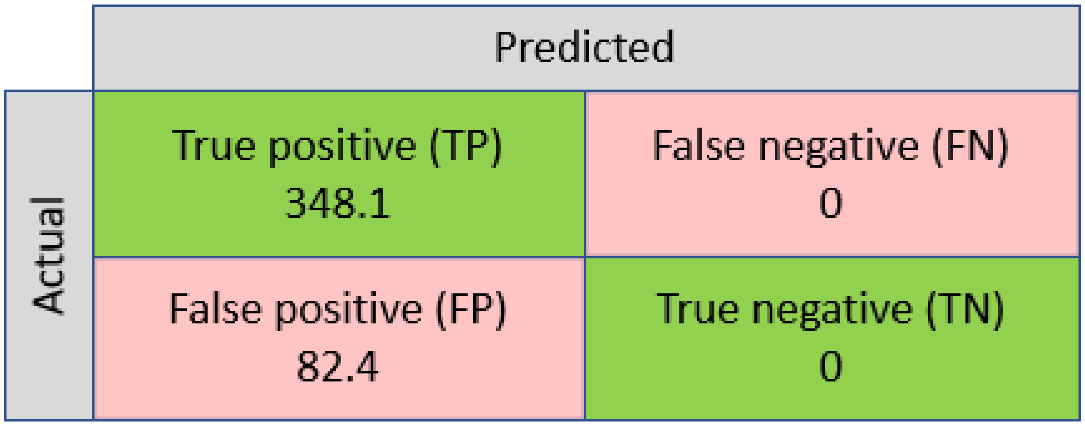
Figure 3.21 shows the confusion matrix when testing our data set with linear discriminant analysis model. From the confusion matrix, we calculate the following measures:

- Precision = 0.9975
- Recall = 0.9052
- Accuracy score = 0.9213



		Predicted	
Actual		True positive (TP) 333.4	False negative (FN) 14.7
		False positive (FP) 17.4	True negative (TN) 65.0

Figure 3.24: Confusion Matrix from Random Forest Model



		Predicted	
Actual		True positive (TP) 348.1	False negative (FN) 0
		False positive (FP) 82.4	True negative (TN) 0

Figure 3.25: Confusion Matrix from Support Vector Machines Model

K-Nearest Neighbors Classifier (KNN)

Figure 3.22 shows the confusion matrix when testing our data set with k-nearest neighbors classifier model. From the confusion matrix, we calculate the following measures:

- Precision = 0.8086
- Recall = 1
- Accuracy score = 0.8086

Decision Tree Classifier

Figure 3.23 shows the confusion matrix when testing our data set with decision tree classifier model. From the confusion matrix, we calculate the following measures:

- Precision = 0.9551
- Recall = 0.9612
- Accuracy score = 0.9285

Random Forest

Figure 3.24 shows the confusion matrix when testing our data set with random forest model. From the confusion matrix, we calculate the following measures:

- Precision = 0.9550
- Recall = 0.9578
- Accuracy score = 0.9255

Support Vector Machines

Figure 3.25 shows the confusion matrix when testing our data set with support vector machines model. From the confusion matrix, we calculate the following measures:

- Precision = 0.8086
- Recall = 1
- Accuracy score = 0.8086

After training machine learning models, we feed testing data into those models and obtain key measures of performance from each model as shown in Table 3.3.

Table 3.3: Average Key Measures of Performance for each ML Model

ML Model	True Positive	False Positive	True Negative	False Negative	Precision	Recall	Accuracy Score
Logistic Regression	304.7	6.7	75.7	43.4	0.9811	0.8753	0.8837
Linear Discriminant Analysis	315.1	0.9	81.5	33	0.9975	0.9052	0.9213
K-Nearest Neighbors	348.1	82.4	0	0	0.8086	1	0.8086
Decision Tree	334.6	17.3	65.1	13.5	0.9551	0.9612	0.9285
Random Forest	333.4	17.4	65.0	14.7	0.9550	0.9578	0.9255
Support Vector Machines	348.1	82.4	0	0	0.8086	1	0.8086

3.4.5 Reason Behind the Observation

Table 3.3 shows a summary of the performance for each machine learning model. From the characteristics of the performance, we group the models into three groups.

Group 1 - Low accuracy for negative selection

Observing the logistic regression model, we notice that this model has a high precision value of 0.9811 and a high recall value of 0.8753. The accuracy score is also relatively good with a score of 0.8837. However, looking at negative prediction, the rate of accurately picking out a negative case is very low. With TN = 75.7 and FN = 43.4, the accuracy percentage of picking a negative case is TN/(TN + FN) = 63.56%, which is relatively low. We can see the same issue with linear discriminant analysis (LDA) model. The LDA model has high precision (0.9975) and high recall (0.9052). But its ability to pick out a correct negative case is low. With TN = 81.5 and FN = 33, the percentage to correctly pick out a negative case is TN/(TN + FN) = 71.18%. This group of machine learning models is not a suitable candidate for predicting offload recommendation for our scenario.

Group 2 - All predictions are positive

From our observation with the k-nearest neighbors classifier and support vector machines models, the prediction outcome is always positive. The size of the testing data set is 431 with 349 positive labels and 82 negative labels. The percentage of positive labels is 80.97%.

And because the models always predict positive outcomes, the accuracy score, precision, and recall are relatively high. However, these two models are not able to predict a negative outcome of our testing data set.

Group 3 - High accuracy

Our top two models are decision tree classifier and random forest. The decision tree classifier model has precision, recall, and accuracy score of 0.9551, 0.9612, and 0.9285 respectively. The random forest model has precision, recall, and accuracy score of 0.9550, 0.9578, and 0.9255 respectively. Most importantly, these two models can also predict negative outcome with higher accuracy than the other models. With TN = 65.1 and FN = 13.5, the percentage to correctly pick out a negative case of decision tree classifier model is TN/(TN + FN) = 82.82%. Similarly, the accuracy for negative prediction of random forest is 81.56%.

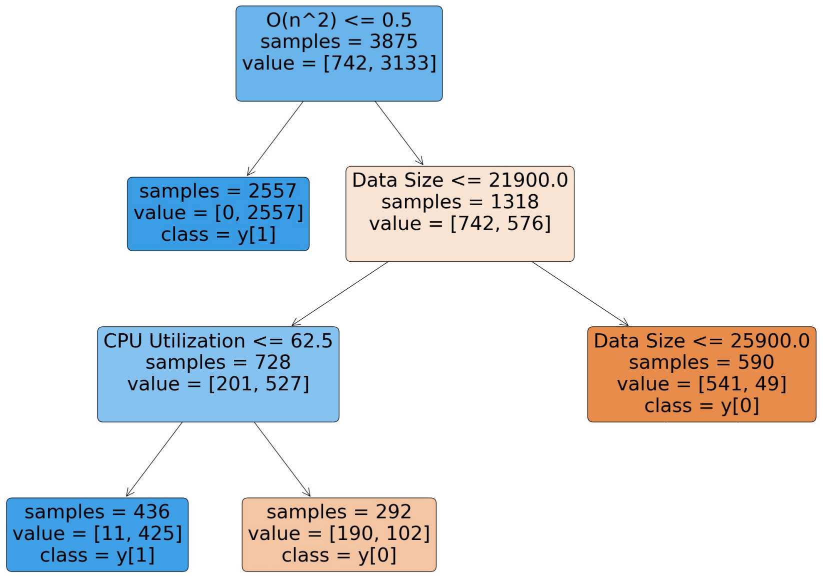


Figure 3.26: Decision Tree Model

3.4.6 Implication: Lessons Learned

From our experiments with different machine learning models, we choose decision tree classifier as the best model to predict our computation offload scenario. We use the whole data set we have as training data and feed this data set into the decision tree classifier algorithm to build a final machine learning model. To avoid overfitting, we set the maximum depth of the tree to 3. The result model is shown in Figure 3.26.

On comparing the decision tree classifier model with Algorithm 1, we find that they are closely aligned. As we see at the first node of decision tree classifier model, if the algorithmic complexity is not O(n^2), y is equal to 1, i.e. the model recommends to offload computation task from the edge layer to the equipment layer. We also see that the decision tree classifier only checks whether the algorithmic complexity is O(n^2) or not. If it is not O(n^2), it doesn't check for O(n) or O(nlogn), this is also aligned with Algorithm 1. Another similarity is that the decision tree classifier also checks for CPU Utilization of 62.5%, which is similar to the CPU Threshold in our algorithm, to determine the prediction outcome. In general, if we have an adequate amount of memory, then memory utilization factor doesn't affect the performance of computation task, therefore we did not have memory utilization as a feature in our data set. But our algorithm checks for RAM Utilization before determining offload recommendation. For simplicity, our Algorithm 1 doesn't check for data size, and that is a minor difference between the decision tree classifier model with our algorithm.

This comparison between the decision tree classifier model and Algorithm 1 shows that our recommendation for offloading computation task from the edge layer to the equipment layer is helpful and proven to be efficient in smart manufacturing for semiconductor.

3.5 Summary

Our experiments showed that computing tasks could be offloaded from the edge layer to the equipment layer in some instances resulting in an improvement in performance and a reduction in the data transfer requirement between layers in a manufacturing environment.

On the basis of these facts, we propose an algorithm to determine which layer should perform a computational task based on the factors of the existing conditions of the system. We also fit our experimental dataset into various machine learning models and determine the model that is suitable for offloading decision.

Chapter 4
Task Scheduling on Edge Layer with Option to Offload Computing Tasks from Edge to Equipment

This chapter presents our resource scheduling scheme for computation tasks, which provides three options. The first option is to offload tasks from the edge layer to the equipment layer. The second option takes into account priority when selecting a task for scheduling. The third option is to raise the priority of a task when it is waiting for a resource. The chapter is organized as follows. Section 4.1 presents the design of the system architecture of our resource scheduling scheme. Section 4.2 explains the experimental setups, followed by the discussion of our datasets shown in section 4.3. Next, we walk through our scheduling algorithms and the performance metrics in section 4.4 and in section 4.5, respectively. Section 4.6 shows our experimental results. Finally, we summarize the chapter in section 4.7.

4.1 System Architecture

In traditional systems, the layers that perform computation tasks are the edge layer and the cloud layer. Some systems have fog/edge layers instead of only the edge layer. Our system architecture consists of both cloud, an edge layer, and an equipment layer. Specifically, the cloud is used for data storage and heavy computing tasks; the edge layer is employed to support smart manufacturing, and the equipment layer services as an interface to hardware. Though our system architecture is similar to other system designs, our system differentiate from other systems in that the equipment layer can also perform computing tasks. Our scheduler determines if tasks can be performed at the equipment level based on multiple factors: algorithm complexity, CPU utilization, RAM utilization, and availability.

Our system architecture contains the following four parts.

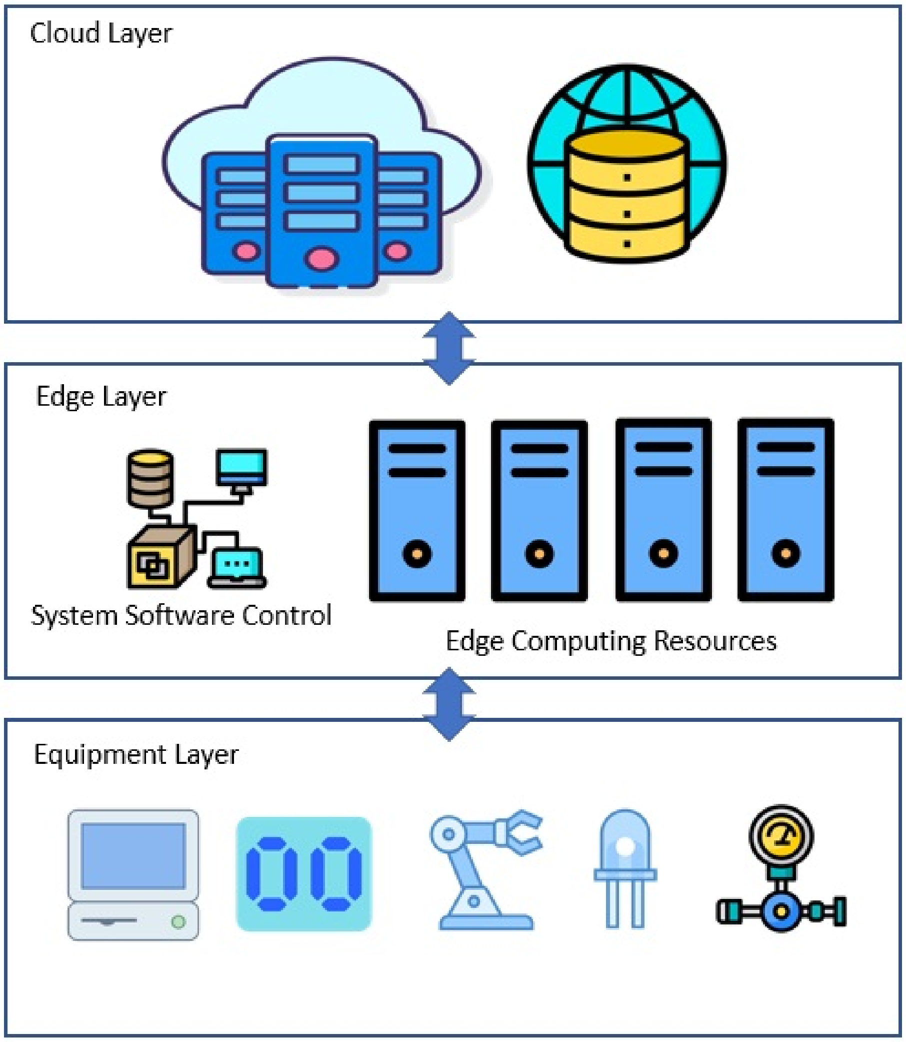


Figure 4.1: System Architecture with Scheduler

- Cloud Layer
- Edge Layer
- Equipment Layer
- System Software Control
  - Scheduler
  - Communication Module

In our system, the System Software Control contains two modules - *Scheduler* and *Communication Module*. Communication Module is responsible for communicating tasks that resources need to complete to resources. Communication Module is also in charge of collecting statuses and results from the tasks.

4.2 Experimental Setup

We create a simulation system to carry out scheduling experiments. This simulation system contains a pool of edge servers at the edge layer for performing computation tasks, and a pool of equipment computers at the equipment layer for offloading tasks from the edge.

The detailed simulation system is outlined as follows:

- Edge Layer
  - 6 Edge Servers
    - Default 5% CPU Utilization
    - Default 20GB Memory Utilization, Max 512GB
- Equipment Layer
  - 7 Equipment Computers
    - Default 5% CPU Utilization

    * Default 2GB Memory Utilization, Max 32GB

- Resources Scheduler

4.3 Datasets

We generate ten datasets, on which we perform experiments to make sure there is no dataset-dependency in our results. Each dataset consists of 1000 randomly generated computation tasks. And each computation task exhibits the following properties:

- T_{Comp}: The computing time for task t, which is in the range of 1 - 30 seconds.
- $T_{Total\_transfer}$: The total time to read, write, and transfer needed data to complete task t, which is in the range of 1 - 10 seconds.
- TA_t: The time the task arrives at the system, ranging between 0 - 99 seconds.
- ϵ_t: CPU Utilization for task t, which is randomly generated between 1 - 30 percent.
- ω_t: Memory Utilization for task t, which is randomly generated between 1 - 16 GB.
- δ_t: algorithm complexity of task t, which can be O(n), O(nlogn), or O(n^2).
- Priority level (e.g., 1, 2, 3, 4, and 5, and 1 denotes the highest priority)

Data transfer time is only applicable if the task is executed on the edge layer. The total time to perform task t on the edge layer can be calculated as

$$T_{Eq\_total} = T_{Total\_transfer} + T_{Comp}, \tag{4.1}$$

If the task is executed on the equipment layer, data transfer time will not be counted in the total execution time. From our experiments (see Chapter 3), the computing time on the equipment layer is longer than the computing time of the same task on the edge layer because the edge layer contains more powerful servers. The computing time also depends on the current CPU utilization on the equipment computer. Therefore, in our simulation scheduling, if the task is assigned to the equipment layer, the computing time T_{Comp} will be modified as follows:

- If $CPU\_UTILIZATION$ on Equipment PC ≤ 50
  $T_{Comp} = T_{Comp} * 1.1$
- If $CPU\_UTILIZATION$ on Equipment PC > 50 and ≤ 75
  $T_{Comp} = T_{Comp} * 1.2$
- If $CPU\_UTILIZATION$ on Equipment PC > 75 and ≤ 90
  $T_{Comp} = T_{Comp} * 1.4$
- If $CPU\_UTILIZATION$ on Equipment PC > 90
  $T_{Comp} = T_{Comp} * 1.6$

4.4 Algorithm

The purpose of our scheduling algorithm is two-fold. First, resources at the edge layer and the equipment layer are seamlessly integrated. Second, appropriate resources are allocated to computation tasks. The scheduling algorithm is outlined in Algorithm 2, 3, and 4. Table 4.1 shows the notations and symbols used in the algorithms throughout this chapter. In the process of making scheduling decisions, the scheduler takes into account the time complexity of computation tasks as well as the availability of resources to determine whether it is feasible to offload computation task to the equipment layer.

In detail, the scheduling algorithm goes through the following four primary stages:

- Checking for completed tasks and removing these completed tasks from the tasks pool.
- Analyzing cancelled tasks and removing these cancelled tasks from the tasks pool.
- Checking if there are any waiting tasks.

Algorithm 2 Scheduling with Offloading Tasks Enabled

```
Input: $PW$, $PE$, $LEN(PW)$, $LEN(PE)$, $WTL$
 1: Start the scheduler
 2: for $i \leftarrow 0$ to $(LEN(PE) - 1)$ do
 3:     $task \leftarrow PE[i]$ // Get the $i^{th}$ task in executing list
 4:     if $\alpha_{task} = COMPLETED$ then
 5:         Remove $task$ from $PE$
 6:     end if
 7: end for
 8:
 9: if $LEN(PW) = 0$ then
10:     No action, wait 1 second, then go back to step 1
11: end if
12:
13: for $i \leftarrow 0$ to $(LEN(PW) - 1)$ do
14:     $task \leftarrow PW[i]$ // Get the $i^{th}$ task in wait list
15:     if $\alpha_{task} = CANCELLED$ then
16:         Remove $task$ from $PW$
17:     end if
18: end for
19:
20: // In case there was some cancelled tasks and PW is empty
21: if $LEN(PW) = 0$ then
22:     No action, wait 1 second, then go back to step 1
23: end if
24:
25: while (Resource is available AND $LEN(PW) > 0$) do
26:     // Pick the next task to assign resource
27:     if (EP is true) then
28:         $selected\_task \leftarrow get\_highest\_priority\_task()$
29:     else
30:         $selected\_task \leftarrow get\_longest\_wait\_task()$
31:     end if
32:
33:     // Start allocating resource for task
34:     $EC \leftarrow null$
```

```
35:     // If the task can run on equipment, try to allocate an equipment for the task
36:     if ($\delta_{selected\_task} < O(n^2)$) then
37:         $EC \leftarrow allocate\_equipment\_pc()$
38:     end if
39:
40:     $\beta_{selected\_task} \leftarrow -1$
41:     // If equipment resource is allocated, executing the task
42:     $task\_has\_resource \leftarrow false$
43:     if $EC \neq null$ then
44:         $execute\_task(selected\_task, EC)$
45:         $\beta_{selected\_task} \leftarrow 0$
46:         Move selected_task from PW to PE
47:         $task\_has\_resource \leftarrow true$
48:     else
49:         // If equipment resource is not available, try to allocate an edge server
50:         $ES \leftarrow allocate\_edge\_server()$
51:         // If the edge server is allocated, execute the task
52:         if $ES \neq null$ then
53:             $execute\_task(selected\_task, ES)$
54:             $\beta_{selected\_task} \leftarrow 1$
55:             Move selected_task from PW to PE
56:             $task\_has\_resource \leftarrow true$
57:         end if
58:     end if
59:     if $task\_has\_resource \neq true$ then
60:         $\gamma_{selected\_task} \leftarrow \gamma_{selected\_task} + 1$
61:         if $\gamma_{selected\_task} = WTL$ then
62:             if $\theta_{selected\_task} > 1$ then
63:                 $\theta_{selected\_task} = \theta_{selected\_task} - 1$
64:                 $\gamma_{selected\_task} \leftarrow 0$
65:             end if
66:         end if
67:     end if
68: end while
69: Wait 1 second, then go back to step 1
```

Algorithm 3 get_highest_priority_task()

```
Input: $PW$, $LEN(PW)$
1: if $LEN(PW) = 0$ then
2:     Return NULL
3: end if
4: for $i \leftarrow 1$ to 5 do
5:     for $j \leftarrow 0$ to $LEN(PW) - 1$ do
6:         $task \leftarrow PW[j]$
7:         if $\theta_{task} = i$ then
8:             return task
9:         end if
10:    end for
11: end for
```

Algorithm 4 get_longest_wait_task()

```
Input: $PW$, $LEN(PW)$
1: if $LEN(PW) = 0$ then
2:     Return NULL
3: else
4:     Return PW[0]
5: end if
```

- Continuously performing resource allocation until there is no waiting task or no available resources.

The scheduling algorithm first accesses the pool of executing tasks (PE) for any completed tasks, followed by deleting those completed tasks from the pool PE (see lines 2 - 7 in Algorithm 2). Next, the scheduling algorithm checks the pool of waiting tasks (PW). If the pool is empty, the scheduling algorithm will await one second and restart from the beginning (see lines 9 - 11 in Algorithm 2). Then, the scheduling algorithm examines the pool of waiting tasks (PW) and remove cancelled tasks from the pool if any (see line 13 - 18 in Algorithm 2). After that, the scheduling algorithm checks the pool of waiting tasks one

Algorithm 5 allocate_equipment_pc()

```
Input: $THLD\_CPU\_EQ$, $THLD\_RAM\_EQ$, task $t$
1: Get the list $LIST\_EC$ of PCs from equipment layer, sorted in ascending order by CPU utilization.
2: $count \leftarrow LEN(LIST\_EC)$
3:
4: $EC \leftarrow NULL$
5: $found = false$
6: for $i \leftarrow 0$ to $count - 1$ do
7:     $EC \leftarrow LIST\_EC[i]$
8:     $found = false$
9:     if $(CPU_{EC} + \lambda_t > THLD\_CPU\_EQ)$ then
10:        break
11:    end if
12:    if $RAM_{EC} + \omega_t \leq THLD\_RAM\_EQ$ then
13:        $found = true$
14:        break
15:    end if
16: end for
17:
18: if $(found)$ then
19:     Return $EC$
20: else
21:     Return $NULL$
22: end if
```

more time and if the pool is empty, it waits for one second and restarts from the beginning (see line 21 - 23 in Algorithm 2).

The scheduling algorithm now starts allocation resource for computation tasks. It always monitors and ensure that there is at least one waiting task and resources are available (see line 24 in Algorithm 2). Next, the scheduling algorithm picks a task to allocate resource. If task priority is considered (EP is true), the scheduling algorithm picks the highest priority task in waiting pool, otherwise it picks the task with the longest wait time (see line 25 - 30

Algorithm 6 allocate_edge_server()

```
Input: $THLD\_CPU\_ED$, $THLD\_RAM\_ED$, task $t$
 1: Get the list $LIST\_ES$ of edge servers from the edge layer, sorted in ascending order by
    CPU utilization.
 2: $count \leftarrow LEN(LIST\_ES)$
 3:
 4: $ES \leftarrow NULL$
 5: $found = false$
 6: for $i \leftarrow 0$ to $count - 1$ do
 7:     $ES \leftarrow LIST\_ES[i]$
 8:     $found = false$
 9:     if $(CPU_{ES} + \lambda_t > THLD\_CPU\_ED)$ then
10:         break
11:     end if
12:     if $RAM_{ES} + \omega_t \leq THLD\_RAM\_ED$ then
13:         $found = true$
14:         break
15:     end if
16: end for
17:
18: if $(found)$ then
19:     Return $ES$
20: else
21:     Return $NULL$
22: end if
```

in Algorithm 2). Algorithm 3 explains how to elect the highest priority task, and Algorithm 4 is the process of picking the longest waited task.

After selecting a task, the next step of the algorithm is to assign resources. If the time complexity of the task is less than $O(n^2)$, the scheduling algorithm tries to pinpoint a resource from the equipment layer (see line 34 - 37 in Algorithm 2). The Algorithm 5 shows the process of allocating a resource from the equipment layer. If a resource is found in the equipment layer, the task will be executed at the equipment layer: the task will be

Table 4.1: Notations in Scheduling Algorithms

Notations	**Description**
PW	The pool of tasks waiting for execution
PE	The pool of tasks that are executing
$LEN(PW)$	The number of tasks waiting for execution
$LEN(PE)$	The number of tasks in executing state
EP	Enabled Task Priority: if the value is true, priority of the task is considered when selecting the next task to allocate resource, if the value is false, ignore task priority
WTL	Wait time limit for upgrading task's priority
ES	Edge server
EC	Equipment computer
CPU_{ES}	Current CPU Utilization of the Edge server ES
RAM_{ES}	Current Memory Utilization of the Edge server ES
CPU_{EC}	Current CPU Utilization of the Equipment computer EC
RAM_{EC}	Current Memory Utilization of the Equipment computer EC
$THLD\_CPU\_ED$	CPU Threshold at the Edge layer
$THLD\_RAM\_ED$	Memory threshold at the Edge layer
$THLD\_CPU\_EQ$	CPU Threshold at the Equipment layer
$THLD\_RAM\_EQ$	Memory threshold at the Equipment layer
$LIST\_ES$	List of edge servers at the Edge layer
$LIST\_EC$	List of equipment PCs at the Equipment layer
$LEN(LIST\_ES)$	The number of edge servers at the Edge layer
$LEN(LIST\_EC)$	The number of equipment PCs at the Equipment layer
α_t	Execution status of task t. Valid values are WAITING, EXECUTING, COMPLETED, CANCELLED
β_t	1: task t runs on edge server, 0: task t runs on equipment, -1: no resource has been assigned
δ_t	Algorithm complexity of task t. Values: O(n), O($nlogn$), O(n^2)
γ_t	Wait time of task t from last upgrade.
θ_t	Current priority of task t
λ_t	CPU requirement for task t
ω_t	Memory requirement for task t

moved from the pool of waiting tasks (PW) to the pool of executing tasks (PE) (see line 39 - 46 in Algorithm 2). If the task is not executed at the equipment layer, the scheduling algorithm attempts to assign resources from the edge layer. The Algorithm 6 sketches the process of allocating resource at the edge layer. If an edge resource is available, the task will be performed at the edge layer (see line 48 - 55 in Algorithm 2).

If there is no available resource for the scheduled task, the scheduling algorithm will expand the wait time of the task. If the wait time of the task reaches a stipulated wait limit for upgrading priority, the scheduling algorithm promotes the priority of the task and resets the wait time to zero (see line 57 - 65 in Algorithm 2). Then, the scheduling algorithm awaits one second and restart the process. The scheduling period of one second, of course, can be configured according to the dynamic workload conditions.

4.5 Performance Metrics

To present the experimental results gleaned from extensive experiments in the next section, we outline the performance metrics below. The total running time of all the submitted tasks resemble the workload to be handled by the smart manufacturing system. We pay particular attention to the total and average response time of tasks at various priority levels ranging from 1 to 5.

All the measured metrics are tabulated in the list below.

- Total running time for 1000 tasks in dataset
- Number of offloaded tasks
- Total response time and Average response time for all tasks
- Total response time and Average response time for tasks with priority 1
- Total response time and Average response time for tasks with priority 2
- Total response time and Average response time for tasks with priority 3
- Total response time and Average response time for tasks with priority 4
- Total response time and Average response time for tasks with priority 5

4.6 Experimental Results

4.6.1 A State-of-the-Art Scheduling Technique: Hybrid Computing Solution (HCS)

To evaluate our scheduling performance, we compare our scheduling results with a State-of-the-Art scheduling technique: the Hybrid Computing Solution or *HCS* as described in the literature [33]. In the Hybrid Computing Solution - HCS, the algorithm selects an edge computing server if the executing time of tasks running on the server is less than the requirement to complete the task.

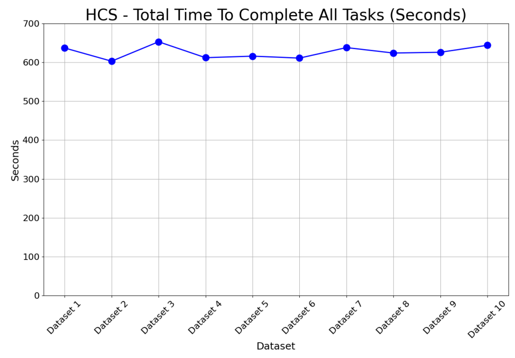
We collect the following data as performance metrics for comparison:

- Total time to complete all tasks (Unit: seconds).
- Throughput (Unit: number of tasks per second).
- Total wait time (Unit: seconds).
- Average wait time per task (Unit: seconds).

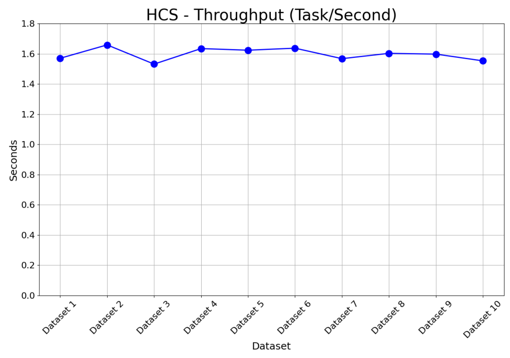
Figure 4.2 sketches the results obtained by running the leading-edge HCS algorithm described in the literature [33].

4.6.2 Comparing Experimental Results with HCS

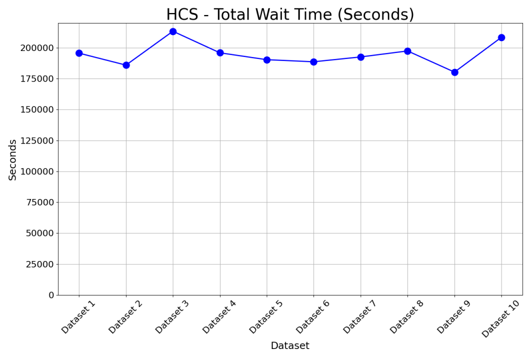
We perform our scheduler as described in Algorithm 2 on the same 10 datasets that are published available 4.6.1. When offloading computation tasks to the equipment layer, our scheduling algorithm selects a resource from the equipment layer with the constraint of the CPU threshold as well as the memory threshold on equipment computers. We repeatedly run our algorithm on all the 10 datasets with different combinations of CPU utilization and memory utilization to cover the wide workload spectrum. For the purpose of comparing our scheduling algorithm with the State-of-the-Art HCS scheduling algorithm, we pick the result of the settings in the middle of the range, which is comprised of CPU threshold of 50% and



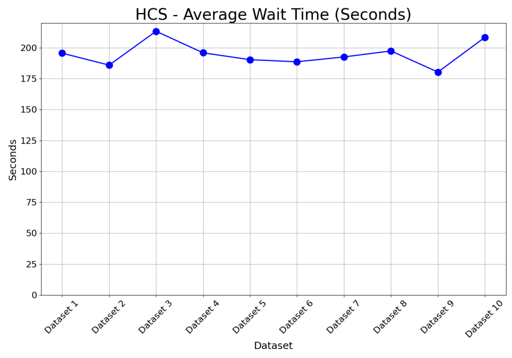
(a) Total Time to Finish All Tasks



(b) Throughput



(c) Total Wait Time



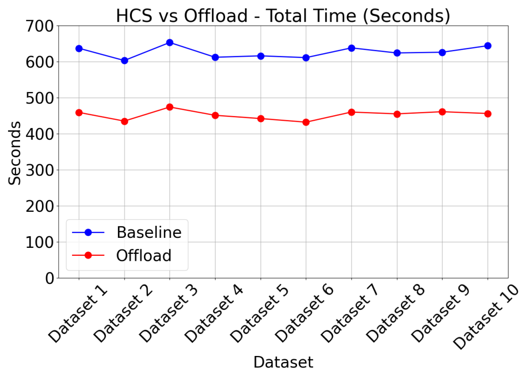
(d) Average Wait Time

Figure 4.2: Results of the Hybrid Computing Solution or HCS.

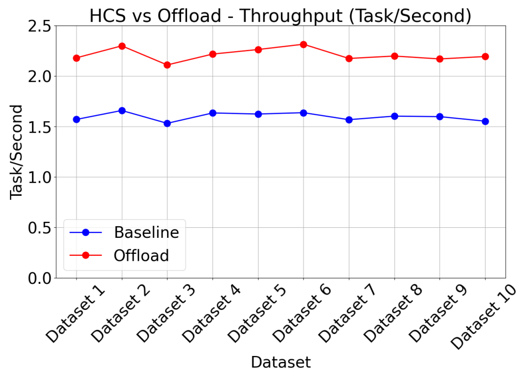
memory threshold of 8GB. In this scenario, the comparison result unveils that the average total time to complete 1000 tasks is drastically shortened by 27.75% when enable offloading tasks to the equipment layer. The comparison result also shows that the throughput is immensely bolstered by 38.45%, the total wait time and average wait time per task are cut back by 38.8%. Figure 4.3 shows the comparison in total running time, throughput, total wait time, and average wait time. The data used for the comprehensive comparison can be found in Appendix B.

4.6.3 Scheduling Experimental Results with Tasks Priority

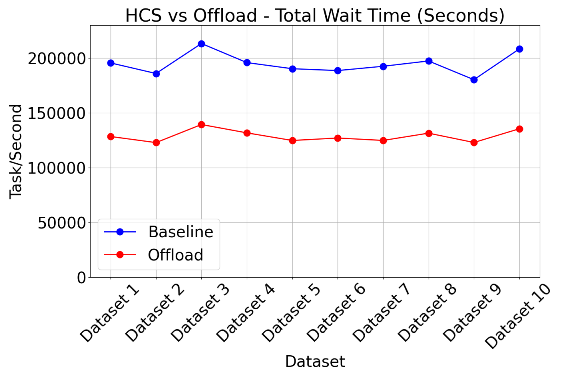
Another factor we consider in resource scheduling is task priority. As discussed in Section 4.3, tasks are assigned priorities - a value ranging between 1 to 5. The five priority



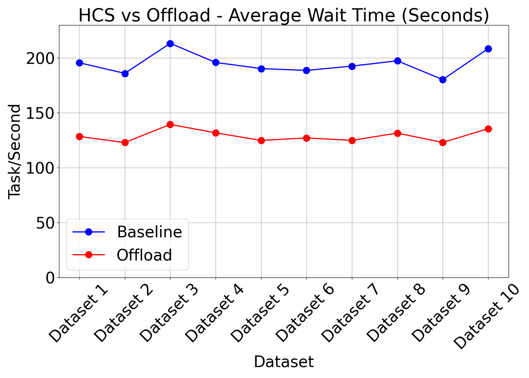
(a) HCS vs Offload - Total Time



(b) HCS vs Offload - Throughput



(c) HCS vs Offload - Total Wait Time



(d) HCS vs Offload - Average Wait Time

Figure 4.3: HCS vs Offload with 50% CPU Threshold and 8GB Memory Threshold

levels only serve as a demonstration. In the real-world setting, of course, the total number of priority levels can be configured. In our experiments, Priority 1 is the highest priority, and priority 5 is the lowest priority. When task priority option is enabled in the resources scheduler and there is the lack of adequate resources, the lower priority tasks would have to wait indefinitely - a serious starvation problem to be addressed. To resolve this issue, we implement the wait-time limit or *WTL* in Table 4.1. WTL is the amount of time a task should wait before its priority is promoted. To compare the effect of different WTL values, we fix, in the equipment level, the CPU threshold at 50% and the memory threshold at 8GB.

We undertake the experiments with the following value of the wait-time limit (WTL):

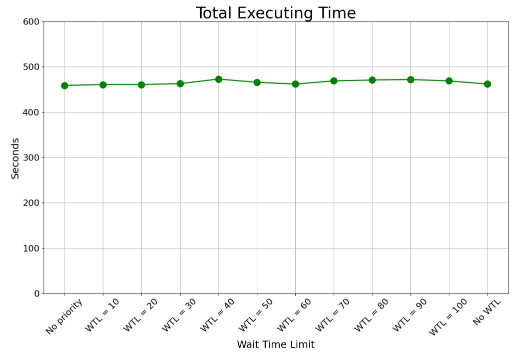
- No priority
- Apply priority with WTL = 10 seconds

- Apply priority with WTL = 20 seconds
- Apply priority with WTL = 30 seconds
- Apply priority with WTL = 40 seconds
- Apply priority with WTL = 50 seconds
- Apply priority with WTL = 60 seconds
- Apply priority with WTL = 70 seconds
- Apply priority with WTL = 80 seconds
- Apply priority with WTL = 90 seconds
- Apply priority with WTL = 100 seconds
- Apply priority and no priority upgrade (No WTL)

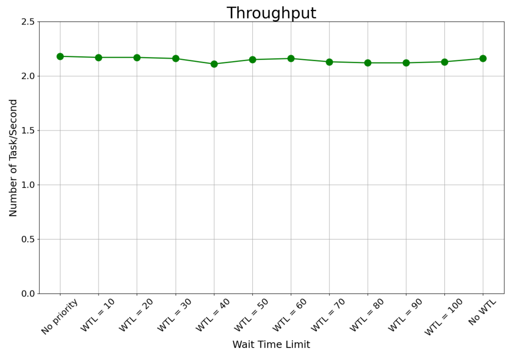
From our experimental results, as shown in Figure 4.4, there is not any significant differences in total executing time, throughput, total wait time, and average wait time of all tasks when running resource scheduling with different values of wait-time Limits. On the contrary, looking at average wait time for each priority group, we see a significant difference in performance of each group when changing wait-time limit value. If wait-time limit is set to a small value – low priority groups can be upgraded in a faster pace – the tasks with priority 1 have no performance edge over those tasks with lower priorities. When wait-time limit increases so that it takes longer time for the low priority groups to be promoted, we observe from 4.5 that the tasks with the highest priority outperform the low-priority tasks in term of average wait time. The data shown in this section can be found in Appendix C.

4.7 Scheduling Summary

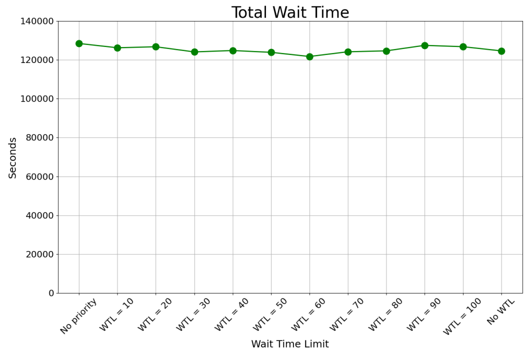
In this section, we proposed a system for smart manufacturing architecture, which consists of a cloud layer, an edge layer, an equipment layer, and a system software control



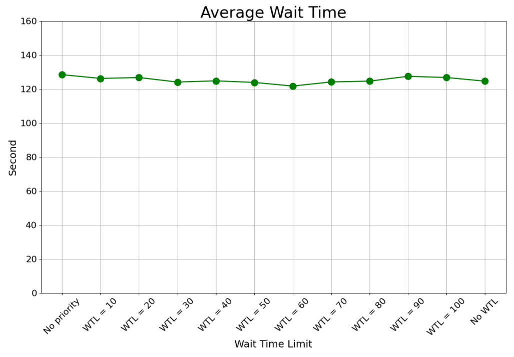
(a) Total Executing Time with Different WTL



(b) Throughput with Different WTL



(c) Total Wait Time with Different WTL



(d) Average Wait Time with Different WTL

Figure 4.4: Offload Scheduling Data with Different Wait Time Limits

module. The system software control module is the vital part of our design. At the heart of the system software control module is a scheduling algorithm that supports resource scheduling with ability of offloading computation tasks from the edge layer to the equipment layer. Besides, the scheduling algorithm can take priorities into consideration when selecting computation tasks to schedule resources. Moreover, the scheduling algorithm can also promote a task to a higher priority if the task wait too long to execute, thus preventing low-priority tasks from sitting at the bottom of the queue without resources to execute.

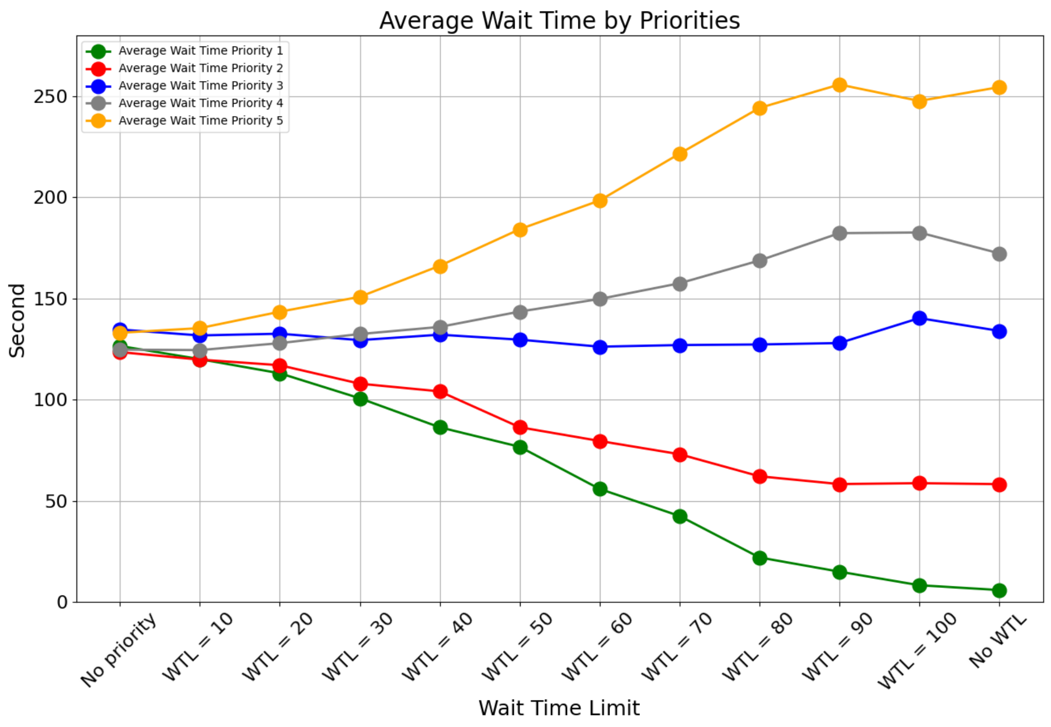


Figure 4.5: Average Wait Time by priorities with different wait-time limit or WTL.

Chapter 5

Dynamic AI Computation Tasks with SECS/GEM in Semiconductor Smart Manufacturing

In this chapter, we present a mechanism for offloading computation tasks from the edge layer to the equipment layer. This chapter is organized as follows. Section 5.1 and section 5.2 respectively discuss the background and motivation of the communication protocol between the host computer on the edge layer and the equipment. Section 5.3 presents the latency when transferring data variables through SECS/GEM interface. Next, section 5.4 discusses the reasons we believe a new protocol is needed for communication between the edge layer and the equipment layer, and section 5.5 presents the system architecture to support the new protocol. Section 5.6 presents how the new protocol works, followed by the scenarios of usages in section 5.7. Finally, the summary of this chapter is given in section 5.8.

5.1 Background

With the rapid development of machine learning and artificial intelligence, advanced data analysis has become essential in smart manufacturing, especially in the semiconductor industry since it helps in early detection of failures in the process [1]. A massive amount of data is collected during manufacturing processes, and analyzing the data in real-time is critical for improving quality and productivity.

The use of multiple layers systems is the most common way of managing smart manufacturing: the equipment layer, the edge layer, and the cloud layer [2]. In semiconductor manufacturing, the equipment layer includes many types of equipment such as process equipment, metrology equipment, and test equipment. The equipment layer is responsible for collecting data during processing and testing. The data will be transferred to the edge layer for analysis and production control. Some data will be transferred to the cloud layer for long term storage, machine learning and artificial intelligence computing tasks. This conventional method of organizing data and analysis creates latency due to the delay in data transfer between layers of the smart manufacturing system. Another problem with this method is that the network bandwidth gets overloaded with vast amounts of data transferring from equipment in the factory to the edge servers.

Some computing tasks can be performed at the equipment layer to reduce the latency of transferring data from the equipment to the edge. The issue is that computing tasks need to be dynamically decided by the edge layer during material processing. The equipment layer is not able to identify which computing tasks need to be performed at a given time during processing.

5.2 Motivation

The issues motivate us to propose a communication protocol in SECS/GEM messages for the host to assign data analysis tasks to the equipment and collect the results for making decisions for the next steps in the process. In semiconductor manufacturing, different types of data analytics need to be performed on process data, including process data monitoring, statistical process control, resources scheduling, and failure prediction [3]. Some of those computing tasks can be performed at the equipment level to achieve real-time response. Without instruction from the host, the equipment always performs certain pre-defined analyses at some pre-defined stages during material processing. It is hence critical for the host to be able to send requests to the equipment to perform data analyses and report the results. The proposed system will enable dynamic analysis computation tasks in semiconductor manufacturing.

Table 5.1: Latency of Transferring Data Variables with SECS/GEM

Number of Data Variables	**Assigning Values to Data Variables Time (seconds)**	**Transferring Data Variables to Host Time (seconds)**
1000	0.33	1.48
2000	0.83	2.15
3000	1.17	3.65
4000	1.84	4.92
5000	2.02	5.84
6000	2.31	6.91
7000	2.87	7.38
8000	3.19	8.53
9000	3.45	9.28
10000	3.66	10.26

5.3 Latency With Data Transfer Using SECS/GEM Messages

To carry out experiments, we set up a small network that includes two computers with CPU Intel i7 Processor, 16G RAM, 1TB HD, and Windows 10 Professional. The computers were connected through a gigabit Ethernet switch. We developed a basic SECS/GEM message transfer program for testing purposes. The host computer executed SECS/GEM in active mode and the equipment computer executed SECS/GEM in passive mode. We collected data variables by using data collection method defined in [5, 6]. The experiments were conducted with different numbers of data variables.

Table 5.1 shows the results we collected in the experiments. It is worth noting that the latency tabulated here is higher than the actual data transfer time over the network. The reason behind this measurement is that the latency of data transferring over SECS/GEM interface includes data transfer time as well as all overheads incurred by the SECS/GEM interface, which is responsible for formatting SECS/GEM messages on the equipment side and decrypting the messages on the host side to extract the values of data variables.

5.4 Adding New Set of Messages for Communication Between the Edge and the Equipment

In semiconductor industry, factory automation is usually implemented through SECS/GEM interface. The SECS/GEM interface supports a set of standards which were defined by the Semiconductor Equipment and Materials International organization (SEMI). In smart manufacturing system, the equipment belongs to the equipment layer, and the host, which controls the equipment and collects data though SECS/GEM interface, belongs to the edge layer.

The current SECS/GEM interface does not include a protocol for implementing dynamic data analysis to support smart manufacturing in the semiconductor industry. This paper proposes a protocol for the host in the edge layer to assign data analysis tasks to the equipment and collect the results through SECS/GEM interface.

We propose to add a new stream, which includes a collection of SECS messages, to support dynamic data analysis during processing. The new stream provides an interface for the host to define computing tasks and send the tasks to the equipment. The new stream also defines an interface for the equipment to acknowledge the tasks and report the results to the host. A computing task can be a simple calculation, a predefined statistical process control rule, or a python script with input parameters.

5.5 System Design

To reduce the latency of data analysis tasks, we design a system that can minimize data transfer from the equipment to the host. Fig. 5.1 shows the system design that supports dynamic analysis computations using SECS/GEM. In this design, the host can send SECS messages to the equipment to define analysis computing tasks and link the tasks to some specific events. On the equipment, we add a new software module, Data Analysis Engine, that executes the analysis tasks and reports the results to the host. With this design, it is not necessary to send data variables to the host for analysis.

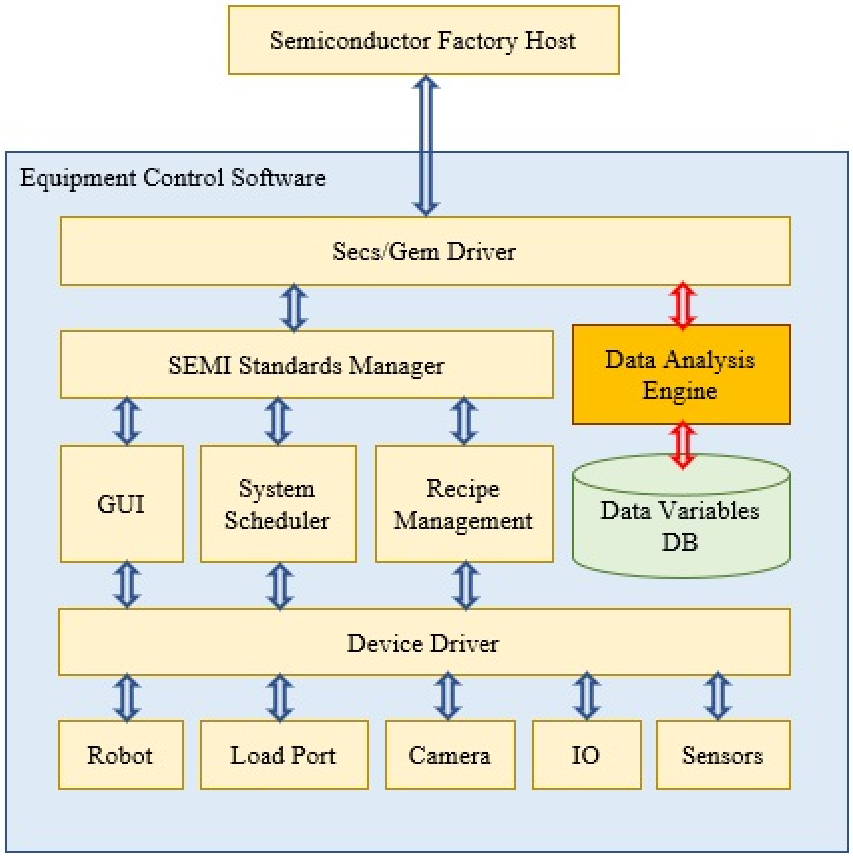


Figure 5.1: System Design

5.6 New Messages to Support Offloading Computing Tasks from Edge to Equipment

We propose a new collection of messages, Stream 22, in SECS/GEM interface to handle all dynamic data analyses communication between the host and the equipment. The reason we use Stream 22 because the latest addition to SECS/GEM interface was Stream 21. Data analysis tasks include a method to perform data analysis, a data variable list used for the task, and a data variable list for storing the results. Then the defined tasks will be linked to some particular events of the semiconductor equipment. Events are anything that happens on the systems. For example, material arrives at the system, processing starts, processing ends, etc. When the event is triggered, the Data Analysis Engine performs the task and sends the results to the host. Data analysis tasks can be classified as predefined analyses or custom analyses.

5.6.1 SECS/GEM Protocol for Predefined Analyses

The host can send a request for predefined analysis task to the equipment using keywords. The initial keywords are “min”, “max”, “average”, “stdev”, and “sum”. More computing keywords can be added into the protocol based on agreements between the chip manufacturers and the equipment providers to support certain analysis tasks. For predefined analyses, we define the following SECS messages to setup analysis tasks on the equipment: S22F1, S22F2, S22F3, S22F4, S22F5, and S22F6.

S22F1, Define analysis task (Host to Equipment)

The purpose of this message is for the host to define analysis tasks using keyword for the equipment to perform. A list of zero-length following <DATAID> deletes all predefined analysis task definitions and associated links. A list of zero-length following <DATAID> deletes the analysis task TASKID. All CEID links to this TASKID are also deleted.

```
Structure:
L, 2
    1. <DATAID>
    2. L, m
        1. L, 2
            1. <TASKID>
            2. L, 3
                1. <KEYWORD>
                2. L, n #VID list for results
                    1. <VID>
                    .
                    .
                    n.  <VID>
                3. L, p #VID list for analysis task
                    1. <VID>
                    .
                    .
                    p.  <VID>
        .
        .
        m.  L, 2
            1. <TASKID>
            2. L, 3
                1. <KEYWORD>
                2. L, n #VID list for results
                    1. <VID>
                    .
                    .
                    n.  <VID>
                3. L, p #VID list for analysis task
                    1. <VID>
                    .
                    .
                    p.  <VID>
```

S22F2, Define analysis task acknowledge (Equipment to Host)

Acknowledge or return error from S22F1.

`Structure:`

`<DATACK>`

DATACK is Define Analysis Task Acknowledge Code. Valid values of DATACK:

- 0: Accept
- 1: Denied. Invalid format.
- 2: Denied. At least one TASKID was already defined.
- 3: Denied. At least one VID does not exist.
- 4: Denied. At least one KEYWORD is unknown.
- 5: Denied. Other error.

S22F3, Link event to analysis task (Host to Equipment))

The purpose of this message is for the host to link analysis tasks to a collection event (CEID). A list of zero-length following <CEID> deletes all links to that CEID.

```
Structure:
L, 2
    1. <DATAID>
    2. L, m
        1. L, 2
            1. <CEID>
            2. L, n
                1. <TASKID>
                .
                .
                n.  <TASKID>
        .
```

```
.
m.  L, 2
    1. <CEID>
    2. L, n
        1. <TASKID>
        .
        .
        n.  <TASKID>
.
.
```

S22F4, Link event to analysis task acknowledge (Equipment to Host)

Acknowledge or return error from S22F3.

```
Structure:
<LATACK>
```

LATACK is Link Analysis Task Acknowledge Code. Valid values of LATACK:

- 0: Accept
- 1: Denied. Invalid format.
- 2: Denied. At least one CEID link was already defined.
- 3: Denied. At least one CEID does not exist.
- 4: Denied. At least one TASKID does not exist.
- 5: Denied. Other error.

S22F5, Task Result Notify (Equipment to Host)

Equipment sends task result to the host.

```
Structure:

L, 2

    1. <TIMESTAMP>

    2. L, m

        1. L, 3

            1. <TASKID>

            2. L, n

                1. <RESULT>

                .

                .

                n.  <RESULT>

            3. <ERRORTEXT>

        .

        .

        m.  L, 3

            1. <TASKID>

            2. L, n

                1. <RESULT>

                .

                .

                n.  <RESULT>

            3. <ERRORTEXT>

        .

        .
```

S22F6, Task Result Notify Acknowledge (Host to Equipment)

The host acknowledges the task result from S22F5.

```
Structure:
<TRACK>
```

TRACK is Task Result Acknowledge Code. Valid values of TRACK:

- 0: Accept
- 1: Denied. Invalid format.
- 2: Denied. At least one TASKID does not exist.
- 3: Denied. The number of results does not match the definition of the task TASKID.
- 4: Denied. Other error.

After defining an analysis task and linking the task to a specific event, the host needs to enable the event with S2F37.

5.6.2 SECS/GEM Protocol for Custom Analyses

For complex analysis algorithms, we design a protocol for the host to send the algorithms to the equipment in Python language format and collect the results. In the new stream 22, we define the following SECS messages to setup custom analysis tasks on the equipment: S22F7 and S22F8.

S22F7, Define analysis script task (Host to Equipment)

The purpose of this message is for the host to define analysis script tasks that use python script. A list of zero-length following <DATAID> deletes all analysis script task definitions and associated links. A list of zero-length following <TASKID> deletes the analysis task TASKID. All CEID links to this TASKID are also deleted. SCRIPTTYPE indicates the type of the script that is stored in the variable SCRIPTTEXT. Example of valid values of SCRIPTTYPE are "python" or "perl". It is determined by the agreement between the equipment supplier and the semiconductor factory to use a different type of script.

```
Structure:
L, 2
    1. <DATAID>
    2. L, m
        1. L, 2
            1. <TASKID>
            2. L, 4
                1. <SCRIPTTYPE>
                2.  <SCRIPTTEXT>
                3. L, n #VID list for results
                    1. <VID>
                    .
                    .
                    n.  <VID>
                4. L, p #VID list for script parameters
                    1. <VID>
                    .
                    .
                    p.  <VID>
        .
        .
        m.  L, 2
            1. <TASKID>
            2. L, 3
```

```
                  1. <SCRIPTTYPE>
                  2.  <SCRIPTTEXT>
                  3. L, n #VID list for results
                      1. <VID>
                      .
                      .
                      n.  <VID>
                  4. L, p #VID list for script parameters
                      1. <VID>
                      .
                      .
                      p.  <VID>
```

S22F8, Define analysis script task acknowledge (Equipment to Host)

Acknowledge or return error from S22F7.

`Structure:`

`<DASTACK>`

DASTACK is Define Analysis Script Task Acknowledge Code. Valid values of DASTACK:

- 0: Accept
- 1: Denied. Invalid format.
- 2: Denied. At least one TASKID was already defined.
- 3: Denied. At least one VID does not exist.
- 4: Denied. Invalid Python script.
- 5: Denied. Other error.

5.6.3 Ad-hoc Analysis

S22F9, Request to perform analysis task (Host to Equipment)

The purpose of this message is to allow the host to trigger the equipment to perform analysis tasks without waiting for the corresponding events. When a corresponding event is triggered, the tasks will be performed again.

```
Structure
L, 2
    1. <DATAID>
    2. <L, m
        1. <TASKID>
        .
        .
        m.  <TASKID>
```

S22F10, Request to perform analysis task acknowledge (Equipment to Host)

Acknowledge or return error from S22F9.

```
Structure:
<REQTASKACK>
```

REQTASKACK is Request Task Acknowledge Code. Valid values of REQTASKACK:

- 0: Accept
- 1: Denied. Invalid format.
- 2: Denied. At least one TASKID does not exist
- 3: Denied. Other error.

5.7 Scenarios

5.7.1 Scenario 1: Computation Tasks are Triggered by Events

Fig. 5.2 shows a communication diagram of a scenario for the host to define an analysis task and to configure for the task to be executed whenever a specific event happens on the equipment. Steps of the communication between the Host (Edge) and the Equipment:

- 1) Define an analysis task: The host sends a message to the equipment to define an analysis task. If the host needs to define a pre-defined task, the host will send a S22F1 message. If the host needs to define an analysis script task, the host will send a S22F7 message.

- 2) Acknowledge: The equipment sends S22F2 to acknowledge the message S22F1 or sends S22F8 to acknowledge the message S22F7 from the host.

- 3) Link the analysis task to an event: The host sends S22F3 to link the analysis task to an event

- 4) Acknowledge: The equipment sends S22F4 to acknowledge the message S22F3 from the host.

- 5) Enable the event: The host sends S2F37 to enable the event. S2F37 is defined in [55, 56].

- 6) Acknowledge: The equipment sends S2F38 to acknowledge the message S2F37 from the host. S2F38 is defined in [55, 56].

- 7) Perform data analysis: When the event is triggered on the equipment, the Data Analysis Engine on the equipment performs the analysis task, and at the same time, the equipment sends S6F11 to the host to inform the host that the event has been triggered. S6F11 is the Event Report Send message. The S6F11 message is defined in [55, 56].

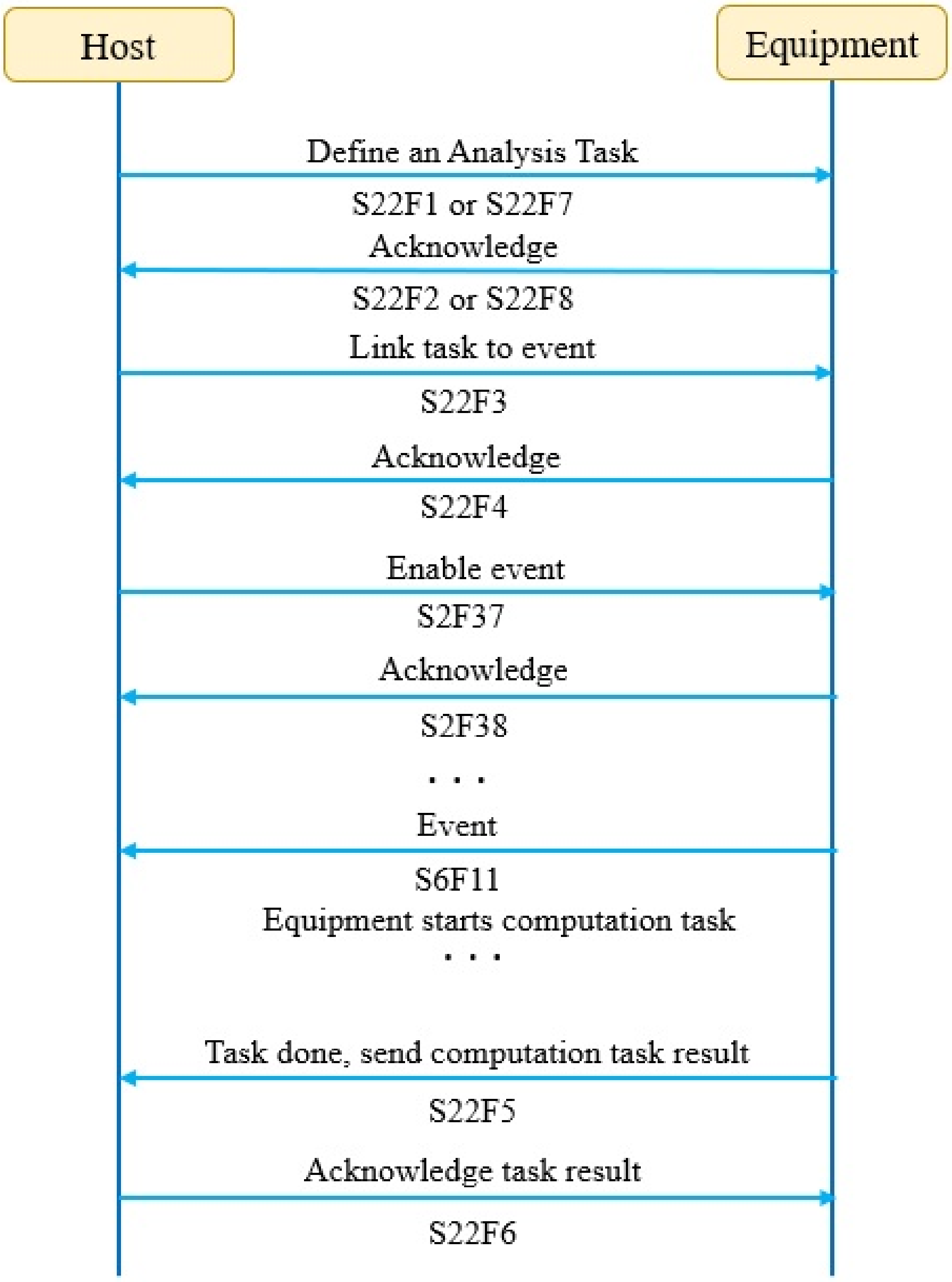


Figure 5.2: Communication Diagram - Computation Tasks Triggered by Events

- 8) Send results to the host: When the Data Analysis Engine finishes the data analysis task, it sends S22F5 to report the results to the host.

- 9) Acknowledge: The host sends S22F6 to acknowledge the message S22F5 from the equipment.

5.7.2 Scenario 2: Ad-hoc Performing Computation Tasks

Fig. 5.3 shows a communication diagram of a scenario for the host to set up an analysis task and request the equipment to perform the task. In this scenario, the computation task is performed only one time. Steps of the communication between the Host (Edge) and the Equipment:

- 1) Define an analysis task: The host sends a message to the equipment to define an analysis task. If the host needs to define a pre-defined task, the host will send a S22F1 message. If the host needs to define an analysis script task, the host will send a S22F7 message.

- 2) Acknowledge: The equipment sends S22F2 to acknowledge the message S22F1 or sends S22F8 to acknowledge the message S22F7 from the host.

- 3) Ad-hoc executing the task: The host sends S22F9 to request the equipment immediately executing the task.

- 4) Acknowledge: The equipment sends S22F10 to acknowledge the message S22F9 from the host. Then the equipment starts executing the task.

- 5) Send results to the host: When the Data Analysis Engine finishes the data analysis task, it sends S22F5 to report the results to the host.

- 6) Acknowledge: The host sends S22F6 to acknowledge the message S22F5 from the equipment.

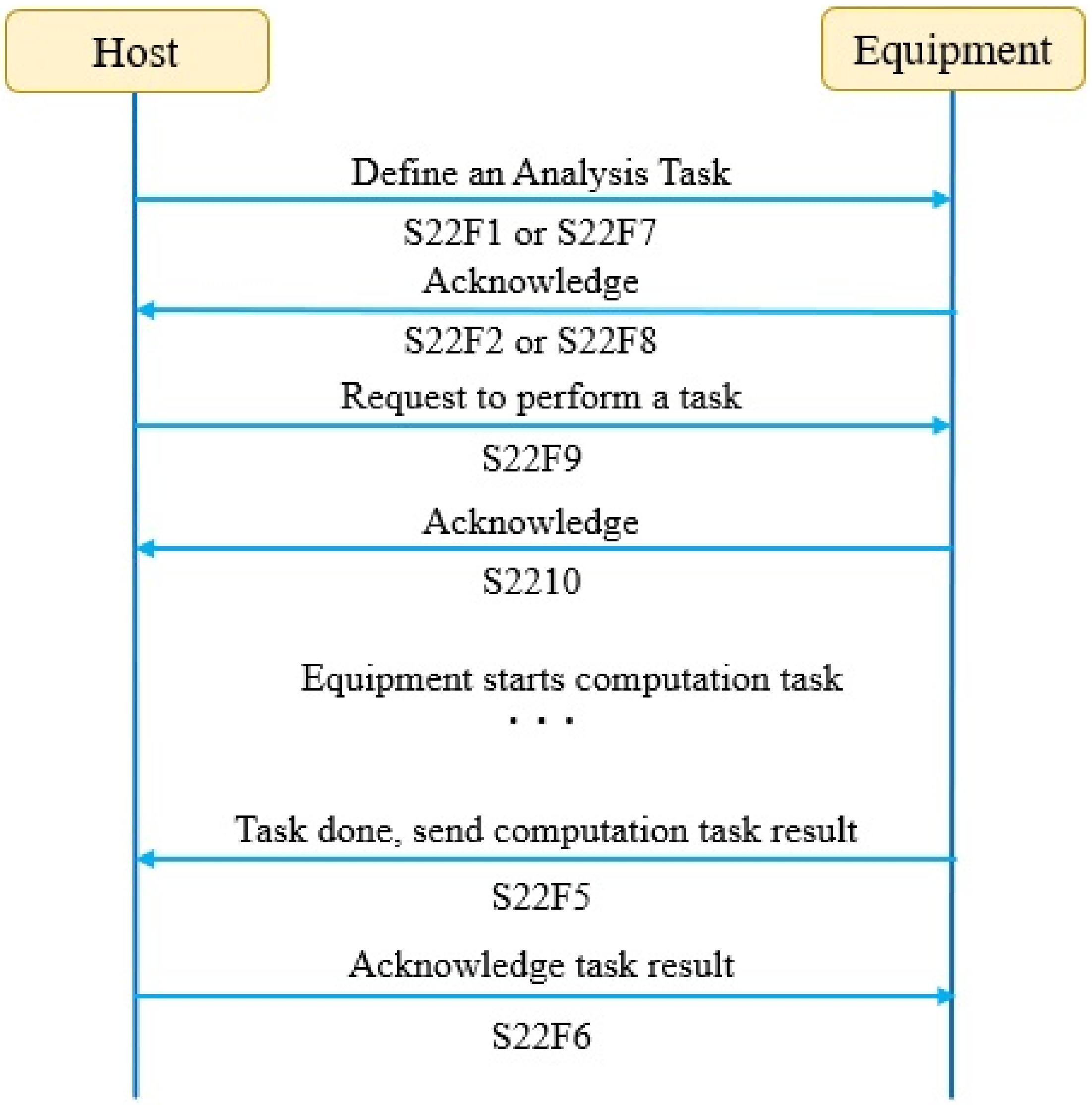


Figure 5.3: Communication Diagram - Ad-hoc Performing Computation Tasks

If the host needs the equipment to execute the task again, the process can be repeated from step 3. The computing task has been defined on the equipment and the host does not need to repeat step 1 to define the task again.

5.8 Summary

In this section, we conduct experiments to measure the latency when transferring data variables from the equipment to the host by using SECS/GEM messages. The experiments show that latency can significantly affect productivity in semiconductor manufacturing. We also propose a new system design with Data Analysis Engine and a new Stream 22 in SECS/GEM interface for the host to setup the equipment to perform real-time data analyses. Table 5.2 shows a summary of new proposed messages for supporting dynamic AI computing tasks.

Table 5.2: Summary of Dynamic AI Messages in SECS/GEM

SECS/GEM Message	**Description**	**Host to Equipment or Equipment to Host**
S22F1	Define Analysis Task	$H->E$
S22F2	Define Analysis Task Acknowledge	$E->H$
S22F3	Link Event to Analysis Task	$H->E$
S22F4	Link Event to Analysis Task Acknowledge	$E->H$
S22F5	Task Result Notify	$E->H$
S22F6	Task Result Notify Acknowledge	$H->E$
S22F7	Define Analysis Script Task	$H->E$
S22F8	Define Analysis Script Task Acknowledge	$E->H$
S22F9	Request to Perform Analysis Task	$H->E$
S22F10	Request to Perform Analysis Task Acknowledge	$E->H$

Chapter 6
Conclusions and Future Work

In this chapter, we first review the studies and benefits of offloading computation tasks from the edge layer to the equipment layer in section 6.1, followed by the summary of our proposed scheduling algorithm in section 6.2. Section 6.3 summarizes our proposed protocol for communication with SECS/GEM interface. Finally, we discuss future research directions in section 6.4.

6.1 Offloading Computing Tasks from the Edge to the Equipment

Real-time computation with low latency becomes a pivotal requirement in smart manufacturing. The current systems with computations on the cloud and edge layers increase latency due to the time for transferring data from the equipment layer to the edge layer. To address this issue, in this study, we revealed that offloading some computing tasks from the edge layer to the equipment layer can significantly reduce computation latency, thereby meeting the real-time requirement for smart manufacturing, especially in the semiconductor industry. Moreover, offloading computing tasks to the equipment layer can also reduce the overhead of handling all different formats of data received from the equipment layer. We devised an algorithm for selecting computing tasks that can be offloaded to the equipment level to achieve the lowest computing latency for smart manufacturing. Additionally, we trained different machine learning models based on current system conditions to predict the most efficient way to offload computing tasks from the edge layer to the equipment layer. To evaluate the effectiveness of our offloading technique, we conducted extensive performance comparisons between edge-based computing and equipment-based computing.

6.2 Task Scheduling for Smart Manufacturing

To address computing-intensive tasks in smart manufacturing systems, we developed a novel resource scheduler, with an expectation to allocate edge computing resources with awareness of the current workload of computing resources at the equipment layer. Our resource scheduler is capable of allocating computing resources at the equipment layer for computing-intensive tasks and free up resources at the edge layer for additional tasks, thereby improving computation throughput.

6.3 Communication Protocol in the SECS/GEM Interface

Lastly, we proposed a new protocol - Stream 22 - in SECS/GEM interface to allow the host computer on the edge layer to communicate with the equipment layer to support offloading tasks. The proposed protocol is a component of the *Communication Module* in *System Software Control* (see section 4.1). The proposed communication protocol allows the edge layer to assign analytic computing tasks to the equipment layer including data monitoring tasks, statistical process control tasks, and failure prediction tasks. This protocol also enables the edge layer to collect the results from the equipment layer after the computing tasks are completed. Furthermore, we designed a novel software architecture that can be seamlessly integrated into the equipment control to enable the equipment to receive and process tasks dispatched from the edge layer. We reckon that Stream 22 is expected to be adopted as a foundation to support dynamic AI computation with offloading tasks in the context of semiconductor smart manufacturing environment.

6.4 Future Research Directions

In terms of future research, there is potential for additional factors to be taken into consideration when determining whether to offload computational tasks from the edge layer to the equipment layer. For example, the current network performance of the system can affect the data transfer latency when there is a large amount of data involved.

In resource scheduling, some computing tasks are critical for the operations of the smart manufacturing system. We plan to look into scheduling high-availability resources for critical tasks, and its impact on the performance of the system. Additionally, we aim to investigate more features during the training of machine learning models to predict the performance of the system. These features may include, but not limit to, the ratio of critical tasks to regular tasks, as well as the percentage of high-availability resources in the overall smart manufacturing system.

Bibliography

[1] R. Alsurdeh, R. N. Calheiros, K. M. Matawie, and B. Javadi. Hybrid workflow provisioning and scheduling on cooperative edge cloud computing. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 445–454, 2021.

[2] A. Anaya, W. Henning, N. Basantkumar, and J. Oliver. Yield improvement using advanced data analytics. In *2019 30th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pages 1–5, 2019.

[3] D. Borsatti, G. Davoli, W. Cerroni, and C. Raffaelli. Enabling Industrial IoT as a Service with Multi-Access Edge Computing. *IEEE Communications Magazine*, 59(8):21–27, 2021.

[4] R. Busch, M. Wahl, P. Czerner, and B. Choubey. Yield prediction with machine learning and parameter limits in semiconductor production. In *2022 International Symposium on Semiconductor Manufacturing (ISSM)*, pages 1–4, 2022.

[5] S. Butte and S. Patil. Big data and predictive analytics methods for modeling and analysis of semiconductor manufacturing processes. In *2016 IEEE Workshop on Microelectronics and Electron Devices (WMED)*, pages 1–5, 2016.

[6] K. Cao, Y. Liu, G. Meng, and Q. Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[7] N. Chandrasekaran. Intelligent, data-driven approach to sustainable semiconductor manufacturing. In *2022 6th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*, pages 1–5, 2022.

[8] A. Chen, R.-S. Guo, and P.-J. Yeh. An effective spc approach to monitoring semiconductor manufacturing processes with multiple variation sources. In *Proceedings of ISSM2000. Ninth International Symposium on Semiconductor Manufacturing (IEEE Cat. No.00CH37130)*, pages 446–449, 2000.

[9] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin. Smart factory of industry 4.0: Key technologies, application case, and challenges. *IEEE Access*, 6:6505–6519, 2018.

[10] C.-C. Chen, W.-T. Su, M.-H. Hung, and Z.-H. Lin. Map–reduce–style job offloading using historical manufacturing behavior for edge devices in smart factory. *IEEE Robotics and Automation Letters*, 3(4):2918–2925, 2018.

[11] S.-J. Chen, H.-W. Liu, and W.-J. Wang. A Fault Tolerance Mechanism for Semiconductor Equipment Monitoring. In *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, pages 171–176, 2017.

[12] C.-F. Chien, Y.-H. Chen, and M.-F. Lo. Advanced quality control (aqc) of silicon wafer specifications for yield enhancement for smart manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 33(4):569–577, 2020.

[13] C.-F. Chien, W.-T. Hung, and E. T.-Y. Liao. Redefining monitoring rules for intelligent fault detection and classification via cnn transfer learning for smart manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 35(2):158–165, 2022.

[14] E. Collins. Big data in the public cloud. *IEEE Cloud Computing*, 1(2):13–15, 2014.

[15] V. Gezer, J. Um, and M. Ruskowski. An introduction to edge computing and a real-time capable server architecture. *Int. J. Adv. Intell. Syst.(IARIA)*, 11(7):105–114, 2018.

[16] M. Ghahramani, Y. Qiao, M. C. Zhou, A. O'Hagan, and J. Sweeney. AI-based modeling and data-driven evaluation for smart manufacturing processes. *IEEE/CAA Journal of Automatica Sinica*, 7(4):1026–1037, 2020.

[17] R. G. Goss and K. Veeramuthu. Heading towards big data building a better data warehouse for more data, more speed, and more users. In *ASMC 2013 SEMI Advanced Semiconductor Manufacturing Conference*, pages 220–225, 2013.

[18] L. Gu and W. Yu. One comprehensive method to analyze semiconductor manufacturing data by "piecewise" regression. In *2020 China Semiconductor Technology International Conference (CSTIC)*, pages 1–2, 2020.

[19] Y.-C. Hsieh, C.-Y. Chen, D.-Y. Liao, P. B. Luh, and S.-C. Chang. Equipment sensor data cleansing algorithm design for ml-based anomaly detection. In *2022 International Symposium on Semiconductor Manufacturing (ISSM)*, pages 1–4, 2022.

[20] H. Huang, Q. Ye, and Y. Zhou. 6g-empowered offloading for realtime applications in multi-access edge computing. *IEEE Transactions on Network Science and Engineering*, pages 1–14, 2022.

[21] P. K. Illa and N. Padhi. Practical Guide to Smart Factory Transition Using IoT, Big Data and Edge Analytics. *IEEE Access*, 6:55162–55170, 2018.

[22] T. Ito, W. Xueting, Y. Oomuro, and K. Nagashima. Advanced process control model for trench shape of power devices. In *2022 International Symposium on Semiconductor Manufacturing (ISSM)*, pages 1–4, 2022.

[23] V. K. Jain and S. Kumar. Big data analytic using cloud computing. In *2015 Second International Conference on Advances in Computing and Communication Engineering*, pages 667–672, 2015.

[24] D. Jia, M. Bayati, R. Lee, and N. Mi. Rita: Efficient memory allocation scheme for containerized parallel systems to improve data processing latency. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 329–336, 2020.

[25] C. Jiang, J. Wan, and H. Abbas. An edge computing node deployment method based on improved k-means clustering algorithm for smart manufacturing. *IEEE Systems Journal*, 15(2):2230–2240, 2021.

[26] M. Khakifirooz, M. Fathi, and K. Wu. Development of Smart Semiconductor Manufacturing: Operations Research and Data Science Perspectives. *IEEE Access*, 7:108419–108430, 2019.

[27] S. Khizar, M. D. de Amorim, and V. Conan. Offloading computing tasks beyond the edge: A data-driven analysis. In *2021 13th IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 79–83, 2021.

[28] N. Kim, H. Choi, J. Chun, and J. Jeong. Introduction of equipment level FDC system for semiconductor wet-cleaning equipment optimization and real-time fault detection. In *2022 33rd Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pages 1–4, 2022.

[29] D. Kobayashi, S. Yasuda, T. Iuti, and S. Ito. Application of natural language processing in semiconductor manufacturing. In *2022 International Symposium on Semiconductor Manufacturing (ISSM)*, pages 1–3, 2022.

[30] C.-C. Kuo, P.-C. Chen, and C.-T. Tseng. Application of big data science in high reliability automotive wafer yield management system and failure analysis. In *2022 International Symposium on Semiconductor Manufacturing (ISSM)*, pages 1–3, 2022.

[31] C. K. M. Lee, Y. Z. Huo, S. Z. Zhang, and K. K. H. Ng. Design of a Smart Manufacturing System With the Application of Multi-Access Edge Computing and Blockchain Technology. *IEEE Access*, 8:28659–28667, 2020.

[32] Y. Lee and Y. Roh. Regression yield analysis for semiconductor manufacturing based on fabrication conditions. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 6747–6748, 2022.

[33] X. Li, J. Wan, H.-N. Dai, M. Imran, M. Xia, and A. Celesti. A Hybrid Computing Solution and Resource Scheduling Strategy for Edge Computing in Smart Manufacturing. *IEEE Transactions on Industrial Informatics*, 15(7):4225–4234, 2019.

[34] Z. Li, Z. Wang, and W. Shi. Automatic wafer defect classification based on decision tree of deep neural network. In *2022 33rd Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pages 1–6, 2022.

[35] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu. Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network. *IEEE Transactions on Industrial Informatics*, 15(7):4276–4284, 2019.

[36] Y. Liu, G. Xiong, F. Zhu, S. Chen, L. Zhang, and X. Liu. Collaborative scheduling of computing tasks for edge computing. In *2021 China Automation Congress (CAC)*, pages 7824–7831, 2021.

[37] Y. Lu, L. Yang, S. X. Yang, Q. Hua, A. K. Sangaiah, T. Guo, and K. Yu. An intelligent deterministic scheduling method for ultralow latency communication in edge enabled industrial internet of things. *IEEE Transactions on Industrial Informatics*, 19(2):1756–1767, 2023.

[38] Z. Lv and R. Lou. Edge-fog-cloud secure storage with deep-learning-assisted digital twins. *IEEE Internet of Things Magazine*, 5(2):36–40, 2022.

[39] A. Mijuskovic, R. Bemthuis, A. Aldea, and P. Havinga. An enterprise architecture based on cloud, fog and edge computing for an airfield lighting management system. In *2020 IEEE 24th International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 63–73, 2020.

[40] J. Moon and J. Jeong. Smart Manufacturing Scheduling System: DQN based on Cooperative Edge Computing. In *2021 15th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, pages 1–8, 2021.

[41] J. Moyne, J. Samantaray, and M. Armacost. Big data emergence in semiconductor manufacturing advanced process control. In *2015 26th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pages 130–135, 2015.

[42] J. Moyne, J. Samantaray, and M. Armacost. Big data capabilities applied to semiconductor manufacturing advanced process control. *IEEE Transactions on Semiconductor Manufacturing*, 29(4):283–291, 2016.

[43] W. Na, Y. Lee, N.-N. Dao, D. N. Vu, A. Masood, and S. Cho. Directional link scheduling for real-time data processing in smart manufacturing system. *IEEE Internet of Things Journal*, 5(5):3661–3671, 2018.

[44] H. H. Nguyen. Dynamic ai computation tasks with secs/gem in semiconductor smart manufacturing. In *2022 International Symposium on Semiconductor Manufacturing (ISSM)*, pages 1–4, 2022.

[45] H. H. Nguyen, Y. Zhou, K. Kushagra, and X. Qin. Computation offloading from edge to equipment for smart manufacturing. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, pages 207–212, 2022.

[46] T.-D. Nguyen and E.-N. Huh. Ecsim++: An inet-based simulation tool for modeling and control in edge cloud computing. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 80–86, 2018.

[47] A. Nowitschkow, C. Saal, and O. Lohse. Factory data management: Definition and differentiation from manufacturing operations management. In *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, volume 1, pages 718–721, 2021.

[48] N. Peter. Fog computing and its real time applications. *Int. J. Emerg. Technol. Adv. Eng*, 5(6):266–269, 2015.

[49] M. S. K. Pheng and L. G. David. Artificial intelligence in back-end semiconductor manufacturing: A case study. In *2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, pages 1–4, 2022.

[50] G. Premsankar, M. Di Francesco, and T. Taleb. Edge computing for the internet of things: A case study. *IEEE Internet of Things Journal*, 5(2):1275–1284, 2018.

[51] Q. Qi and F. Tao. A Smart Manufacturing Service System Based on Edge Computing, Fog Computing, and Cloud Computing. *IEEE Access*, 7:86769–86777, 2019.

[52] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu. Edge computing in industrial internet of things: Architecture, advances and challenges. *IEEE Communications Surveys & Tutorials*, 22(4):2462–2488, 2020.

[53] A. K. Sandhu. Big data with cloud computing: Discussions and challenges. *Big Data Mining and Analytics*, 5(1):32–40, 2022.

[54] D. Scotece, C. Fiandrino, and L. Foschini. On the Efficiency of Service and Data Handoff Protocols in Edge Computing Systems. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2021.

[55] Semiconductor Equipment and Materials International (SEMI). Standard SEMI E30-0418, Specification for the Generic Model for Communications and Control of Manufacturing Equipment (GEM), 2018.

[56] Semiconductor Equipment and Materials International (SEMI). Standard SEMI E5-0219, Specification for SEMI Equipment Communications Standard 2 Message Content (SECS-II), 2019.

[57] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[58] S. Trinks and C. Felden. Edge Computing architecture to support Real Time Analytic applications : A State-of-the-art within the application area of Smart Factory and Industry 4.0. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2930–2939, 2018.

[59] C.-P. Tsai, H.-C. Hsiao, Y.-C. Chao, M. Hsu, and A. R. Chang. Bridging the Gap between Big Data System Software Stack and Applications: The Case of Semiconductor Wafer Fabrication Foundries. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1865–1874, 2018.

[60] T. Tsuda, S. Inoue, A. Kayahara, S.-i. Imai, T. Tanaka, N. Sato, and S. Yasuda. Advanced semiconductor manufacturing using big data. *IEEE Transactions on Semiconductor Manufacturing*, 28(3):229–235, 2015.

[61] J. Vater, L. Harscheidt, and A. Knoll. A Reference Architecture Based on Edge and Cloud Computing for Smart Manufacturing. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7, 2019.

[62] G. Villareal and J. Lee. The benefits of real-time cloud analytics in semiconductor. In *2020 International Symposium on Semiconductor Manufacturing (ISSM)*, pages 1–4, 2020.

[63] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, and C. Liu. Fog computing for energy-aware load balancing and scheduling in smart factory. *IEEE Transactions on Industrial Informatics*, 14(10):4548–4556, 2018.

[64] P. Wang, J. Mu, and Y. Zhang. Research on Edge Computing of Automatic Control System of Unattended Intelligent Manufacturing Equipment. In *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pages 1075–1078, 2021.

[65] Y. Wang, C. Zhao, S. Yang, X. Ren, L. Wang, P. Zhao, and X. Yang. Mpcsm: Microservice placement for edge-cloud collaborative smart manufacturing. *IEEE Transactions on Industrial Informatics*, 17(9):5898–5908, 2021.

[66] J. W. Webb and J. Webb. A method for storing semiconductor test data to simplify data analysis. In *2016 IEEE AUTOTESTCON*, pages 1–10, 2016.

[67] C. Yang, S. Lan, L. Wang, W. Shen, and G. G. Q. Huang. Big Data Driven Edge-Cloud Collaboration Architecture for Cloud Manufacturing: A Software Defined Perspective. *IEEE Access*, 8:45938–45950, 2020.

[68] C. Yang, W. Lou, Y. Liu, and S. Xie. Resource allocation for edge computing-based vehicle platoon on freeway: A contract-optimization approach. *IEEE Transactions on Vehicular Technology*, 69(12):15988–16000, 2020.

[69] S. Yasuda, T. Tanaka, M. Kitabata, and Y. Jisaki. Chamber and recipe-independent fdc indicator in high-mix semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 34(3):301–306, 2021.

[70] L. Yin, J. Luo, and H. Luo. Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. *IEEE Transactions on Industrial Informatics*, 14(10):4712–4721, 2018.

[71] J. Ying, J. Hsieh, D. Hou, J. Hou, T. Liu, X. Zhang, Y. Wang, and Y.-T. Pan. Edge-enabled cloud computing management platform for smart manufacturing. In *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*, pages 682–686, 2021.

[72] Z. Yu, Y. Lu, Q. An, C. Chen, Y. Li, and Y. Wang. Real-time multiple gesture recognition: Application of a lightweight individualized 1d cnn model to an edge computing system. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 30:990–998, 2022.

[73] J. Yuan, Y. Xiang, Y. Deng, Y. Zhou, and G. Min. Upoa: A user preference based latency and energy aware intelligent offloading approach for cloud-edge systems. *IEEE Transactions on Cloud Computing*, 2022.

[74] C. Zhang and W. Ji. Edge Computing Enabled Production Anomalies Detection and Energy-Efficient Production Decision Approach for Discrete Manufacturing Workshops. *IEEE Access*, 8:158197–158207, 2020.

[75] D. Zhang, N. Vance, and D. Wang. Demo abstract: Real-time heterogeneous edge computing system for social sensing applications. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 101–102, 2018.

[76] Q. Zhang, L. Gui, S. Zhu, and X. Lang. Task offloading and resource scheduling in hybrid edge-cloud networks. *IEEE Access*, 9:85350–85366, 2021.

[77] S. ZhangKun, Z. Zhisheng, X. Zhijie, and D. Min. A new approach to accelerate edge computing process based on multi-user computation offloading. In *2022 2nd International Conference on Computer, Control and Robotics (ICCCR)*, pages 186–190, 2022.

[78] C. Zhao, L. Ren, Y. Laili, and L. Lai. An architecture of knowledge cloud based on manufacturing big data. In *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, pages 4176–4180, 2018.

[79] Z. Zhao, L. Wang, L. Zha, W. Wang, L. Gong, Y. Gao, H. Gao, Y. Lei, and J. Dong. Research on edge cloud architecture of intelligent power plant. In *2022 IEEE 2nd International Conference on Digital Twins and Parallel Intelligence (DTPI)*, pages 1–3, 2022.

[80] Y. Zhou, S. Taneja, C. Zhang, and X. Qin. Greendb: Energy-efficient prefetching and caching in database clusters. *IEEE Transactions on Parallel and Distributed Systems*, 30(5):1091–1104, 2018.

[81] M. Zhu, X. Peng, Y. Sun, S. Fuyang, and D. Jiao. Simulation study of semiconductor communication protocol secs/gem. In *2021 International Conference on Wireless Communications and Smart Grid (ICWCSG)*, pages 148–152, 2021.

Appendices

Appendix A

Data From Experiments for Offloading From Edge to Equipment

Table A.1: Equipment Performance - CPU Utilization 1%

Data Size	**Total Computing Time with** $O(n)$	**Total Computing Time with** $O(nlogn)$	**Total Computing Time with** $O(n^2)$
2000	0.00999999	0.0139997	0.325999736
4000	0.019499302	0.027500391	1.314498901
6000	0.028564672	0.040564756	2.944564561
8000	0.034500122	0.050499201	5.254499436
10000	0.046998978	0.068499088	8.268498898
12000	0.047500134	0.073500157	11.8824997
14000	0.054497243	0.084995032	16.2074976
16000	0.059999943	0.095000029	21.14850021
18000	0.068503142	0.108502865	26.84500266
20000	0.074498653	0.119002103	33.16449953
22000	0.082499265	0.132999181	40.26399899
24000	0.089000702	0.144000292	47.90350127
26000	0.099000693	0.159000635	56.0320003
28000	0.103999853	0.16849947	65.11799908
30000	0.110998631	0.180499316	75.3109994
32000	0.117999315	0.192501068	85.14599896
34000	0.132499694	0.211999416	95.50099969
36000	0.133498907	0.218497753	107.5029997
38000	0.139000416	0.230000973	120.3210001
40000	0.146499872	0.241000414	132.3540001
42000	0.154498339	0.254998684	150.0849988
44000	0.161500931	0.267999172	168.8009992
46000	0.168000221	0.280499935	183.122
48000	0.175498724	0.291498899	201.1619985
50000	0.183000803	0.304000616	213.4540012

Table A.2: Equipment Performance - CPU Utilization 15%

Data Size	**Total Computing Time with** $O(n)$	**Total Computing Time with** $O(nlogn)$	**Total Computing Time with** $O(n^2)$
2000	0.014999628	0.018500328	0.33199954
4000	0.018497706	0.026998044	1.306995392
6000	0.025998592	0.038498163	3.002997398
8000	0.036498785	0.052999258	5.26699853
10000	0.045500755	0.067000388	8.296000718
12000	0.047996998	0.073498488	11.93249798
14000	0.06349452	0.09399374	16.26549419
16000	0.06099987	0.096998453	21.21799946
18000	0.068000078	0.108998775	26.85249972
20000	0.075501204	0.120501518	33.06550097
22000	0.083499193	0.133999347	40.24549699
24000	0.090002537	0.146502495	47.64300346
26000	0.098500252	0.158998967	56.25449824
28000	0.111001014	0.176002263	65.91649985
30000	0.112000465	0.182998657	75.35999989
32000	0.121501923	0.196501971	84.83150196
34000	0.12700057	0.208501577	96.10000229
36000	0.134001493	0.220500946	107.5040011
38000	0.142500639	0.235002756	120.475501
40000	0.149000645	0.244999886	132.5884997
42000	0.155997515	0.257499218	150.8334992
44000	0.167999029	0.275499821	169.747
46000	0.17099905	0.283998489	186.464999
48000	0.186999798	0.304499865	204.1690004
50000	0.187000751	0.311000347	220.4165006

Table A.3: Equipment Performance - CPU Utilization 35%

Data Size	**Total Computing Time with** $O(n)$	**Total Computing Time with** $O(nlogn)$	**Total Computing Time with** $O(n^2)$
2000	0.010999918	0.014499664	0.329500675
4000	0.01850009	0.027000666	1.442502261
6000	0.02649808	0.038497687	2.973998786
8000	0.034499168	0.050999164	5.289999723
10000	0.041001321	0.062500716	8.297000886
12000	0.047997236	0.073998213	11.91649771
14000	0.061598605	0.093998909	16.27299904
16000	0.061499596	0.09749651	21.31749654
18000	0.070498228	0.111999035	26.92399955
20000	0.075497865	0.121498823	33.24149704
22000	0.084499358	0.135501384	40.81200027
24000	0.091001034	0.148000002	47.99799943
26000	0.098499536	0.15950036	56.40800071
28000	0.10450077	0.170999765	65.45649886
30000	0.113003015	0.183499575	75.60000109
32000	0.121502399	0.196502685	85.41700387
34000	0.128500224	0.210500956	96.5369997
36000	0.135502815	0.222001552	108.3455026
38000	0.141496181	0.23249793	120.8934975
40000	0.151501895	0.250002146	135.6870022
42000	0.166500807	0.269499063	153.3685005
44000	0.164504528	0.273003101	170.6510053
46000	0.172998667	0.286999464	185.6289985
48000	0.179999828	0.300999403	203.8985002
50000	0.186001062	0.310002089	220.0145023

Table A.4: Equipment Performance - CPU Utilization 50%

Data Size	**Total Computing Time with** $O(n)$	**Total Computing Time with** $O(nlogn)$	**Total Computing Time with** $O(n^2)$
2000	0.015500307	0.017500162	0.343496323
4000	0.023000002	0.030499936	1.370999575
6000	0.028002023	0.040501832	3.115501165
8000	0.036002398	0.053002119	5.54450345
10000	0.04399991	0.066000938	8.692003012
12000	0.050500869	0.07700014	12.43550277
14000	0.063497782	0.094997644	16.51549649
16000	0.065003396	0.101003886	21.43050576
18000	0.072998762	0.113997459	27.47449755
20000	0.080500365	0.126999856	34.12850332
22000	0.086494922	0.137994766	41.78749895
24000	0.094498873	0.150999308	49.01199889
26000	0.10300231	0.164501667	57.86649871
28000	0.111498213	0.177998161	66.96399674
30000	0.119500398	0.193500041	77.64950085
32000	0.1249969	0.201998234	87.33599734
34000	0.13100028	0.213502645	98.97700143
36000	0.140995502	0.228498459	111.3654971
38000	0.150500536	0.242500782	123.9145022
40000	0.156499624	0.255000114	136.3150008
42000	0.184510469	0.288506746	158.024508
44000	0.182498932	0.292500973	173.7095003
46000	0.175998926	0.29300046	191.9594994
48000	0.190000772	0.311500549	212.2210011
50000	0.194495201	0.320001125	235.2645011

Table A.5: Equipment Performance - CPU Utilization 75%

Data Size	**Total Computing Time with** $O(n)$	**Total Computing Time with** $O(nlogn)$	**Total Computing Time with** $O(n^2)$
2000	0.01399827	0.016999245	0.373497009
4000	0.02149868	0.029498816	1.45550108
6000	0.030500174	0.043499709	3.313498974
8000	0.035997391	0.052997828	5.729500056
10000	0.049006223	0.073004007	9.129505634
12000	0.059003114	0.08650279	12.8855021
14000	0.067002828	0.100500877	18.67150026
16000	0.068001032	0.105499744	24.09150029
18000	0.073498488	0.115999699	30.62650085
20000	0.082000732	0.131497145	40.47550082
22000	0.095999957	0.152501822	44.00650049
24000	0.095989227	0.156990528	52.00348783
26000	0.11250186	0.177001	61.6220007
28000	0.112999677	0.184997081	72.78099966
30000	0.121004105	0.199998618	82.75899911
32000	0.129507541	0.210007667	93.62900829
34000	0.137999057	0.224998473	107.7025013
36000	0.148999453	0.244995595	118.4889996
38000	0.156997681	0.256996155	132.3199971
40000	0.162001372	0.267003775	145.6865013
42000	0.168502331	0.275500298	167.2755034
44000	0.180500031	0.295001745	187.6184993
46000	0.194998503	0.338499069	204.4334984
48000	0.193490267	0.323501348	224.1260006
50000	0.20199728	0.335497618	240.3464959

Table A.6: Equipment Performance - CPU Utilization 90%

Data Size	**Total Computing Time with $O(n)$**	**Total Computing Time with $O(nlogn)$**	**Total Computing Time with $O(n^2)$**
2000	0.014999629	0.018497468	0.424497366
4000	0.026001454	0.035501958	1.743502617
6000	0.037999153	0.051998376	3.915498972
8000	0.044496536	0.06801033	6.997498035
10000	0.055502415	0.084502459	10.70100164
12000	0.061500311	0.09449935	15.3684988
14000	0.06999755	0.107998133	20.98999762
16000	0.079499006	0.128499746	29.26650024
18000	0.093003749	0.147497415	34.64300013
20000	0.102500201	0.163500786	43.32750011
22000	0.116004229	0.178497791	52.58699846
24000	0.121000528	0.19850111	61.8670013
26000	0.128003359	0.212501765	72.39000297
28000	0.143000841	0.22950077	84.65600038
30000	0.151501417	0.250499487	97.2869997
32000	0.164002419	0.265004397	110.1500029
34000	0.172499419	0.282499552	124.2804992
36000	0.188515664	0.30700779	139.5220074
38000	0.194499493	0.31999898	156.0659985
40000	0.214999437	0.338000536	171.4825006
42000	0.209501266	0.346001625	192.689501
44000	0.220997334	0.364499807	221.5559986
46000	0.23199439	0.38549614	238.231997
48000	0.236501455	0.396500111	258.441
50000	0.253002167	0.417504549	281.3395023

Table A.7: Equipment Performance - CPU Utilization 99%

Data Size	**Total Computing Time with** $O(n)$	**Total Computing Time with** $O(nlogn)$	**Total Computing Time with** $O(n^2)$
2000	0.020498276	0.021499396	0.499998331
4000	0.023998976	0.032500506	1.984499455
6000	0.045501232	0.065503597	4.120001793
8000	0.057004214	0.078001977	7.323001862
10000	0.062001228	0.097501993	11.51700282
12000	0.080500126	0.122002363	16.57650041
14000	0.099003553	0.154004812	22.56000591
16000	0.103006363	0.159994603	29.55049872
18000	0.104990244	0.174499512	37.44049931
20000	0.121499539	0.193998814	46.06700135
22000	0.14150238	0.215502977	56.64850282
24000	0.142999887	0.235999822	66.52949881
26000	0.16699791	0.271500349	79.08500123
28000	0.177499533	0.290997267	91.35699964
30000	0.191500902	0.313496828	105.4665025
32000	0.205500603	0.338001251	119.4160013
34000	0.204001665	0.344002962	135.9595015
36000	0.213000775	0.353000403	151.654005
38000	0.239001751	0.395497561	170.0684993
40000	0.25449872	0.42399931	187.6294975
42000	0.260500431	0.425998211	217.7010002
44000	0.278496981	0.465997458	239.2329993
46000	0.298503637	0.480004549	262.3705049
48000	0.310505629	0.502503633	286.525003
50000	0.318001032	0.536998749	304.5695012

Table A.8: Edge Performance

Data Size	**Total Computing Time with** $O(n)$	**Total Computing Time with** $O(nlogn)$	**Total Computing Time with** $O(n^2)$
2000	1.733159465	1.736802701	2.029076699
4000	2.162054525	2.169898973	3.363678919
6000	1.764053024	1.775770105	4.487607397
8000	1.943600845	1.959469986	6.812240076
10000	2.231365361	2.251736798	9.857959904
12000	2.133716332	2.158566461	13.31153534
14000	2.508967311	2.538674981	17.8970913
16000	2.4974642	2.531521341	22.61502363
18000	2.865437801	2.904316241	28.13518291
20000	2.632671649	2.676464612	34.12620812
22000	3.027922548	3.07635466	40.06665722
24000	3.480059031	3.532942656	48.18244994
26000	2.994064965	3.051556268	56.01033584
28000	3.478096574	3.540417998	64.10312756
30000	3.610690477	3.676864985	74.37795604
32000	3.247574554	3.318192945	82.91229104
34000	3.323814651	3.400343677	93.81036856
36000	3.553130818	3.634977531	104.7217047
38000	3.643384988	3.730532462	116.2898655
40000	3.853545822	3.94382564	126.9629783
42000	3.730803067	3.830066259	143.081718
44000	3.887523876	3.992079006	155.8463445
46000	3.897168078	4.005313076	170.9015299
48000	4.097793565	4.210493313	184.6555419
50000	4.127615336	4.247290495	200.8538202

Appendix B

Data For Scheduling Algorithm Comparison - Hybrid Computing Solution (HCS) vs. Offloading

Table B.1: Tasks Scheduling Data - HCS vs Offloading

Dataset	**Total Time HCS**	**Total Time With Offload**	**Through-put HCS**	**Through-put With Offload**	**Total Wait Time HCS**	**Total Wait Time With Offload**	**Avg Wait Time HCS**	**Avg Wait Time With Offload**
1	637	459	1.57	2.18	195602	128431	195.60	128.43
2	603	435	1.66	2.30	185947	122887	185.94	122.89
3	653	474	1.53	2.11	213336	139386	213.34	139.39
4	612	451	1.63	2.22	195899	131778	195.90	131.78
5	616	442	1.62	2.26	190326	124829	190.32	124.83
6	611	432	1.64	2.31	188641	127074	188.64	127.07
7	638	460	1.57	2.17	192557	124870	192.56	124.87
8	624	455	1.60	2.20	197397	131502	197.40	131.50
9	626	461	1.60	2.17	180300	122993	180.30	122.99
10	644	456	1.55	2.20	208422	135511	208.42	135.51

Appendix C

Data For Scheduling Algorithm Comparison - Different Wait Time Limits

Table C.1: Tasks Scheduling Data - Different Wait Time Limits (WTL)

WTL	**Total Executing Time**	**Throughput**	**Total Wait Time**	**Avg Wait Time**	**Avg Wait Time Priority 1**	**Avg Wait Time Priority 2**	**Avg Wait Time Priority 3**	**Avg Wait Time Priority 4**	**Avg Wait Time Priority 5**
No priority	459	2.18	128431	128.43	126.50	123.48	134.64	124.69	133.02
WTL = 10	461	2.17	126228	126.23	119.99	119.76	131.71	124.47	135.34
WTL = 20	461	2.17	126745	126.75	113.01	116.99	132.57	127.91	143.44
WTL = 30	463	2.16	124096	124.10	100.66	107.87	129.40	132.40	150.80
WTL = 40	473	2.11	124803	124.80	86.31	104.07	132.08	135.92	166.13
WTL = 50	466	2.15	123877	123.88	76.68	86.35	129.63	143.50	184.20
WTL = 60	462	2.16	121739	121.74	55.85	79.56	126.16	149.74	198.40
WTL = 70	469	2.13	124161	124.16	42.48	73.01	126.95	157.48	221.54
WTL = 80	471	2.12	124637	124.64	21.98	62.10	127.25	168.75	244.15
WTL = 90	472	2.12	127470	127.47	14.96	58.25	127.95	182.26	255.73
WTL = 100	469	2.13	126774	126.77	8.25	58.70	140.27	182.61	247.62
No WTL	462	2.16	124620	124.62	5.86	58.21	133.98	172.30	254.44

Appendix D

Supporting Variables for New SECS/GEM Communication Protocol

The Table D.1 lists all variables that would be used in new proposed messages in Section 5.6.

Table D.1: Variables in Dynamic AI Messages

Variable	**Description**	**In Message**	**Values**
DATACK	Define Analysis Task Acknowledge Code	S22F2	0: Accepted 1: Denied, invalid format 2: Denied, at least one TASKID was already defined 3: Denied, at least one VID does not exist 4: Denied, at least one KEYWORD is unknown 5: Denied, other error
LATACK	Link Analysis Task Acknowledge Code	S22F4	0: Accepted 1: Denied, invalid format 2: Denied, at least one CEID was already defined
			Continued on next page

Table D.1 – continued from previous page

Variable	Description	In Message	Values
			3: Denied, at least one CEID does not exist 3: Denied, at least one TASKID does not exist 5: Denied, other error
TRACK	Task Result Acknowledge Code	S22F6	0: Accepted 1: Denied, invalid format 2: Denied, at least one TASKID does not exist 3: Denied, the number of results does not match the definition of the task TASKID 4: Denied, other error
DASTACK	Define Analysis Script Task Acknowledge Code	S22F8	0: Accepted 1: Denied, invalid format 2: Denied, at least one TASKID was already defined 3: Denied, at least one VID does not exist 4: Denied, invalid script 5: Denied, other error
REQTASKACK	Request Task	S22F8	0: Accepted

Continued on next page

Table D.1 – continued from previous page

Variable	Description	In Message	Values
	Acknowledge Code		1: Denied, invalid format 2: Denied, at least one TASKID does not exist 3: Denied, other error