UNIVERSITY OF THESSALY

Department of Informatics and Telecommunications

Doctoral Dissertation

Data and Uncertainty Driven Optimization for Dynamic Edge Computing

by

Athanasios Moustakas

Lamia, April, 2025

Three-Member Supervisory Committee

Chairperson of the Supervisory Committee:

Dr. Konstantinos Kolomvatsos, University of Thessaly,
Department of Informatics and Telecommunications

Member of the Supervisory Committee:

Dr. Georgios Stamoulis, University of Thessaly,
Department of Electrical and Computer Engineering

Member of the Supervisory Committee:

Dr. Christos Anagnostopoulos, University of Glasgow,
School of Computing Science

Seven-Member Evaluation Committee

Abstract

In the era of digital transformation, the convergence of groundbreaking computing paradigms, namely the Internet of Things, Edge Computing, Cluster Computing, and Pervasive Computing stands at the forefront of technological evolution. These paradigms are not only reshaping the technological landscape, but are also redefining the boundaries of efficiency, scalability, resource optimization, and adaptability in distributed systems. This thesis embarks on an in-depth exploration of these interconnected technologies, focusing on enhancing data processing and machine learning efficiency in edge computing environments, driven by both data characteristics and uncertainty.

The research develops novel frameworks and algorithms in five key areas: (1) a synopsis-based similarity extraction framework that enables distributed nodes to identify and collaborate with peers holding similar datasets, optimizing data exchange and reducing redundancy in distributed environments; (2) a data-aware training acceleration framework that optimizes machine learning model training in resource-constrained environments, reducing the training time and improving efficiency while maintaining models' performance over time; (3) a transfer learning model that enhances knowledge transfer between nodes in edge environments, leveraging model selection techniques to improve accuracy and efficiency in resource-constrained settings; (4) a drift-aware task management mechanism that dynamically adapts task allocation strategies based on evolving data distributions, ensuring efficient resource utilization and system adaptability in edge computing; (5) a correlation-aware task scheduling approach that optimally assigns tasks by detecting dependencies and resource availability, improving execution efficiency and minimizing bottlenecks in distributed systems.

These contributions provide robust solutions to critical challenges in edge and cluster computing environments, where traditional methods struggle with scalability, resource constraints, and real-time data evolution. By integrating advanced statistical methods, adaptive learning, and task optimization techniques, the frameworks proposed in this thesis pave the way for more efficient and adaptive distributed systems. These developments contribute to the ongoing evolution of Internet of Things, Edge Computing, and related domains, offering practical applications in smart cities, autonomous systems, and large-scale industrial networks.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Kostas Kolomvatsos, for his invaluable guidance, continuous support, and insightful feedback throughout the duration of this journey. His mentorship has been instrumental in shaping my academic growth.

I am deeply grateful to my family for their unconditional love, support, and patience during this demanding period. To my partner, Georgia, thank you for your endless encouragement and understanding.

Patience and perseverance, inspired by the words and actions of my beloved, late grandparents.

This work is dedicated to my beloved uncle Nikos, who unexpectedly left us for the company of angels...

Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	**Definition**
IoT	Internet of Things
EC	Edge Computing
CC	Cluster Computing
PC	Pervasive Computing
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
TL	Transfer Learning
DRL	Deep Reinforcement Learning
RL	Reinforcement Learning
DNN	Deep Neural Network
NN	Neural Network
MSE	Mean Squared Error
MAPE	Mean Absolute Percentage Error
PCC	Pearson Correlation Coefficient
CDF	Cumulative Distribution Function
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error

Nomenclature

η	The learning rate of the optimizer.
γ	The momentum of the optimizer.
$\mathcal{L}$	The training loss of $\mathcal{T}$.
$\mathcal{N}$	The set of autonomous nodes.
$\mathcal{S}$	Synopsis of the dataset D_i.
$\mathcal{T}$	The training process performed locally at an autonomous node.
$\mathcal{T}^i$	The stream of tasks at the ith node.
c	Number-indicator for correlation between Sl and Sp.
cg_j	The number of the correlated group that task t_j belongs to.
$cgint$	1×N-dimensional vector containing the intersected vectors for all the correlated groups.
$cgint_i$	The intersected vector for the correlated group i.
$CORR(Sl, Sp)$	l-dimensional correlation vector.
$CORR_i(Sl_i, Sp_i)$	Correlation of the Sl_i, Sp_i pair.
d^i	Services demand vector.
d^ij	Demand for the jth service at the ith node.
D_i	The dataset belonging to the i-th node.
$D_{\mathcal{T}}$	Local data processed by $\mathcal{T}$.
$D_{\overline{\mathcal{T}}}$	Local data that are not part of $\mathcal{T}$.
l^it	Load of the i-th node at time t.
l_i	The load of the i-th node.
m_i	d-dimensional data vector containing the mean value of the dataset for each dimension.
mb	d-dimensional data vector containing the mean value of the data-batch for each dimension.
mp, mq	d-dimensional data vectors containing the mean values for each dimension of P, Q.
n_i	The i-th edge computing node.
ncg_i	The number of correlated groups for the i-th node.
NFM_i	i-th node's mean vector with d dimensions.
NFV_i	i-th node's variance vector with d dimensions.
$NROV_i$	i-th node's range of values vector with d dimensions.
nts_i	The accumulated task size of node n_i.
op_i	The number of suitable nodes for the i-th correlated group's execution.
P, Q	Samples with M data vectors.
P_i	The processing capability of node n_i.

pbc	Percentage of tasks scheduled based on correlation.
pm_{ij}	The metric's value for node i at period j.
ptf	Percentage of task failures.
q_i	List comprising the tasks assigned to node n_i.
s	Synopsis vector with l dimensions.
Sl	Synopsis of the local node.
$Sl[j]$	Synopsis of the local node at time j.
Sl_i	The i-th dimension belonging to the synopsis vector of the local node.
Sp	Synopsis of the peer node.
sp	Number-indicator for the distance-based similarity extracted by the probability between Sl and Sp.
$Sp[j]$	Synopsis of the peer node at time j.
Sp_i	The i-th dimension belonging to the synopsis vector of the peer node.
su_j	N-dimensional vector representing the suitable nodes for the j-th task's execution.
sz_i	The size of task t_i.
sz_j	The size of task t_j.
sz_{rt}	The size of the model retraining task.
T	Discrete time domain.
tfm_j	j-th task's desired range of mean vector with d dimensions.
tfv_j	j-th task's desired range of variance vector with d dimensions.
tm_{ij}	Vector with d dimensions comprising the mean of every feature for the j-th period.
$trov_j$	j-th task's desired range of values vector with d dimensions.
v_i	d-dimensional data vector containing the variance value of the dataset for each dimension.
v_{ij}	Vector with d dimensions comprising the variance of every feature for the j-th period.
vb	d-dimensional data vector containing the variance value of the data-batch for each dimension.
vp, vq	d-dimensional data vectors containing the variance values for each dimension of P, Q.
W_t^i	Weights of the i-th node at time t.
x	Data vector with d dimensions.
x_i	The realization of a specific data dimension.

1 Introduction

The convergence of Internet of Things (IoT), Edge Computing (EC), Cluster Computing (CC), Pervasive Computing (PC), and Artificial Intelligence (AI) forms the crux of contemporary digital ecosystems, presenting unparalleled technological advancements alongside complex challenges. This thesis delves into these interconnected paradigms, scrutinizing their roles, synergies, and the collective impact they wield on modern digital infrastructures. The rapid growth of data generated by distributed systems, such as those found in IoT networks and cloud computing environments, has necessitated the development of advanced data processing and Machine Learning (ML) techniques. This thesis explores several cutting-edge methodologies aimed at enhancing the efficiency and effectiveness of data handling and learning processes in distributed systems. The focus is on five key areas:

- similarity extraction;
- data-aware incremental learning;
- Transfer Learning (TL);
- task management/scheduling and task offloading.

Each of these areas addresses a unique challenge faced by distributed systems. Similarity extraction techniques enable the identification of related datasets across distributed nodes, facilitating collaborative processing, knowledge sharing and redundancy reduction. Commonly used approaches in distributed systems include locality-sensitive hashing for scalable similarity search in high-dimensional spaces [1], and vector-based similarity search through efficient approximate nearest neighbor algorithms [2], so as to enable efficient data alignment and retrieval. Data-aware incremental learning provides a framework for continuously updating ML models as new data arrives, ensuring that the models remain relevant and accurate over time. This is crucial in dynamic environments where data distributions are subject to changes over time. In [3] the authors introduce an approach which preserves knowledge on not recently encountered tasks through selective decelerating learning on the weights related to them. Furthermore, continual learning strategies on streaming data where data distributions alter over time are proposed in [4].

TL allows models trained in one context, e.g. large-scale source domain to be adapted for use in another domain with limited data availability, significantly reducing the time and computational resources required for training new models. In computer vision, fine-tuning a pretrained Convolutional Neural Network (CNN), e.g., ResNet, VGG, has demonstrated strong performance in domain-specific applications such as medical imaging and remote sensing [5]. Similarly, in natural language processing, pretrained transformer-based models like BERT have shown that previously acquired knowledge can be efficiently transferred to downstream tasks with minimal fine-tuning, yielding state-of-the-art results across benchmarks [6]. Finally effective task management and scheduling are crucial for optimizing resource utilization and ensuring timely execution of processes in distributed environments. In [7] a framework that utilizes Deep Reinforcement Learning (DRL), DeepRM, is introduced. This framework can learn scheduling policies that adapt to varying resource demands and job characteristics, outperforming classical methods in terms of job completion time and system utilization. Additionally scheduling algorithms that consider task deadlines and energy constraints have been shown to significantly improve quality of service and system efficiency [8].

The existing literature often overlooks critical aspects such as the continuous update requirements for ML models in distributed systems, the need for efficient similarity extraction techniques across distributed datasets, and the challenges of transferring learning models across different contexts in EC environments. Despite the exponential growth of IoT networks and cloud computing driving advancements in data processing and ML, traditional methods for task management, scheduling, and learning struggle with the resource constraints of EC. These constraints are particularly significant in data similarity extraction, adaptive task management , and effective TL. This thesis addresses these gaps by proposing innovative frameworks and algorithms specifically designed for edge and CC environments. It builds upon the outcomes of five research papers [9], [10], [11], [12], [13], each tackling different aspects of the aforementioned challenges. We integrate their methodologies and findings into a set of complementary approaches that collectively enhance distributed system performance through advanced ML techniques. These frameworks provide robust solutions to the unique challenges posed by edge and CC environments, paving the way for more efficient and adaptive distributed systems.

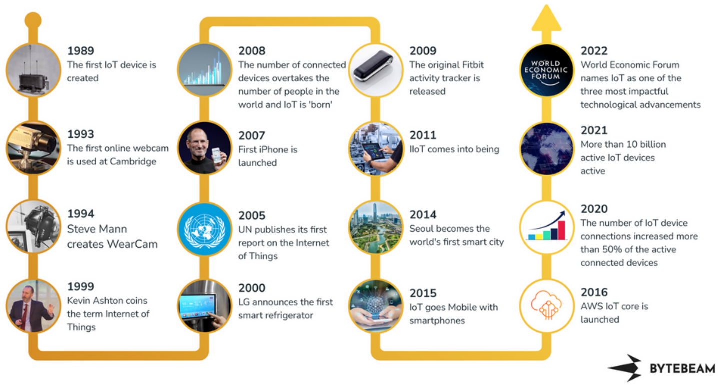

1989
The first IoT device is created
1993
The first online webcam is used at Cambridge
1994
Steve Mann creates WearCam
1999
Kevin Ashton coins the term Internet of Things
2008
The number of connected devices overtakes the number of people in the world and IoT is 'born'
2007
First iPhone is launched
2005
UN publishes its first report on the Internet of Things
2000
LG announces the first smart refrigerator
2009
The original Fitbit activity tracker is released
2011
IIoT comes into being
2014
Seoul becomes the world's first smart city
2015
IoT goes Mobile with smartphones
2022
World Economic Forum names IoT as one of the three most impactful technological advancements
2021
More than 10 billion active IoT devices active
2020
The number of IoT device connections increased more than 50% of the active connected devices
2016
AWS IoT core is launched
BYTEBEAM

Figure 1: Timeline of the historical development and evolution of IoT

1.1 The Internet of Things

The term IoT refers to a network of interconnected physical devices [14]. IoT represents a significant advance in digital connectivity, as billions of devices are now connected to the internet, gathering and exchanging data worldwide. Digital intelligence is now present in the actual world thanks to this enormous network of linked gadgets, giving rise to a level of digital omnipresence that was previously only seen in science fiction. The percentage of IoT devices in regard to total devices has exploded from 9% in 2010 to 66% in 2023 and is expected to reach 75% by 2025 [15]. It is important to note that the number of devices has experienced enormous growth as well. More specifically the number has gone from 8.8 billion devices in 2010 to 30 billion devices in 2023 and is projected to reach 41.2 billion by 2025. This growth aligns with key milestones in IoT development [16], such as those illustrated in Figure 1.

The aforementioned devices are equipped with sensors, actuators, connection capabilities, memory and software. The main objectives are, capturing data from the environment, storing them, perform computations and sending raw data or calculated results to other IoT devices, an Edge server or the Cloud, through the network for further processing. In detail, sensors are utilized so as to capture data from their environment. Some examples include temperature/CO2 measurements in a room, the soil moisture in an field, etc. Actuators guide the behaviour of the devices based on a predefined control system, making task automation feasible. Various instructions are given based on the data that are captured by the device. For example a ventilation system is opened when the CO2 in the room exceeds a certain threshold value or an air conditioner stops working when the desirable temperature is achieved. Connection capabilities allow the interaction between devices through Wi-Fi or Bluetooth and the transfer of the collected data to the Edge or the Cloud through the Internet. Transferring the data aims to utilize the increased computational capabilities that both Edge and Cloud present compared to IoT devices, in order to provide insights through advanced processing and analysis.

The architecture of IoT consists of multiple elements that collaborate in order to capture data from the environment, transfer them if needed, process these data and take decisions that lead to actions based on the insights derived through processing of the data. These elements are structured in four basic layers [17], which include:

- **Sensing Layer**: it represents the physical layer of the IoT architecture and is the basis for all operations. It consists of sensors, actuators and connects the real world to the digital one.

- **Network Layer**: this layer is responsible for transmitting the data captured by the previous layer, between devices, to a server or the Edge/Cloud. Some basic components are Internet/network gateways and data acquisition systems.

- **Data Processing Layer**: in this layer, the data transmitted through the previous layer are subject to advanced processing through ML algorithms, analytics and aggregation techniques. Based on this processing, insights are derived which are, then, used for decision making.

- **Application Layer**: this layer allows users to interact with the system through an interface, and use services provided based on the processing of data conducted in the previous layer.

IoT has a significant impact on various industries including healthcare, smart buildings, predictive maintenance, agriculture and logistics among others. It would be safe to say that it vastly revolutionizes these industries, through the ability of extracting knowledge from the data. Commonly used practices are altering as new possibilities arise. By enabling automation, IoT causes a foundational shift in the way industries operate, interact and innovate. IoT integration will only improve in the future, having a greater and greater impact on various industries as it enhances productivity, efficiency, cost reduction and innovation. To prosper in this networked environment, businesses will have to embrace IoT technologies and fully utilize their capabilities. To thoroughly understand the impact of IoT in industry we will examine a practical use case scenario.

Case 1: In the context of smart buildings, an 'intelligent' Heating, Ventilation, and Air Conditioning (HVAC) system, that is enhanced with IoT capabilities, can optimize energy efficiency, occupant comfort and cost reduction. By monitoring building/room measurements such as CO2 concentration, energy consumption, temperature, humidity, etc. through devices equipped with sensors, the system can adjust heating, turn on/off devices, etc. The basic components of such a system are:

- the IoT devices, placed throughout the building to collect data;
- the actuators that are integrated into HVAC components to adjust the HVAC system operation based on the observed data;
- a control system that processes the data in real time through AI/ML algorithms and sends instructions to actuators based on the knowledge derived from the data;
- a user interface, where occupants can receive notifications, view building conditions and set preferences.

1.2 Edge Computing

EC is an emerging distributed computing paradigm that aims to bring storage and computation near to the location that data are initially generated [18]. Hence, improving response times, reducing latency, saving bandwidth and providing results in real time. This contradicts the well established Cloud computing , where data are transferred to remote data centers to be processed. The core idea of the EC is to minimize the distance between data and computational resources. EC shifts the processing of data from remote data centers to distributed architectures at the edge of the network. The basic principles of EC [18] are:

- **Decentralization**: EC decentralizes data storage and processing as it assigns these tasks to a wide network of devices and computers/servers that work/function as a whole;

- **Real-time Processing**: By processing data close to the source, EC makes real-time processing viable;
- **Latency Reduction**: By performing data processing close to the source, EC significantly reduces the reaction time of the system to data;
- **Bandwidth Conservation**: By processing the majority of data close to the source, EC minimizes bandwidth usage;
- **Scalability**: EC enables scalability through utilizing multiple devices and computers, making it feasible to adapt workloads and redistribute resources based on application needs;
- **Resilience**: By distributing processing to multiple independent edge nodes, EC enhances the resilience of the system. Even if some nodes fail the system will continue to function;
- **Security**: Transmitting data through various networks to the Cloud increases the probability for unauthorized access due to security breaches in some of these networks. Instead by sending data to the edge, this probability is significantly reduced, as data remain in the same network they are captured in.

EC consists of four basic components, which are synchronized to process data at the edge. The basic components of EC [19] are listed below:

- **Edge devices**: The aforementioned IoT devices, present at the edge of the network.
- **Gateways**: functions that translate IoT protocols to data formats that are compatible for edge processing and aggregation. These enable communication between IoT devices and edge nodes.
- **Edge nodes**: The computing resources, i.e., computers/servers, located close to the source of data. These nodes store the data sent to them by the IoT devices and perform processing activities.
- **Communication networks**: Wired and wireless technologies such as Wi-Fi, 5G and Ethernet, that allow data transfer between IoT devices and edge nodes.

EC has already been adopted in many real-world applications to improve data management and enhance operational efficiency. Some examples include smart cities, healthcare, retail, energy sector, agriculture and autonomous vehicles. We extend the use case provided in the previous subsection, Case 1, to exhibit the benefits that EC provides in such a system. As mentioned before, the main targets are to optimize energy efficiency, occupant comfort and cost reduction. Data captured by the established IoT devices throughout the building can be processed in real time with EC. Hence, allowing instant detection of anomalies or alterations in occupancy and any other measurement being monitored. This gives the system the ability to make rapid adjustments, such as adjusting the temperature based on occupancy or turning on the ventilation based on CO2 values to improve air quality. The superior computational capabilities of edge nodes compared to IoT devices makes it feasible to process the data through ML/AI algorithms, to derive knowledge that enables the system to adapt to patterns in building measurements. For example, the system can predict future CO2 values ahead of time and proactively open the ventilation, if 'high' CO2 values are expected, to maintain optimal air quality in the room.

1.3 Cluster Computing

CC is a collaborative approach, where multiple computers work as a whole in order to compose a powerful system [20]. Thus allowing the processing of large data sets and performance of resource intensive computations that would be unfeasible in a single computer. The vast amount of available data significantly prevails the technological advancements in terms of processing power and storage capacity. CC and its ability to combine the resources of multiple computers is a vital solution to this problem. Each computer in a cluster is referred to as a node. A cluster consists of several nodes (individual computers), connected through a fast Local Area Network (LAN). These nodes work together under the guidance of a specific node (server/master) which is responsible for receiving tasks and distributing the workload to other nodes in the system. This distributed computing approach is extremely effective for tasks that can be parallelized, so large tasks can be divided in smaller sub-tasks that can be solved concurrently.

CC has revolutionized large-scale data processing. Frameworks such as Apache Hadoop [21] and Apache Spark [22] are built on the principles of CC, enabling efficient processing of vast amounts of data across many nodes.

These technologies are the base for big data analytics, allowing organizations to derive knowledge from huge datasets at remarkable speeds. CC has also had a significant impact in advancing scientific research, accelerating discoveries and innovations. Complex data analysis and simulations that used to take months to complete are now realized in days or even hours. Clusters can be categorized based on their purpose or the type of tasks they perform. Some common types include [23]:

- **High-Performance Computing Clusters**: designed for intensive computation tasks that require a large amount of processing power, such as simulations and data analysis;
- **High-Availability Clusters**: focused on ensuring that services remain available at all times, even in the event of hardware or software failures;
- **Load-Balancing Clusters**: used to distribute incoming requests among multiple servers to improve the performance of web services, databases, and other networked services;
- **Data Processing Clusters**: optimized for processing and analyzing large datasets, such as those used in big data analytics and ML applications;
- **Storage Clusters**: provide reliable and scalable storage solutions, distributing data across multiple nodes to improve access speed and data redundancy.

CC's ability to process large volumes of data efficiently complements IoT and EC by providing a robust back-end infrastructure that can handle the aggregation, processing, and long-term storage of the collected data. CC's integration with IoT and EC is extremely beneficial for data processing. Edge nodes can process data locally for immediate insights, and more complex analyses are offloaded to CC setups, which can be formed through combining various edge nodes, for in-depth processing. This hybrid approach not only enables real-time analytics and decision-making at the edge but also leverages the computational power of clusters for resource intensive tasks. As IoT devices multiply, leading to an exponential growth in data generation, CC offers the necessary scalability. More nodes can be added to the cluster with minor modifications, ensuring that the system will be able to cope with this immense data increase. CC distributive nature provides fault tolerance, which is crucial for IoT applications where data availability and integrity are crucial. If one node fails, others can undertake the execution of tasks, ensuring uninterrupted data processing. Another important benefit of CC is load balancing, the workload can be divided in smaller tasks, which are then assigned to multiple nodes in the system. Hence, ensuring that all nodes have tolerable computational load, enhancing the system's reliability and performance.

We further extend the aforementioned Case 1, to depict how CC can improve the computational power, scalability and reliability of the HVAC system. A cluster of nodes/servers can be utilized to aggregate data collected by edge nodes in various buildings (instead of one building). Upon these aggregated data, advanced analytics/processing and ML/ Deep Learning (DL) algorithms can be performed in order to identify trends and predictive maintenance needs. Complex processing that requires data from various sources and induces heavy computational burden can be realized. Also, simulations for HVAC operations under various conditions can be generated in order to contribute to optimizing the overall system's efficiency.

1.4 Pervasive Computing

PC refers to the concept of integrating computing capabilities (usually microprocessors) into everyday objects, making technology and computing power widely available. PC transcends the traditional desktop computing setting as it includes various types of devices such as laptops, smartphones, smartwatches, notebooks, tablets, IoT devices etc. In this setting, computing devices are omnipresent, embedded in everyday environments and activities, and operate discreetly in the background to support people's daily lives. Connectivity functionalities are integrated into all these devices, making it possible to interact with one another and automate tasks, requiring minimal human effort if any. PC systems are aware of their context and environmental conditions such as location, time of day, occupancy, user behavior etc. Hence, allowing them to anticipate needs and provide relevant information/services to the end user without explicit instructions. PC systems can scale based on the number of devices and the scope of services, supporting many applications from smart homes to large-scale urban environments.

The relationship between PC and the underlying infrastructures of IoT, EC, and CC is foundational to the modern digital ecosystem, enabling seamless, efficient, and scalable applications that improve people's lives. IoT devices are the sensory and actuation endpoints of pervasive applications. They gather data from the environment (e.g., temperature, motion, humidity) and perform actions (e.g., adjusting thermostats, lighting, and security systems) based on the applications' requirements. This enables pervasive applications to be context-aware and responsive to the physical world. EC serves as an intermediate processing layer for IoT data. By processing data closer to where they are generated, EC platforms can analyze and act on IoT data in real-time, supporting pervasive applications that require immediate response, such as autonomous vehicles or real-time analytics for industrial control systems. CC provides the back-end computational power needed for processing the vast amounts of data generated by IoT devices and aggregated through EC platforms. This can involve complex analytics, ML model training, and long-term data storage. CC environments can scale to handle the processing load, ensuring that pervasive applications have the necessary computational resources to function effectively. Together, these technologies enable pervasive applications to be both intelligent and responsive, adapting to changes in the environment and user needs with minimal latency. This integration is crucial for applications in smart cities, healthcare, industrial automation, and beyond, where real-time data processing and analysis are vital for decision-making and operational efficiency.

PC has already been adopted in many real-life applications, such as smart cities, agriculture, healthcare, marketing, etc. Finally, we once again extend the aforementioned Case 1, to depict how PC can be combined with IoT, EC and CC in order to improve the HVAC system in terms of intelligence, interactivity, and ubiquity. By extending the smart HVAC system with pervasive computing, the focus shifts to creating an intuitive, adaptive environment that not only optimizes operational efficiency but also prioritizes the comfort, health, and preferences of occupants. By incorporating pervasive computing, the HVAC system can recognize individual occupants and adjust environmental conditions according to their preferences and patterns. For example, if a particular user tends to get cold easily, the system could automatically increase the temperature in spaces they frequently occupy. Pervasive computing allows the HVAC system to interact with wearable devices, enabling it to monitor health indicators like body temperature or stress levels. This data can be used to make real-time adjustments to the environment, enhancing comfort and well-being.

1.5 Research Question

The rapid proliferation of IoT devices and the evolution of EC and CC have introduced significant challenges in managing and processing large volumes of distributed data efficiently. Traditional methods for task management, scheduling, and learning are not well-suited to the resource-constrained environments of EC. Specifically, the issues of data similarity extraction, efficient training, adaptive task management/scheduling, and effective TL in edge environments present critical challenges. These problems are compounded by the dynamic nature of data and resource availability in IoT ecosystems. This thesis aims to address these challenges by developing novel frameworks and algorithms that enhance the efficiency and effectiveness of data processing, task management, and learning in edge and CC environments. The research will focus on:

- Efficient similarity extraction methods for distributed datasets to improve data processing tasks.
- A data-aware incremental learning framework that provides a continuous update of ML models as new data arrives. This ensures that the models remain relevant and accurate over time without the need for complete retraining, thus, saving computational resources and improving response times.
- Effective TL techniques tailored for EC, optimizing when and how knowledge transfer occurs to improve model accuracy and reduce training times.
- Adaptive task management mechanisms that can dynamically respond to data and concept drift, ensuring optimal resource utilization and task performance.
- Advanced task scheduling algorithms that consider task-data correlation to minimize resource wastage and execution failures.

By tackling these issues, the thesis aims to contribute to the development of more robust and scalable EC frameworks, capable of supporting the growing demands of IoT and pervasive applications.

1.6 General Setting

We contemplate a group of N edge nodes $n_i, i = 1, \ldots, N$ with every member being able to directly communicate with the other ones. Each node processes and stores data coming from registered IoT devices, forming its own dataset $D_i, i = 1, 2, \ldots, N$. Each dataset D_i is consisted of multivariate vectors $x = [x_1, x_2, \ldots, x_d]$, $x \in \mathbb{R}^d$ where each x_i depicts a distinct feature (e.g., CO_2 measurements, occupancy, ventilation, etc.). Nodes derive insights, support predictions, or trigger actions, by processing their datasets with statistical or ML methodologies. However, these nodes operate under resource constraints and may hold only a partial view of the global system. The problem involves developing frameworks and algorithms to enhance the performance and efficacy of data processing, task management, and learning in EC and CC environments.

2 Dataset Similarity in Distributed Systems

2.1 Context and Motivation

Data are created and stored in several nodes which makes it difficult to retrieve valuable insights from local datasets alone in IoT and EC environments. The amount of data each node possesses is limited and often a non-representative subset of the global data. Hence, obtaining the similarity across datasets is very important for simultaneous processing across nodes. This capability is especially critical for the effectiveness of ML and data analytics, which often require broader context and diversity to produce reliable results. The problem is figuring out how to determine dataset similarity without sharing big amounts of data which is time and resource demanding. Most approaches are built around mostly static datasets and do not account the distributed and dynamic nature of data in edge systems [24–27]. This work intends to fill the void through the development of a synopsis-based similarity detection mechanism designed for dynamic, high volume datasets. These mechanism will enable distributed systems to work together and utilize resources more efficiently. Hence, enhancing decision making, and real-time analytics performance.

2.2 Related Work

The research community has given significant effort in developing distance functions that identify similarity among datasets. So as to assess the overall similarity various methods have been proposed.

2.2.1 Methods that utilize L1/L2 Norms

Some distance measures utilizing L1 and L2 norms are Edit Distance with Real Penalty (ERP), and Dynamic Time Warping (DTW). ERP [28], combines L1-norm with edit distance and allows local time shifts. DTW [29] achieves a correspondence between two time series by ‘warping’ the time axis. Further improvements by [30] on DTW add incremental warping windows for improved performance. Various methods use matching thresholds for similarity scores, such as the Edit Distance on Real Sequence (EDR) [31] or Longest Common Sub-sequences (LCSS) [32]. These methods are considered robust against noise, shifts in data, scaling errors, sensor drops, misdetection, and changes in rates of sampling [31], [32]. Re-sampling a time series to match the length of another time series is explored, and Euclidean distance is used as the similarity measure in [33]. The results indicate that similarity detection is not affected by this transformation in many experimental cases. Also, in [34] it is shown that uniform re-sampling sequences of different lengths is not detrimental to the accuracy and recall/precision metrics.

2.2.2 Methods that utilize Transformations of Time-Series Representations

Representations of time series data can be transformed and subsequently analyzed with common methods, such as Euclidean distance on Discrete Wavelet Transform (DWT) or Discrete Fourier Transform (DFT) coefficients [35]. As explained in [36], DFT transforms signals in the time domain to the frequency domain, allowing for the capture of global periodic frequencies. DWT, however, takes wavelets and discretely samples them, since it analyzes both frequency and time characteristics through space [36]. [37] analyzes DFT and DWT for the purpose of performing similarity searches in time-series databases. The analysis performed, shows that both approaches are practically achievable since the results are similar.

2.2.3 Methods that utilize Clustering

Time-series data clustering is an unsupervised approach of grouping time-series data based on similarity measures, which is a technique of exploratory data analysis that allows for a certain degree of freedom when analyzing [38]. In [39], a clustering approach that utilizes global structural features of time-series such as trend, seasonality, periodicity, or skewness is presented. To perform clustering, an optimal combination of features needs to be applied, and a feature selection process can be used. In a similar way, [40] proposes a two-stage approach where distance for each samples-pair is calculated first, and then samples are clustered based on distance.

2.2.4 Methods that utilize Data Summarization

Recent developments have paid great attention to the problem of sharing data synopses with the aim of decision making which is based on similarity of datasets. A synopsis data structure aims to summarize a dataset using succinct representations [41]. On the one hand, sharing datasets in their full form is rarely possible because of time and storage limitations. On the other hand, edge nodes need to gain deep insight into peer datasets for proper execution of queries and algorithms on appropriate data [42]. As IoT devices are capturing data continuously, the distributed datasets are expected to grow and possibly alter. Hence, synopses created out of such datasets will be required to be periodically updated. Recent progress in the area of data summarization has been described in [43] while some gaps are also identified on aspects to be researched in the future. Outlined in [44] are multiple synopsis approaches and their applications which deal with selectivity estimation, kmeans clustering and spatial partitioning. In [45], Synopses Data Engine (SDE) is presented, where streamline data summarization and parallel processing are integrated together in order to offer scalable analytics. Likewise, the Condor framework [46] permits streaming tasks based on synopses, abstracting away the internal execution details of the processes. In addition, [47] presents the SJoin algorithm that keeps a joint synopsis for dynamic databases that undergo continuous changes.

The Tree Summaries model of [48] is used to generate block summaries on multidimensional hierarchically organized datasets. A Tree Summary is an embedded weighted tree in the lattice that is generated using the cross-product of all hierarchies of the dimensions. The weight of the nearest ancestor in the tree is used to estimate each data element. The authors in [49] propose a model for supporting the main concepts of a wide numeric domain, aiming at integrating attributes which have an outcome effect attribute. The EDA4Sum system [50] is designed to interconnect different summaries at various stages of an Exploratory Data Analysis (EDA)process. The aim is to process EDA techniques so that the combined use of heterogeneous datasets is maximized, while permitting the user to specify the importance of uniformity, diversity, and novelty during the generation of summaries.

2.2.5 Trade-Off in Decision Making and Cost of Synopsis Exchange

The balance in the transfer of synopses and the quality of decision making represents a critical trade-off. In [51], the authors utilize the L1-norm to quantify differences between synopses pairs, use a clustering model to assess the temporal changes of these differences, and generate a final ranking of the most similar peers based on hosted data. Based on these ideas, [42] recommends a time-effective procedure for supervising the range of changes in synopsis updating. This scheme identifies when the synopses should be sent for maximum effectiveness with minimum communication cost. Also, [52] applies a probabilistic model for estimating the distance between two uncertain time-series. It allows to ensure accuracy/performance trade-off, where small sacrifices in accuracy result in large gains in scalability and processing speed.

2.2.6 Existing literature gap

Despite the remarkable advancement of similarity extraction techniques, contemporary approaches still rely heavily on static datasets or single-point-in-time analysis. This creates an obvious gap for working with updated sets of data, especially in the dispersed EC context with rapid and large volumes of data. Some approaches using distance functions based on L1/L2 norms, DTW, and some forms of clustering do not take into consideration the fact that data evolves over time. Recently developed models continuously depend on simplistic distance functions that disregard temporal change and inter-correlations within multi-dimensional data. Along with this, the problem of the trade-off between the computational resources needed for detection of similarities, and accuracy of the output has not been fully investigated, especially for the edge nodes that have resource constraints. There is a need for enhanced methods that integrate time series analysis and probabilistic models, in order to enhance similarity detection accuracy and efficiency, dynamically determining dataset similarity in the context of continuous data updates and real-time processing.

2.3 Problem Statement

Having established the broader problem space in 1.6, we now turn our attention to the critical issue of dataset similarity within distributed systems. A data synopsis $\mathcal{S}$ is a concise summary of a dataset, designed in such a way that captures its key characteristics, while significantly reducing its size. Without limiting the scope, each synopsis is depicted as an l-dimensional vector $s = [s_1, s_2, ..., s_l] \in \mathrm{R}^l$. The purpose of a data synopsis is to enable efficient analysis, query processing, or decision-making without requiring access to the full dataset. It is especially useful in scenarios where the full dataset is too large to be processed or stored effectively. Types of data synopses techniques include sampling, sketching, histograms, wavelets, clustering, dimensionality reduction, data cubes etc. [53].

Synopsis use case: Clustering refers to the procedure of grouping a set of similar data points into groups (clusters). The grouping is conducted in such a way that data points in the same cluster are more similar to each other than to those in other clusters. These clusters are summarized through their centroids and sizes. For the commonly used Iris dataset [1], a clustering synopsis could group flowers based on their petal length and petal width into clusters using a clustering algorithm (e.g. K-Means, number of clusters set to 3). Each cluster represents a group of flowers with similar petal dimensions, and the clustering could produce centroids such as (1.5 cm petal length, 0.2 cm petal width, 50 flowers), (4.5 cm petal length, 1.5 cm petal width, 50 flowers), and (5.5 cm petal length, 2.2 cm petal width, 50 flowers). In this case, the dataset's synopsis vector is $\mathbf{s} = (1.5, 0.2, 50, 4.5, 1.5, 50, 5.5, 2.2, 50)$.

Our goal is to identify similar datasets through their synopses vectors. Since these vectors are multidimensional, we investigate each dimension separately and summarize the findings in a final similarity score. Relying solely on the distance between two vectors is insufficient to effectively capture the semantics of the similarity between pairs of synopses. Since each synopsis is a lightweight representation of a dataset, the obtained results are approximate [41].

Definition 1. *A synopsis vector $\mathcal{S}$ is a lightweight representation of the dataset D_i hosted by the node n_i, extracted in a specific time interval.*

2.4 Similarity Extraction based on Synopses

This section addresses the problem of how to evaluate the similarity between two pairs of synopses, which is the enabling factor of decision making and collaborations. As it was explained in [51], the incoming data is gradually updating synopses vectors at local nodes. In this context, the sequences of synopses vectors are considered as time series with a specific window $T = [1, \ldots, w]$. We involve an interval strategy in which we select the most recent w observations of the synopses vectors that are selected for integration into the decision making process. By utilizing a sliding window technique, we focus on recent data to increase relevance of the result which is particularly important in collaborative processes. In this view, Sl is the local node's synopsis, while Sp is the peer node's synopsis, which is the one undergoing

[1]https://archive.ics.uci.edu/dataset/53/iris

similarity detection. Now, specifically, $Sl[j] = (Sl_1[j], Sl_2[j], \ldots, Sl_l[j])$ is the synopsis vector for the local node at time j ($j = 1, \ldots, w$) and $Sp[j] = (Sp_1[j], Sp_2[j], \ldots, Sp_l[j])$ is the corresponding synopsis vector for the peer node. This window-based approach enables us to observe the dynamics of Sl and Sp, rather than focusing solely on the most recent observation.

2.4.1 Synopses Correlation Analysis

The identification of correlation within the data is an essential step regarding the relations between data and their direction. Our methodology adopts the classic measure of correlation of two variables (datasets synopses' Sl and Sp within our setting), the Pearson Correlation Coefficient (PCC) [54]. PCC scores lie between -1 and 1, with zero signifying that Sl and Sp have no correlation, positive values mean that Sl is increasing with Sp and negative correlation means that Sl is decreasing with Sp. The phenomenon of both variables tightly coupling is termed as correlation, and values closer to 1 denote correlation being strong positive. Correlation between the two is evaluated for each point in time for Sl and Sp through different perspectives Sl_i and Sp_i.

$$CORR_i(Sl_i, Sp_i) = \frac{\sum_{j=1}^{w}(Sl_i[j]-MSl_i)\cdot(Sp_i[j]-MSp_i)}{\sqrt{\sum_{j=1}^{w}(Sl_i[j]-MSl_i)^2}\cdot\sqrt{\sum_{j=1}^{w}(Sp_i[j]-MSp_i)^2}} \tag{1}$$

with:

$$MSl_i = \frac{\sum_{j=1}^{w} Sl_i[j]}{w}, MSp_i = \frac{\sum_{j=1}^{w} Sp_i[j]}{w} \tag{2}$$

Individual elements from Sl's and Sp's correlation for $i = 1, 2, \ldots, l$, are compiled and represented as a vector to depict the correlation of Sl and Sp for a specific period $T = [1, ..., w]$. The outcome correlation is computed as:

$$CORR(Sl, Sp) = (CORR_1, CORR_2, \ldots, CORRR_l) \tag{3}$$

Definition 2. *The correlation for the i-th dimension between Sl_i and Sp_i is assumed to be strong-positive if $CORR_i(Sl_i, Sp_i) \geq \theta$. In this context, θ is a fixed parameter.*

In order to classify outcomes of correlation we have defined a two component decision function:

$$f(CORR_i(Sl_i, Sp_i)) = \begin{cases} 0, & \text{if } CORR_i(Sl_i, Sp_i) < \theta \\ 1, & \text{if } CORR_i(Sl_i, Sp_i) \geq \theta \end{cases} \tag{4}$$

This is then further aggregated to provide an overall correlation summary across all dimensions as follows:

$$CORR(Sl, Sp) = \frac{\sum_{i=1}^{l} f(CORR_i)}{l}, i = 1, \ldots, l \tag{5}$$

which is used to decide if there exists strong-positive correlation between Sl and Sp.

Definition 3. *If $CORR(Sl, Sp) \geq \hat{\theta}$, the synopses vectors Sl and Sp are considered to have strong-positive correlation. In this context, $\hat{\theta}$ is a fixed parameter.*

2.4.2 The Probabilistic Model

Two synopsis vectors can differ significantly from each other in one or more dimensions, even if they show a strong-positive correlation. Hence, apart from correlation, a distance-based similarity metric is also integral for assessing discrepancies between synopsis vectors and determining the similarity potential. The probabilistic model estimates the relevance of two synopsis vectors for future data, while correlation displays if the vectors are tracking each other's trends through the most recent observations. As data streams come continuously at high volumes, synopses vectors change swiftly. The precise measurement of distance between two synopsis vectors over a specified time interval can be difficult to obtain when dealing with high volumes of data, and constantly evaluating each dimension can be extremely costly. To solve this problem, we use approximate distance measurement that accurately addresses our problem while simultaneously increasing system response speed.

Each dimension of the synopsis vectors Sl_i and Sp_i, is modeled as a stochastic process, a time series over a given window w. According to [52], the i-th dimension of Sl over the time interval is considered a continuous random variable ($Sl_i = [Sl_i[1], Sl_i[2], \ldots, Sl_i[w]]$). These random variables are normally labeled as unknown continuous random variables. However, their mean and standard deviation can be derived from observed samples. In the same manner, so is Sp_i given by $Sp_i = [Sp_i[1], Sp_i[2], \ldots, Sp_i[w]]$. In order to determine a distance-based similarity, we begin with the two time series as uncertain random variables that are separated by an unquantified distance. The $L1$ norm serves as a starting metric along this uncertain distance.

Definition 4. *The distance measure, dist, between the two unknown random variables is defined as:*

$$dist(Sl_i, Sp_i) = |Sl_i - Sp_i| = \sum_{j=1}^{w} |(Sl_i[j] - Sp_i[j])| \tag{6}$$

We run into some challenges in determining a precise distance metric using the $L1$ norm. For this reason, we establish a probabilistic distance similarity measure query over the uncertain time series. First, it has to be established that distance-based similarity measures will not work if the two time series random variables, Sl_i and Sp_i, are weakly linked. Thus, we declare that they are strongly correlated random variables before using the probabilistic model. The distance $dist(Sl_i, Sp_i)$, is itself a random variable and thus has a mean and variance. These may be computed as follows:

$$\mathbb{E}(dist(Sl_i, Sp_i)) = \mathbb{E}(|Sl_i - Sp_i|) = |\mathbb{E}(Sl_i) - E(Sp_i)| \tag{7}$$

$$\begin{aligned} Var(dist(Sl_i, Sp_i)) = Var(Sl_i) + Var(Sp_i) \\ -2 \cdot Cov(Sl_i, Sp_i) \end{aligned} \tag{8}$$

where

$$Cov(Sl_i, Sp_i) = \sqrt{Var(Sl_i)} \cdot \sqrt{Var(Sl_i)} \cdot CORR_i(Sl_i, Sp_i) \tag{9}$$

and $CORR_i(Sl_i, Sp_i)$ accounts for the earlier established correlation coefficient.

To evaluate the gap between the two random variables, Chebyshev's inequality is employed, as it ensures that only a limited proportion of values fall outside a specified range, defined by k standard deviations from the mean. In the case of Chebyshev's inequality, it can easily be shown that for any real value k, only the case $(k \geq 1)$ is feasible, as other values lead to either negative or exceeding unity probability values. Hence while applying Chebyshev's inequality, we conclude:

$$\begin{aligned}P[|dist(Sl_i, Sp_i) - \mathbb{E}(dist(Sl_i, Sp_i))| \\ \geq k \cdot \sqrt{Var(dist(Sl_i, Sp_i))}] \\ \leq \frac{1}{k^2}\end{aligned} \tag{10}$$

This can be transformed to:

$$\begin{aligned}P[|dist(Sl_i, Sp_i) - \mathbb{E}(dist(Sl_i, Sp_i))| \\ \leq k \cdot \sqrt{Var(dist(Sl_i, Sp_i)}] = \\ 1 - P[|dist(Sl_i, Sp_i) - \mathbb{E}(dist(Sl_i, Sp_i))| \\ > k \cdot \sqrt{Var(dist(Sl_i, Sp_i)}] \\ \geq 1 - \frac{1}{k^2} = \frac{k^2 - 1}{k^2}\end{aligned} \tag{11}$$

We set $\mathbb{E}(dist(Sl_i, Sp_i)) = \mu$, $Var(dist(Sl_i, Sp_i) = \sigma$ thus:

$$\begin{aligned}|dist(Sl_i, Sp_i) - \mu| \leq k \cdot \sigma \Leftrightarrow \\ -k \cdot \sigma + \mu \leq dist(Sl_i, Sp_i) \\ \leq k \cdot \sigma + \mu\end{aligned} \tag{12}$$

Since we are interested in identifying an upper bound, we consider only the right hand part of the inequality.

If $k \cdot \sigma + \mu \leq \beta$ (β is a constant number) then $dist(Sl_i, Sp_i) \leq \beta$ with probability greater or equal to $\frac{k^2-1}{k^2}$.

Definition 5. *Let β be a distance bound, ρ is a probability threshold, $k > 1$ is a real number, and let μ, σ be the mean and standard deviation of the random variable $dist(Sl_i, Sp_i)$. The pair Sl_i and Sp_i are considered similar if the following conditions are satisfied:*

$$\beta \geq k \cdot \sigma + \mu, \text{ and } \frac{k^2 - 1}{k^2} \geq \rho. \tag{13}$$

In light of this definition, we formally propose a multivariate indicator function to summarize the similarity results from all of the dimensons:

$$f(k,\sigma,\mu) = \begin{cases} 0, & \text{if } k \cdot \sigma + \mu > \beta \text{ or } \frac{k^2-1}{k^2} < \rho, \\ 1, & \text{if } k \cdot \sigma + \mu \leq \beta \text{ and } \frac{k^2-1}{k^2} > \rho. \end{cases} \tag{14}$$

The average for all dimensions denotes the similarity indicator between Sl and Sp:

$$sp = \frac{\sum_{i=1}^{l} f(k, \mathbb{E}(dist(Sl_i, Sp_i)), \sqrt{Var(dist(Sl_i, Sp_i))})}{l}, \ i = 1, ..., l. \tag{15}$$

Definition 6. *If $sp \geq \hat{\beta}$ then Sl, Sp are distance-based similar, and unsimilar in any other case. $\hat{\beta}$ depicts a fixed parameter.*

Thus, two synopses vectors Sl and Sp are considered similar if both $CORR(Sl, Sp) \geq \hat{\theta}$ and $sp \geq \hat{\beta}$ conditions are true. This means that the distance between the vectors is less than or equal to β with high probability ρ, and the vectors are strongly positively correlated. Within the definition of an EC system, we describe a local node and search for peers that contain similar data, which can be investigated with synopses vectors Sl and Sp. The result is a subset of nodes that hold data that are considered appropriate for allocation of tasks, as well as migration of data.

2.4.3 Similarity Extraction Mechanism

The proposed Data Similarity Computation Mechanism (DSCM) is illustrated in Algorithm 1. The algorithm computes whether the local node (Sl) and the peer node (Sp) share similar data at a specific level. The algorithm receives these two vectors together with the correlation threshold θ, probability threshold ρ, and distance bounds $\beta, \hat{\beta}$ as inputs. As long as the number of strongly correlated dimensions is equal to or more than a threshold fraction $\hat{\theta}$, and their distance is lesser or equal to β, with a probability value larger or equal to ρ, the two are considered similar. The algorithm computes $CORR_i(Sl_i, Sp_i)$, for every dimension, and evaluates if it adheres to the condition in Definition 2. If this holds true, the expected distance and variance $\mathbb{E}(dist(Sl_i, Sp_i))$ and $Var(dist(Sl_i, Sp_i))$ are computed next. A variable k is defined to be 1.01 and increased continuously by 0.01 until the value 4 is reached. During the process, if the condition in Definition 5 is satisfied somewhere along the expected limit of four standard deviations, the counter for similarity increases by 1 and the procedure continues with the next dimension. If not, the algorithm moves to a new dimension without any further calculations. Finally, we compute $sp = counter/l$. If the result meets or exceeds $\hat{\beta}$, we conclude that the two nodes possess similar data.

Algorithm 1 DSCM: Algorithm description

```
**Input:** $Sl = Sl_1, ...Sl_l$, $Sp = Sp_1, ...Sp_l$, $\theta, \rho, \beta, \hat{\beta}$
**Output:** Nodes with datasets similar to the local node.
────────────────────────────
counter $\leftarrow$ 0
for i=1 to l (i = i +1) do
    Compute $CORR_i(Sl_i, Sp_i)$
    if $CORR_i(Sl_i, Sp_i) \geq \theta$ then
        Compute $\mathbb{E}(dist(Sl_i, Sp_i))$, $Var(dist(Sl_i, Sp_i))$
        for k=1.01 to 4 (k = k + 0.01) do
            if $k * \sigma + \mu \leq \beta$ and $\frac{k^2-1}{k^2} \geq \rho$ then
                counter $+= 1$
                break
            end if
        end for
    end if
end for
Compute $sp = counter/l$
if sp $\geq \hat{\beta}$ then
    $Sl$ is similar to $Sp$
end if
```

2.5 Experiments and Results

2.5.1 Setup and Performance Metrics

We compare the performance of DSCM against the DTW algorithm [29] by conducting various experimental scenarios on three performance metrics. We regard *Mean Squared Error* (MSE) [55] as the first metric we consider. It measures the quality of the forecasting model, and all models aim to achieve an MSE close to zero. If the MSE value approaches 0 the similar peer nodes are determined with high accuracy, whereas if the value approaches 1 the similar peer nodes are misidentified. A measure derived from precision and recall, F-score is the second metric we evaluate. An outcome of F-score close to 1 points out that the model is highly effective in retrieving similar peer nodes, while close to zero portrays a poor ability to identify similar peer nodes. The final evaluation metric we will assess is *Mean Absolute Percentage Error* (MAPE) [55] . It measures the effectiveness of the predicting technique in numeral proportions. A MAPE value of 0% implies a significant ability of retrieving similar peer nodes, while a value of 100% indicates a poor ability to identify similar peer nodes.

We perform the experiments on three datasets:

- **Dataset** D_1 [56]: This dataset contains cooler fan vibration accelerometer data with attached weights. Each vector has five dimensions ($l = 5$) and includes the weight configuration ID, fan RPM speed percentage, and x, y, and z axis accelerometer readings. Data was gathered at 20 ms intervals for 1 minute over a range of 17 rotation speeds (20% to 100% of maximum fan speed). There is a 5% increment after each RPM range.

- **Dataset** D_2 [57]: This dataset holds recordings of a gas multi-sensor device used in a Italian city. This device along with a certified analyzer's gas concentration references, record hourly averages. The data comprises fifteen-dimensional vectors ($l = 15$) which contain temperature, relative humidity, absolute humidity, and other parameters. The date and time dimensions are omitted from the experiments.

- **Dataset** D_3 [58]: is synthetic and implements real-data structures encountered in industry. Each vector has fourteen dimensions ($l = 14$) , with relative humidity, sensor resistances, and target pollutants (e.g. CO, NO_2) being some of the attributes.

To ensure the amount of data is manageable, each dataset is subdivided into five sub-datasets. The first sub-dataset contains the dataset of the local node while the remaining four reflect the datasets of peer nodes. Each sub-dataset is independently divided into six sub-divisions and hence we simulate 6 timestamps. For every dimension of the data vectors, synopses vectors are calculated which gives rise to two distinct scenarios. Mean and standard deviation synopses are calculated as the model assumes all data to be from the same source. Hence, similarity is assumed between the local node and peer nodes.

Parameters (θ, ρ, β) of the model are set by users for every specific task or dataset and differ for various cases. In this particular setting:

- The correlation threshold is set to $\theta = 0.7$.
- The values regarding the probability threshold are $\rho = 0.6$ and $\rho = 0.8$
- The bound of distance β, is a delicate parameter and can potentially change through the different dimensions of a single scenario. In particular, for each dimension, β is set to a value which is a fraction, $\frac{t}{100}, t = 1, ..., 100$ of the average of the dimension's mean values.

For comparison, β is also used as a threshold for the DTW algorithm so that a constant upper bound is achieved. The correlation/probability thresholds θ and ρ are not meaningful in DTW, thus they are ommited. When working with data streams, a common practice is to calculate simple statistical measures such as averages, means, and standard deviations. In many cases, sketches or histograms may prove useful in predicting frequency counts [59]. Nonetheless, this set of experiments is restricted to numerical values and attributes such as frequency counts and item identifiers [60] are irrelevant. Under these conditions, synopsis sampling is performed by using basic aggregates like mean and standard deviation.

2.5.2 Evaluation on Dataset D_1

Initially, we evaluate our approach using dataset D_1. Figures 2 and 3 illustrate the performance when synopses are computed using the mean and standard deviation, respectively. The probability threshold ρ is set to 0.6 and 0.8 in both cases.

- When the synopses are calculated using the mean (Figure 2), our model achieves MSE = 0 at $t = 17$ for $\rho = 0.6$ and at $t = 18$ for $\rho = 0.8$. Correspondingly, F-Score = 1 and MAPE = 0% are attained at the same time steps.
- Using standard deviation for synopses (Figure 3), the lowest MSE of 0 is achieved at $t = 65$ ($\rho = 0.6$) and $t = 69$ ($\rho = 0.8$). The same trend holds for F-Score and MAPE.

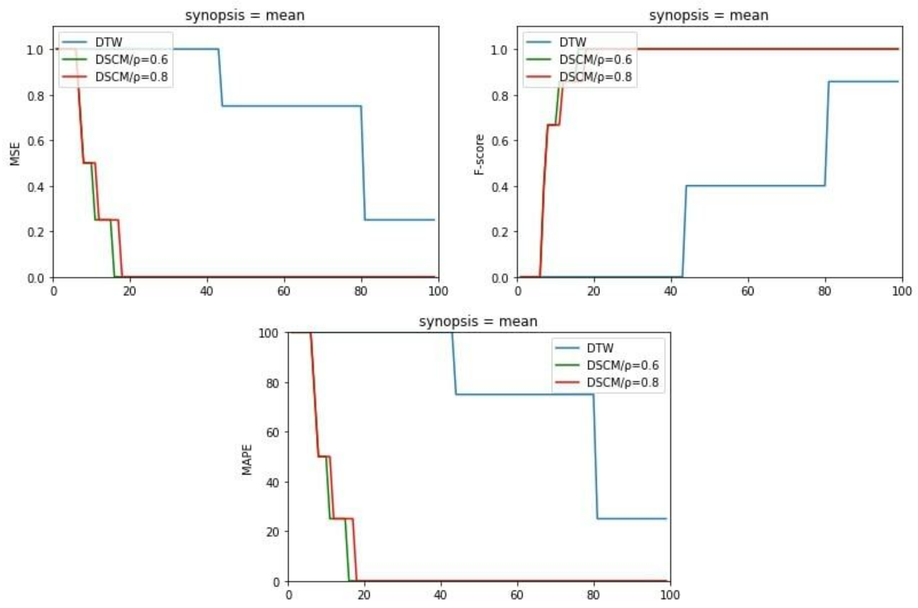


Figure 2: Performance metrics of MSE, F-Score and MAPE (Dataset D1 - Synopses = Mean).

- In comparison, DTW reaches a minimum MSE of 0.25, a lowest MAPE of 25%, and a peak F-Score of 0.85 at $t = 80$.

Our findings indicate that MSE, F-Score, and MAPE remain similar for both probability thresholds $\rho = 0.6, 0.8$, regardless of whether mean or standard deviation is used for synopses. However, utilizing mean enables our model to identify similar nodes more rapidly and precisely relative to the distance bound β.

2.5.3 Evaluation on Dataset D_2

We extend our evaluation to dataset D_2, with results depicted in Figures 4 and 5. The performance trends observed are consistent with those from D_1.

- With mean-based synopses (Figure 4), our model attains MSE = 0 at $t = 15$ for $\rho = 0.6$ and $t = 16$ for $\rho = 0.8$. Corresponding values of F-Score = 1 and MAPE = 0% are reached at the same time steps.

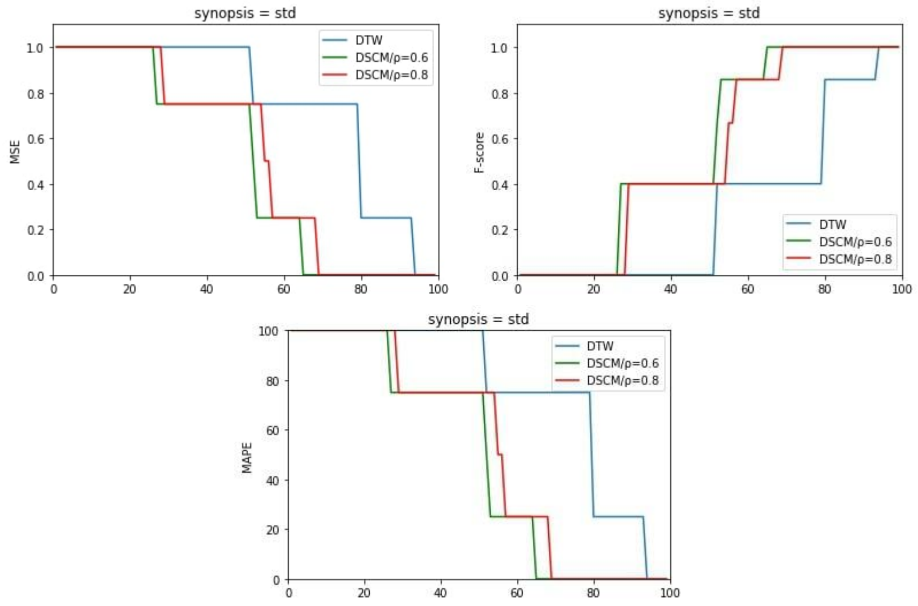


Figure 3: Performance metrics of MSE, F-Score and MAPE (Dataset D1 - Synopses = Standard Deviation).

- Using standard deviation-based synopses (Figure 5), MSE reaches 0 at $t = 44$ ($\rho = 0.6$) and $t = 46$ ($\rho = 0.8$), with F-Score and MAPE following the same pattern.
- DTW, in contrast, achieves a minimum MSE of 0.75, a lowest MAPE of 75%, and a peak F-Score of 0.4 at $t = 58$.

Again, results are comparable for both probability thresholds, but using the mean improves the speed and precision of identifying similar nodes relative to β. When relying on standard deviation, DTW fails to retrieve 75% (3 out of 4) of similar nodes.

2.5.4 Evaluation on Dataset D_3

Finally, we assess our approach using dataset D_3, applying the same methodology. Figures 6 and 7 summarize the outcomes.

- With mean-based synopses (Figure 6), MSE = 0 is attained at $t = 40$ for $\rho = 0.6$ and $t = 43$ for $\rho = 0.8$. F-Score and MAPE reach optimal values at these respective time steps.

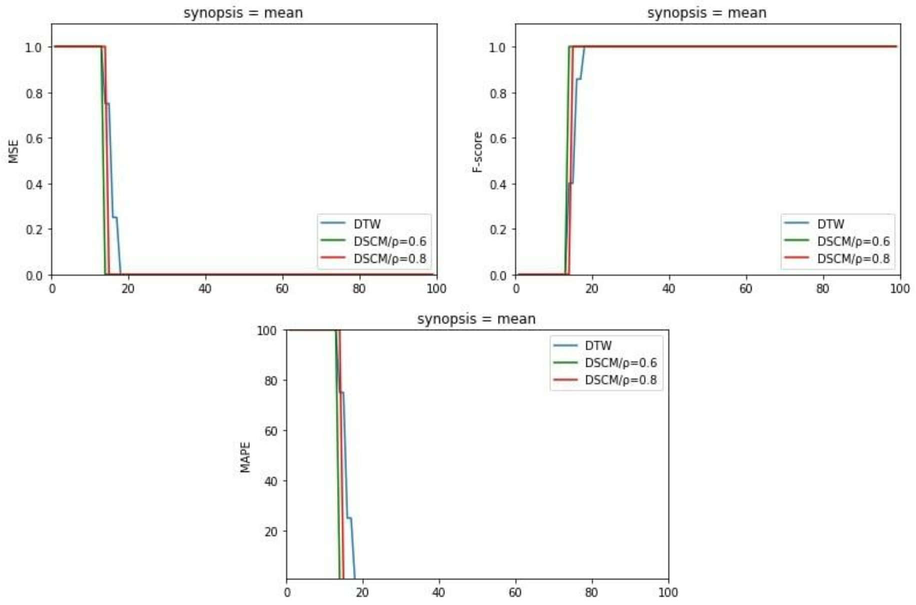


Figure 4: Performance metrics of MSE, F-Score and MAPE (Dataset D2 - Synopses = Mean).

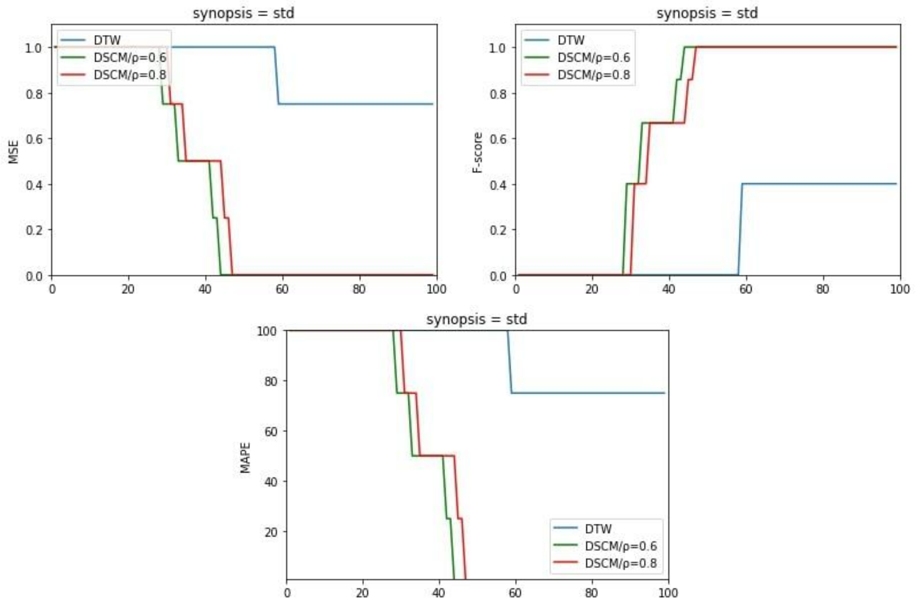


Figure 5: Performance metrics of MSE, F-Score and MAPE (Dataset D2 - Synopses = Standard Deviation).

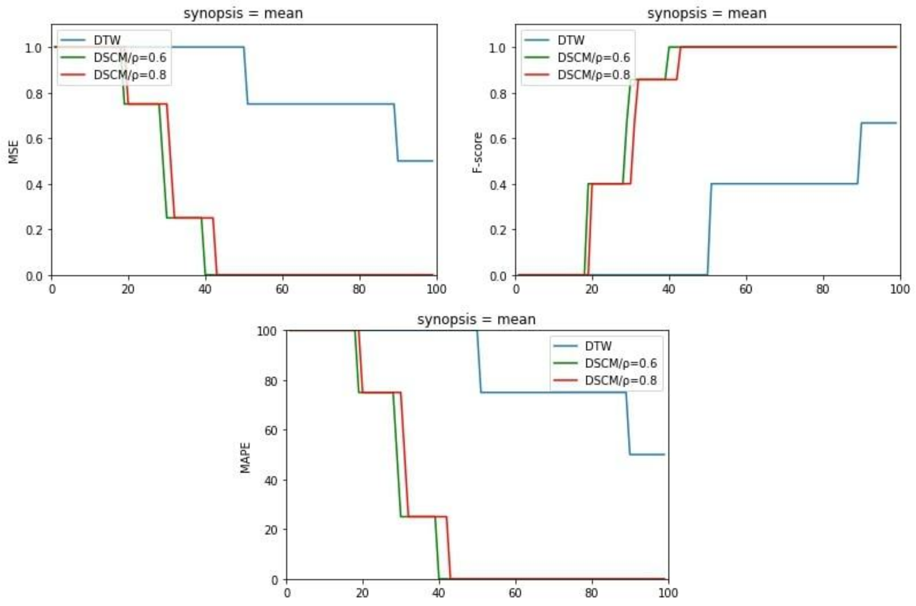


Figure 6: Performance metrics for MSE, F-Score, and MAPE (Dataset D3 - Synopses = Mean).

- When using standard deviation-based synopses (Figure 7), MSE = 0 is achieved at $t = 62$ ($\rho = 0.6$) and $t = 64$ ($\rho = 0.8$). F-Score and MAPE follow a similar trend.
- DTW attains a lowest MSE of 0.5, a minimum MAPE of 50%, and a peak F-Score of approximately 0.66 at $t = 90$.

Again, results for Dataset D_3 are consistent across both probability thresholds ($\rho = \{0.6, 0.8\}$), but using the mean significantly enhances both the speed and accuracy of identifying similar nodes with respect to β. When relying on standard deviation for synopses, DTW struggles to retrieve 50% (2 out of 4) of the similar nodes, further highlighting the advantages of our approach.

Across all three datasets, our model consistently outperforms DTW in terms of effectiveness and efficiency. Additionally, we observe slightly improved performance for $\rho = 0.6$ compared to $\rho = 0.8$. This is attributed to the fraction $\frac{k^2-1}{k^2}$ increasing as k grows, leading to faster threshold attainment for $\rho = 0.6$. Consequently, our model identifies similar nodes more quickly in these cases.

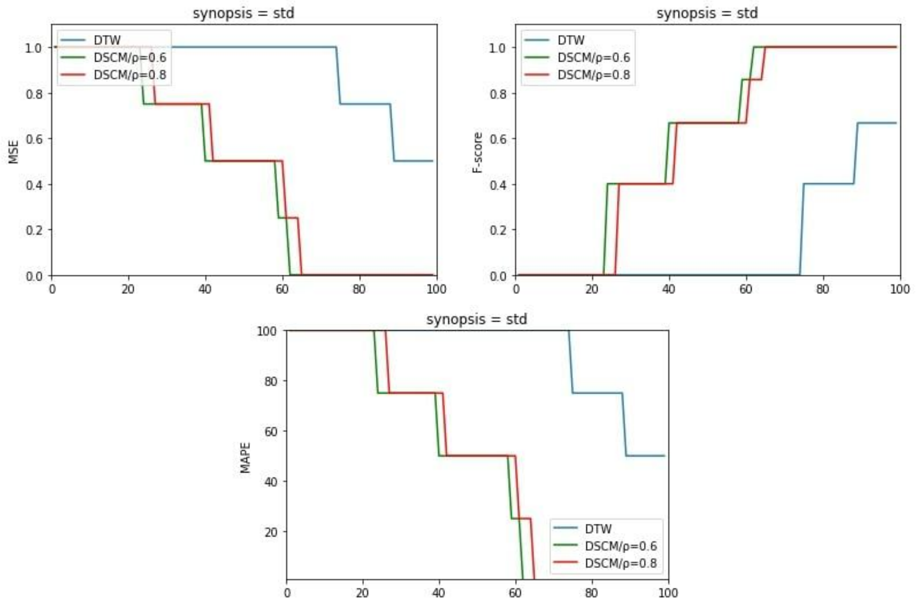


Figure 7: Performance metrics for MSE, F-Score, and MAPE (Dataset D3 - Synopses = Standard Deviation).

2.6 Summary and Key Findings

In the previous section, we evaluated the accuracy of the similarity extraction mechanism based on the provided synopses. The sliding window method facilitated seamless pre-fetching of summaries at the different distributed nodes in a timely manner. With this approach the model was able to establish the existence of correlations between the synopses and then, using the distance estimation model, find the corresponding nodes with similar datasets. The most important results are:

- **Enhanced accuracy of similarity detection**: With the correlation approach the robustness of the similarity detection algorithm and its performance was much higher than the compared DTW solution and, thus, allowing for more efficient task offloading based on dataset similarity.

- **Reduction in resource consumption**: Using the above methods, instead of full datasets, only a minimum statistical information was needed to be exchanged, thus greatly alleviating network congestion.

- **Quick response to changes**: The use of the sliding window approach allowed the system to track the evolving patterns of data outflow as they occur and thus simultaneuosly keep the similarity detection system relevant.

3 Accelerating Machine Learning/Deep Learning Training

3.1 Context and Motivation

Models for ML and DL are becoming increasingly significant for real-time analytics in IoT and EC systems. These models are expensive to train due to the extensive time and resources they consume. In EC settings, which have resource constraints, this presents a bottleneck which adversely affects the speed of intelligent application deployment. With the ever increasing need for real-time data pulled from edge systems, it is clear that more efficient training processes are needed while maintaining proper accuracy. Traditional training approaches are oftentimes less than ideal for these scenario. This work is primarily aimed at meeting the challenge of improving model quality while simultaneously reducing training times. It develops methods for better data selection and loss function minimization, thus enabling more effective ML in environments that lack sufficient resources.

3.2 Related Work

Accelerating ML and DL training is essential for real-time data processing in EC environments, where computational resources are often limited. There are several research efforts that focus on making the training process more efficient.

3.2.1 Broad Strategies for Enhancing Training Efficiency

This section outlines key strategies, which can be used in various neural networks (NNs) and ML models. These strategies seek to enhance the entire training paradigm, frequently implementing optimizations that are applicable to a broad range of settings. The objective here is to increase training velocity and efficacy at different levels. In [61], the authors provide a new architectural design to increase the training efficacy of NNs. Aiming at a faster convergence speed, Batch Normalization [62] and Dropout [63] are incorporated into an Independent Component layer, reducing the correlation coefficient and the mutual information among all the possible neuron pairs. The Selective-Backprop model presented in [64] further improves performance by restricting the focus of the backpropagation process to examples that the model had higher loss values during the last training iteration. The approach is to establish whether an example's gradient is computed and parameters changed, or just the next example is taken.

In [65], the authors suggest the deployment of the greedy DropSample technique, which improves the training of CNNs when used as image classifiers. It works by concentrating on those samples which after the forward pass are found to yield a big gradient and discarding those which do not affect the minimization loss value. Instead of computing the entire gradient, model parameters are updated with a limited number of entries utilizing the sparse backpropagation method in [66]. The method of forward propagation remains the same, however, the gradient vectors that are set to be modified, are sparsified by preserving only the top k components with the highest absolute values. Consequently, just a subset of k rows or columns in the weight matrix are updated, effectively reducing the computational overhead. The multi-sample dropout technique [63] aims to reduce training time by generating multiple dropout samples within a single forward pass. The average loss is computed by getting the individual loss from each of the dropout samples, which increases the efficiency of the original dropout method. An instance shrinking operation is applied in the AutoAssist framework in [67] in order to speed up the training time. This operation discards instances that do not improve the outcome and focuses on the informative ones, hence aids in computationally expensive processes.

3.2.2 Specific Techniques for Speeding Up Training

These types of models focus on specific forms of NNs or specific tasks in the training process. They allow for strategic optimizations to be implemented at the bottlenecks of the unique model features. In [68], the authors replace traditional random dropout schemes with systematically defined patterns through the Approximate Random Dropout. This approach avoids needless computations and access to data. A Search Algorithm using Stochastic Gradient Descent (SGD) is also developed to mitigate possible deficits in accuracy by creating optimized dropout distributions. As discussed earlier, batch normalization in [62] alleviates internal covariance shift, which in turn speeds up DNN training. The activations are normalized, and some methods used for optimization during training are ensured to work thereafter. The $L1$ norm batch normalization technique presented in [69] is a more sophisticated form of batch normalization which only involves linear operations in the process of forward and backward propagation. This method avoids network overhead associated with nonlinear square and square root operations that are part of $L2$ norm batch normalization. A deep extreme learning image classifier has been proposed based on DL in [70]. The model implements a dropout technique which randomly disables neurons during the training to decrease the computational time.

3.2.3 Handling the Training Samples

Whereas the former techniques are concentrated towards the internal aspects of the training algorithm and model structure, other techniques are focusing towards the control of training samples. The approach in [71] introduces an importance-based sampling strategy aimed at selecting the most informative samples. With this scheme, the stochastic gradients are less variant in the training phase. The scheme also provides an upper bound on the gradient norms and a variance reduced estimator. In [72], a DNN is trained with a new acquisition method that is based on weighted sampling. The new data assets are iteratively chosen and the DNN model is trained and then updated. A new weighting factor, derived from the sample's probability density, is introduced to achieve uniform selection in the sampling space.

3.2.4 Existing literature gap

Although several approaches have aimed at improving the training efficiency of ML/DL algorithms, little is done towards the management and dynamic adaptation of the training datasets. Most work concentrates on frameworks that optimize internal levels of the training algorithms and model architectures, such as batch normalization and dropout based methods. Current methods do not adequately address the distributional divergence of data assets throughout the training process, which can lead to sub-optimal model performance and prolonged training times. Importance sampling and active learning techniques that supplant less 'informative' examples with more 'informative' ones do not suffice as they do not focus on monitoring the statistical properties of training samples gradually to mark low-contribution data for exclusion. To effectively train a model in an restricted setting, systems that offer means for continuous monitoring of training data and construction adaptation are critical.

3.3 Problem Statement

Following what we defined in Section 1.6, let us imagine that the aforementioned nodes are responsible for executing multiple instances of processing tasks created by users or applications. These nodes have defined computational resources, which allow them to train and infer ML models on their datasets. Notably, these capabilities influence both training and execution performance, especially in the aspect of time. The range of ML models employed is broad, from regression to NN models, and even various types of classifiers dedicated to aid the decision-making process. The envisioned approach is depicted in Figure 8, showing the conceptual logic behind the approach and the general description of the runtime environment. In addition to data manipulation, nodes receive requests for task execution that typically require training instances of the ML models. These models, as well as the tasks that integrate them, are designed to create knowledge in an autonomous fashion, allowing nodes to cooperate and share information within the ecosystem or the cloud infrastructure.

The training of ML/DL models through a node's dataset frequently requires a lot of processing power and memory resources. Additionally, it should be emphasized that nodes need to carry out multiple processing tasks at the same time. These various responsibilities can only be achieved with effective management, which can also lead to improved overall performance. In this section, tuples are used to represent the training samples. These are fed to a processing module that attempts to create the final ML model. Nodes exist in a setting that is constantly changing. Due to this, they are subjected to what can only be described as infinite streams of data and tasks. While there is an effort to mix new data from the IoT infrastructure, continuously training is going to be extremely resource heavy and nearly impossible. For that reason, the system simply accepts what is currently available to it. By using a dataset frozen in time, the system effectively eliminates the overwhelming task of gathering new resources. This allows for the introduction of models like the proposed one, which is built upon the assumption given above and deals with an 'isolated' view of the training process each time. The training process is represented with the symbol $\mathcal{T}$. Additionally, we divide the local dataset into two parts, $D_{\mathcal{T}}$ and $D_{\overline{\mathcal{T}}}$, where the first is the subset of the processed data and the second is the subset of the unprocessed ones. As $\mathcal{T}$ progresses, data are transferred from $D_{\overline{\mathcal{T}}}$ into $D_{\mathcal{T}}$ and as a result, the statistical properties of both subsets potentially change.

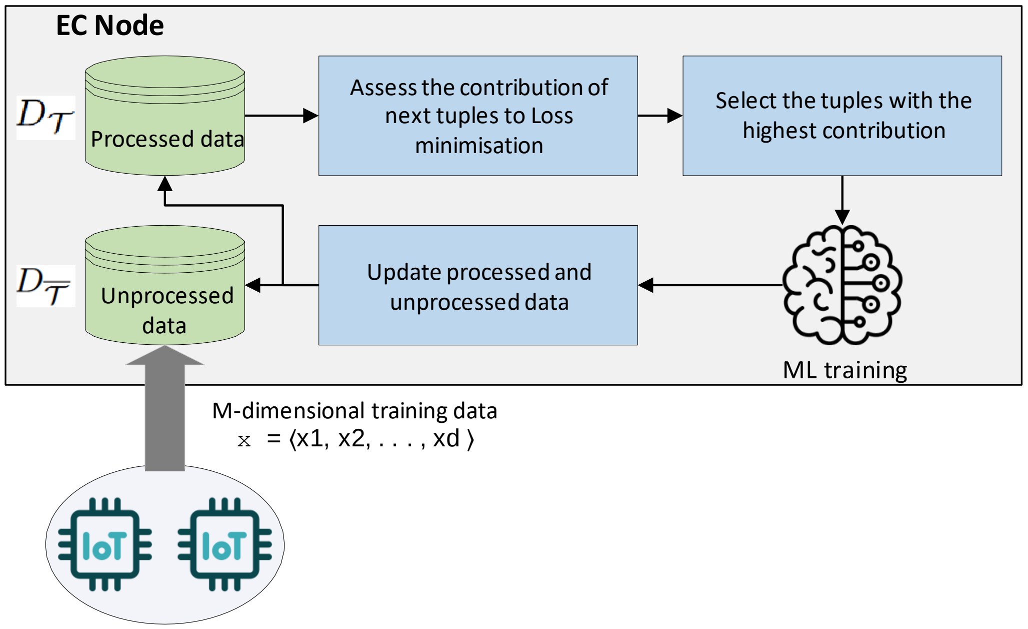

EC Node
$D_{\mathcal{T}}$
Processed data
Assess the contribution of next tuples to Loss minimisation
Select the tuples with the highest contribution
$D_{\overline{\mathcal{T}}}$
Unprocessed data
Update processed and unprocessed data
ML training
M-dimensional training data
x = ⟨x1, x2, . . . , xd ⟩
IoT
IoT

Figure 8: Overview of the integrated solution presented which is deployed on an EC node.

Nodes have a copy of the tasks queue where tasks that arrive are kept enabling them to manage $\mathcal{T}$ and complete the training of the ML model. The nodes also have copies of the subsets $D_{\mathcal{T}}$ and $D_{\overline{\mathcal{T}}}$. There is a maximum threshold size on this queue that affects the workload on the node n_i denoted by l_i. Once this threshold is reached, the node is forced in to an overusing state failing to deliver real-time tasks which increases the response times. The two basic approaches for carrying out the training process $\mathcal{T}$ are: **(a)** Part of the tasks a node has to work on is paused until the defined work cycle is completed, or **(b)** Tasks and $\mathcal{T}$ are performed concurrently, however this has the drawback of having to manage the work load more efficiently. Approach (a) is used in this paper, resulting in less time inactively waiting for the ML models to be updated for tasks that are queued for processing. The second approach, is not being investigated in this effort and would be the subject of future research.

We elaborate on how a single node n_i would accomplish the task of training the model. n_i calls $\mathcal{T}$ at specific time intervals and gains knowledge over the ML model via training that starts with a local dataset snapshot.

While $\mathcal{T}$ is running, n_i may receive new tasks which results in the increase of the load l_i. Gradually, when $\mathcal{T}$ is proceeding, the node n_i takes in the data available at hand and modifies within the ML model according to the prerequisites of $\mathcal{T}$ and statistical requirements of the particular model. For instance, the ready data for training in the NN training is represented by $D_{\mathcal{T}}$ while the left over data meant for feeding into the network is represented by $D_{\overline{\mathcal{T}}}$. In the context of n_i, the training loss of the model, denoted as $\mathcal{L}$, serves as a performance metric which node n_i monitors when $\mathcal{T}$ is in operation and can indicate whether $\mathcal{T}$ is truly effective. If the statistical properties of $D_{\overline{\mathcal{T}}}$ indicate that further training will yield only marginal improvements in $\mathcal{L}$, the node may opt to terminate $\mathcal{T}$ early. For instance, given that when $\mathcal{T}$ is to find coefficients of linear regression, fresh data indeed would change the slope and intercept, the node needs to carry more training. However, when the new data is flown with similar features, $\mathcal{T}$ can be stopped and set with an acceptable error upper bound.

Such an approach is plausible in circumstances when tolerance for small errors exists and does not result in a chain of defects in the service provision. In cases where accuracy is of utmost importance, all data points should be passed through the nodes prior to the determination of $\mathcal{T}$. This balance is determined by the extent of training time spent versus the amount of the acceptable error. The proposed method is geared towards determining the conditions under which it is possible to obtain the acceptable level of error without processing the whole dataset. This method is advantageous for the nodes with limited resources because it allows for better allocation of resources and multitasking.

3.4 Adaptive ML Training based on Data Context

Contextual awareness is crucial in optimizing ML/DL training processes, particularly in dynamic and distributed computing environments. This section addresses the incorporation of contextual features like data specificities within the ML/DL training procedure. Due to the incorporation of contextual knowledge, the training process can be adaptive because models can now react to real changes in data distribution and resource allocations. This method achieves higher training efficiency because it concentrates computational power on the most relevant data subset. The methods presented here reveal how context can be utilized to develop ML/DL models that are robust and resilient even in complex and real world situations.

3.4.1 Optimization of the Loss Function

The loss function $\mathcal{L}$ serves the purpose of measuring the efficacy of the training process $\mathcal{T}$. An ML model is undergoing training and $\mathcal{L}$ captures how well the predicted outcome $\hat{y}$ approaches the *target* value y. In this approach, it is assumed that the ML model is trained with under a supervised learning paradigm, meaning there exists a training dataset with the target value y for each input sample. Minimizing the difference between the expected value and the predicted value, $\mathcal{L} = |\hat{y} - y|$ is the primary goal during training, with the ideal value being zero. In some cases $\mathcal{L} \to 0$ is an overfitting indication and something that will not be addressed in this effort. In particular, $\mathcal{L}$ can be estimated as a transformed loss computed when the system is trained with a dataset generated by x: $\mathcal{L}(x, \hat{y}, y)$. Transformations of the loss function $\mathcal{L}$ include mean square error, cross entropy, hinge and quantile among others. These do contradict the basic principle that there is an increased value of loss function when the prediction differs grossly from the ground truth.

During training, each input x generates an output $\hat{y}$. Then, applying the training process (e.g. backpropagation for NNs) the model weights or coefficients will be slightly altered at every step in order to minimize $\mathcal{L}(x, \hat{y}, y)$. Effective model training includes having a $D_{\mathcal{T}}$ which is wide enough to capture the heterogeneity needed to alter the weights of the model properly. If the training subset, $D_{\mathcal{T}}$, has a low variance, (σ_D), the inputs x will come from a narrow distribution which will limit the contributions to the steps taken to fit the model with respect to the underlying data distribution. Models trained on such datasets may fail to generalize, especially when the test data distribution is sufficiently different compared to $D_{\mathcal{T}}$ and whenever the data is coming from a combination of probability distributions. In order to prevent this, the proposed methodology detects training samples based on whether they stem from a different distribution when put to T and attempts to classify them through indirect means.

By integrating the aforementioned observations, we infer that given a model pre-trained on the dataset $D_{\mathcal{T}}$, any input sample x will have an insignificant impact on minimizing $L(x, \hat{y}, y)$ if $x \sim f_X(D_{\mathcal{T}})$. Under this interpretation, the steepness of the slope of $L(x, \hat{y}, y)$ is associated with the input x. If x originates from a distribution identical to that of dataset $D_{\mathcal{T}}$, the expected variation in both ϵ and $L(x, \hat{y}, y)$ remains minimal (i.e., the loss function exhibits a non-steep slope). Consequently, the evolution of the slope of $L_t(x, \hat{y}, y)$ across consecutive training rounds $(1, 2, \ldots, t, \ldots)$ will be influenced by the distribution of $D_{\mathcal{T}}$ and the characteristics of the input sample x. Assuming that the training process T starts at $t = 1$ (where we consider discrete time instances for convenience) with the first training sample, the objective is to monitor the statistical properties of $D_{\mathcal{T}}$ along with the loss function $L_t(x, \hat{y}, y)$ to identify the most impactful training samples for T and determine whether stopping criteria exist to save computational time. In general, at time t, a sample x exits $D_{\overline{\mathcal{T}}}$ (i.e., $D_{\overline{\mathcal{T}}} = D_{\overline{\mathcal{T}}} - x$) and enters $D_{\mathcal{T}}$, (i.e $D_{\mathcal{T}} = D_{\mathcal{T}} \cup x$), producing a new loss value $L_t(x, \hat{y}, y)$. We assume that $D_{\mathcal{T}}$ and $D_{\overline{\mathcal{T}}}$ follow two different distributions governed by random variables A and B with means μ_A, μ_B and standard deviations σ_A, σ_B, respectively. The statistical properties of both datasets can be utilized to estimate the probability that a given training sample x belongs to either $D_{\mathcal{T}}$ or $D_{\overline{\mathcal{T}}}$.

The fundamental idea behind our approach can be summarized as follows: At each time step t, retrieve the next training sample x from $D_{\mathcal{T}}$. If x is significantly related to $D_{\mathcal{T}}$, discard it and proceed to the next available sample. This approach effectively reduces the number of training samples used, consequently decreasing the duration of each training epoch. In the next subsection, we provide a detailed explanation of this implementation and outline the methodology used to assess whether a training sample x is relevant to $D_{\mathcal{T}}$.

Use case: NN Training procedure. The change of weights in NNs during training is given by the equation:

$$W_{t+1}^i := \gamma^i W_t^i - \eta_t^i \left(\frac{\partial \mathcal{L}(W_t^i, x)}{\partial W} \right)^\nu, \quad x \in D_{\mathcal{T}}, \tag{16}$$

where W_{t+1}^i is an element of the weight matrix of the i-th node after the (t+1)-th training round, γ^i is the momentum, η_t^i is the learning rate, and $\mathcal{L}(W_t^i, x)$ is the loss for example x in the t-th round. After n rounds of training, the weights are:

$$W_n^i := \sum_{t=1}^{n-1} \gamma^i W_t^i - \eta_t^i \left(\frac{\partial \mathcal{L}(W_t^i, x)}{\partial W} \right)^\nu, \quad x \in D_{\mathcal{T}}. \tag{17}$$

Several studies, e.g., [73] suggest that altering η between consecutive training rounds can have a positive impact on model generalization and accuracy [73]. In many cases the training procedure is initiated with a high η and gradually its value is being decreased. This ensures that each term does not contribute equally to the computation of W_n^i. Given this, a critical question emerges regarding the treatment of terms with low numerical values that have minimal impact on W_n^i. To address this issue, we leverage the proposed model, which is thoroughly described in the subsequent sections.

3.4.2 Adaptive Training Samples Management in a Data-Sensitive Environment

As previously discussed, we assume that the two subsets, $D_{\mathcal{T}}$ and $D_{\overline{\mathcal{T}}}$, may follow distinct distributions, with the respective random variables denoted as A and B. As samples transition from the unseen dataset to the seen dataset after being utilized in $\mathcal{T}$, their distributions evolve accordingly. Following an initial warm-up phase where all training samples are processed by $\mathcal{T}$, we can estimate the probability of a training sample x from $D_{\overline{\mathcal{T}}}$ belonging to the underlying distribution of $D_{\mathcal{T}}$. If this probability is sufficiently high, then including x into $\mathcal{T}$ could be deemed unnecessary, with only a minimal impact on the final ML model's loss.

To balance this trade-off effectively, we introduce two thresholds: θ_w and θ_s, where threshold θ_w represents the upper bound beyond which the expected loss becomes unacceptable and θ_s defines the lower limit below which the probability of alignment between the next training sample x and $D_{\mathcal{T}}$ is considered negligible. In more complex scenarios where the random variable A follows a mixture of distributions (potentially distinct ones), we must fit $D_{\mathcal{T}}$ to this unknown mixture. To mitigate computational overhead, we propose estimating the mixture parameters at predefined intervals. The probability that x belongs to this mixture is given by:

$$p_x = f\left(x\right) = \sum_{k=1}^{K} w_k f_k\left(x; \Theta_k\right) \tag{18}$$

where w_k represents the weight of the k-th distribution (k total distributions, such that $\sum w_k = 1$), and Θ_k denotes its parameter set. Before computing p_x, our objective is to estimate Θ_k using the samples within $D_{\mathcal{T}}$ through the well-established Maximum Likelihood Estimation (MLE) method [74]. This method seeks to determine $\hat{\Theta}$ that maximizes the likelihood function:

$$\hat{\Theta} = \arg\max_{\Theta} f(\left\{D_{\mathcal{T}}\right\}; \Theta) \tag{19}$$

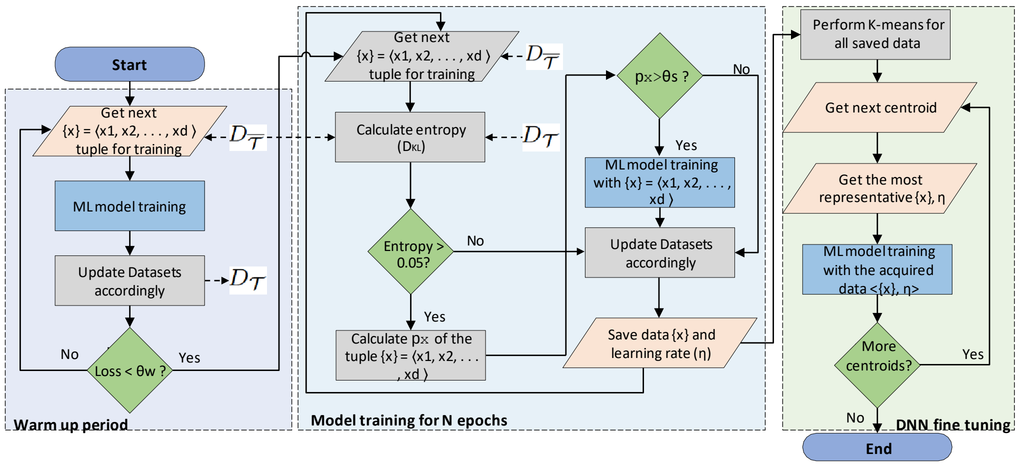

Start
Get next {x} = ⟨x1, x2, . . . , xd ⟩ tuple for training
ML model training
Update Datasets accordingly
Loss < θw ?
No
Yes
Warm up period
Get next {x} = ⟨x1, x2, . . . , xd ⟩ tuple for training
Calculate entropy (DKL)
Entropy > 0.05?
No
Yes
Calculate px of the tuple {x} = ⟨x1, x2, . . . , xd ⟩
px>θs ?
No
Yes
ML model training with {x} = ⟨x1, x2, . . . , xd ⟩
Update Datasets accordingly
Save data {x} and learning rate (η)
Model training for N epochs
Perform K-means for all saved data
Get next centroid
Get the most representative {x}, η
ML model training with the acquired data <{x}, η>
More centroids?
Yes
No
DNN fine tuning
End

Figure 9: The algorithmic perspective of the data-driven ML training process

For computational convenience, we employ the log-likelihood function instead of the standard likelihood formulation. Under the assumption of independent and identically distributed (iid) variables, this simplifies to:

$$\hat{\Theta} = \arg\max_{\Theta} \prod_{i=1}^{|D_{\mathcal{T}}|} f(\{D_{\mathcal{T}}\}_i\,; \Theta) \tag{20}$$

The second component of our model focuses on determining when to halt $\mathcal{T}$ if the distributions of the two subsets exhibit similarity. To incorporate contextual awareness into the ML training process, we explore an additional insight into the distributions of $D_{\mathcal{T}}$ and $D_{\overline{\mathcal{T}}}$, their divergence. Divergence serves as a statistical measure to quantify the difference between two distributions. In this work, we utilize the Kullback-Leibler (KL) divergence, defined as:

$$D_{KL}(A||B) = \sum_{x \in X} A(x) \ln \frac{A(x)}{B(x)} \tag{21}$$

KL divergence employs the samples from distributions A and B to compute their relative entropy, effectively measuring the information loss when replacing A with B. A low $D_{KL}(A||B)$ value indicates a strong similarity between A and B, whereas higher values signify greater divergence.

3.4.3 Reducing Training Time with a Novel Method

On an algorithmic level, our approach is applicable to any ML technique that processes a dataset and trains a model over a predefined number of epochs. In this subsection, we outline a structured sequence of steps that utilize the parameters examined throughout this study, specifically θ_w, θ_s, p_x, $D_{\mathcal{T}}$, $D_{\overline{\mathcal{T}}}$, and $\{x\}$. It should be noted that θ_w and θ_s are system hyper-parameters, and hence they are set by the user. Hyper-parameter θ_w is a warm-up phase loss function threshold and higher values of it tend to shorten warm-up periods, which increases model loss and speeds up training. At the same time, θ_s functions as a threshold for accepted tuple rejections rates in the data-aware ML methodology. When higher values of θ_s are set, more tuples are rejected and the training time is decreased. Thus, these hyper-parameters are either set statically, according to the available computational resources of each node, or dynamically, according to its workload. Their aim is mainly to tune the training deformulation in regards to time efficiency versus model quality.

Algorithm 2 and Figure 9 provide a detailed illustration of our methodology, where the ML model undergoes training with the entire dataset during the warm-up phase. Once the loss value falls below the predefined threshold θ_w, the data-aware methodology is activated. At this stage, the sets $D_{\mathcal{T}}$ and $D_{\overline{\mathcal{T}}}$ are initialized to represent the observed and unobserved training data, respectively. During each epoch, the probability p_x is evaluated against the threshold θ_s. If p_x exceeds θ_s, it is inferred that $\{x\}$ is closely related to the training set $D_{\mathcal{T}}$, and the tuple is therefore excluded from training. The datasets $D_{\mathcal{T}}$ and $D_{\overline{\mathcal{T}}}$ are subsequently updated. Conversely, if $p_x \leq \theta_s$, the tuple $\{x\}$ is included in the training process. This procedure is repeated for each epoch as long as there are elements in $D_{\overline{\mathcal{T}}}$ (i.e., $D_{\overline{\mathcal{T}}} \neq \emptyset$) and the KL-divergence between $D_{\overline{\mathcal{T}}}$ and $D_{\mathcal{T}}$ remains above 0.05. The KL-divergence quantifies the entropy between two distributions, with higher values indicating a lower degree of shared information. In this work, we set a threshold of 0.05, above which the distributions of $D_{\overline{\mathcal{T}}}$ and $D_{\mathcal{T}}$ are regarded significantly different. When $D_{KL}(D_{\overline{\mathcal{T}}}||D_{\mathcal{T}}) > 0.05$, the datasets are deemed sufficiently distinct, and training continues. However, if $D_{\overline{\mathcal{T}}}$ and $D_{\mathcal{T}}$ become nearly identical ($D_{KL}(D_{\overline{\mathcal{T}}}||D_{\mathcal{T}}) \leq 0.05$), further training with $D_{\overline{\mathcal{T}}}$ is unlikely to improve model quality, prompting the process to skip to the next epoch.

Use case: Reduced Data-driven training for NNs. Regarding NNs, specialized ML training in a data-aware fashion has a missing dimension regarding their learning rate (η). This is due to some contemporary NN

Algorithm 2 Data-driven ML training reduction mechanism

```
procedure DATA-AWARE ML TRAINING($\theta_w, \theta_s, \{x\}$)
    while $\mathcal{L}(D_{\overline{\mathcal{T}}}) > \theta_w$ do
        Perform training with $\{x\}$
    end while
    for all training epochs do
        $D_{\mathcal{T}} = \emptyset$
        $D_{\overline{\mathcal{T}}} =$ Train set
        Calculate $D_{KL}(D_{\overline{\mathcal{T}}}||D_{\mathcal{T}})$
        while $D_{\overline{\mathcal{T}}} \neq \emptyset$ and $D_{KL}(D_{\overline{\mathcal{T}}}||D_{\mathcal{T}}) > 0.05$ do
            Calculate $p_x$
            if $p_x \geq \theta_s$ then
                Perform training with $\{x\}$
                $D_{\mathcal{T}} = D_{\mathcal{T}} \cup x$
            end if
            $D_{\overline{\mathcal{T}}} = D_{\overline{\mathcal{T}}} - \{x\}$
        end while
    end for
end procedure
```

designs applying epoch-dependent learning rates. Therefore, learning rates must be taken into consideration when accepting or rejecting tuples. For instance, in the early stages of the epochs, learning rates are quite elevated compared to later epochs which means that rejecting these tuples will have a greater impact to the model's performance.

Algorithm 3 describes the modified data-aware training method for NNs. This change is also illustrated in Figure 9 located in the 'DNN Fine Tuning' section detailing the changes needed for training Deep Neural Networks (DNN). The methodology remains the same as in Algorithm 2 except for one aspect: in Algorithm 3, rejected tuples are kept in the D array and rates corresponding to them are kept in the E array for each training epoch. When the data-aware method is complete, K-means clustering is performed on D and E in order to assign centroids that most accurately represent the values of these two vectors. Each centroid is assigned to a set of tuples x along with a learning rate η. Afterwards, the NN model is tuned with these centroids by subseting the rejected tuples and setting different learning rates for epochs. The number of centroids is a critical factor influencing the model's quality. Increasing centroids allows for better generalization of the model but increases computation costs. Decreasing the number of centroids increases loss values but decreases training time. However, how to determine the optimal amount of K-means centroids to increase this methodology effectiveness is not within the limits of this paper.

3.5 Experiments and Results

3.5.1 Experimental setup and performance metrics

This section examines how the proposed data-aware method affects different ML models. We aim to assess the impact on training time and overall model performance. Since this technique discards certain samples of the dataset while training is concluded, it is anticipated to improve the execution time, but may also degrade model performance. In order to study the above, we prepared three different experiments in different ML models and datasets, as described below:

- **Clustering:** We apply a DNN for clustering the IRIS dataset [75]. The dataset consists of 150 images, which are labeled into three groups, each one corresponding to a different species of the Iris plant. A supervised clustering approach is implemented in which a labeled data set is available for the creation of clusters for which class based probability density functions are maximized.

- **Support Vector Machine (SVM):** In this case, an SVM classifier is trained with the MNIST dataset [76]. MNIST is a well-known dataset consisting of 60,000 images containing hand-written digits in grayscale and divided into 10 classes.

- **DNN:** In this setting we deploy four popular DL architectures: VGG-19 [77], MobilenetV2 [78], ResNet101 [79], and DenseNet121 [80] in the CIFAR-10 [81] and CIFAR-100 [82] datasets. The first one, consists of 60,000 images divided into ten classes, and the second, consists of 60,000 images divided into 100 classes.

In each of the experimental settings, we set the parameters θ_w and θ_s, analyzing the outcomes against a baseline model termed *Base* (as it does not utilize the data-aware method). Also, we implemented another control

Algorithm 3 Data-driven ML training time reduction for NNs

```
procedure DATA-AWARE NN TRAINING($\theta_w, \theta_s, \eta, \{x\}$)
    while $\mathcal{L}(D_{\overline{\mathcal{T}}}) > \theta_w$ do
        Perform training with $\{x\}$
    end while
    for all training epochs do
        $D_{\mathcal{T}} = \emptyset$
        $D_{\overline{\mathcal{T}}} =$ Train set
        Calculate $D_{KL}(D_{\overline{\mathcal{T}}}||D_{\mathcal{T}})$
        while $D_{\overline{\mathcal{T}}} \neq \emptyset$ and $D_{KL}(D_{\overline{\mathcal{T}}}||D_{\mathcal{T}}) > 0.05$ do
            Calculate $p_x$
            if $p_x \geq \theta_s$ then
                Perform training with $\{x\}$ and $\eta$
                $D_{\mathcal{T}} = D_{\mathcal{T}} \cup x$
            else
                $E[epoch] = \eta$
                $D[i] = D[i] \cup x$
                continue
            end if
            $D_{\overline{\mathcal{T}}} = D_{\overline{\mathcal{T}}} - \{x\}$
        end while
    end for
    $C =$K-means$\{E[], D[]\}$
    for all C do
        $\eta = C[\eta]$
        $x = C[x]$
        Perform training with $\{x\}$ and $\eta$
    end for
end procedure
```

model termed *Random* that randomly removes training samples but ensures the number of samples removed is equal to the data-aware count. The performance comparisons of the *Random* model are dealt with in the subsequent sections. In order to assess the results of the data-aware approach, we outline the parameters that need to be monitored:

- **Normalized Mutual Information (NMI):** This metric quantifies how different two clusterings of a set of records is against each other. A score of 0 indicates no similarity at all and a score of 1 indicates a complete match. If the score is near 1, it indicates that method used was highly efficient.
- **Test Accuracy:** Illustrates the trained model's accuracy for classifying a new independent test dataset. If the accuracy is high, the model's generalization is good. If the accuracy is low, there is potential overfitting occurring.
- **Warm-up Execution Time:** Amount of time taken for the warm-up to complete before the data-aware method can be deployed.
- **Data-Aware Execution Time:** Amount of time spent training once data-aware methodology is implemented and actively in effect.
- **Image Drop-Off Rate:** Represents the ratio of training samples that were dropped for each epoch.

The relationship between the drop-off rate and θ_s is displayed in Figure 10. Overall, as θ_s increases, the percentage of samples rejected during training increases as well. Still, dataset characteristics, such as the number of classes, impact rejection rates. For instance, at $\theta_s = 0.05$, rejection rates for CIFAR-10, CIFAR-100, and MNIST are 63.4%, 0.5%, and 78.87%, respectively. If θ_s is increased to 0.6, rejection rates escalate for all datasets: CIFAR-10 (99.99%), CIFAR-100 (70.9%), and MNIST (92%). For the evaluations, we select θ_s values that keep the drop-off rate between $[20\%, 70\%]$, ensuring a balance between computational efficiency and model accuracy.

3.5.2 DNN Clustering Performance

The aforementioned metrics NMI, image drop-off rate and training time (in seconds) are evaluated across multiple values of the θ_s threshold with the

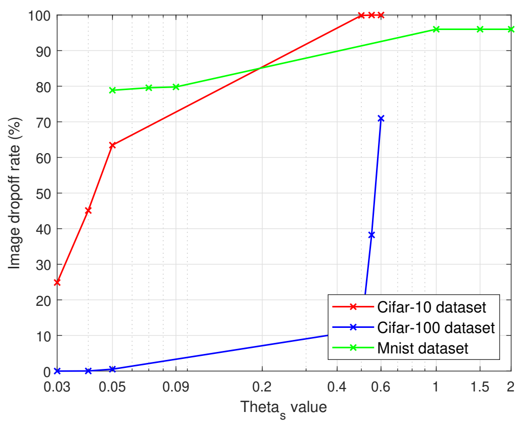


Figure 10: Image drop-off rate over θ_s, for various datasets.

data-aware approach and compared against the *Base* and *Random* models. In this case, the model is trained for 200 epochs. Results are depicted in Table 1. As θ_s increases, the proportion of discarded images rises, leading to a lower NMI, as the model is exposed to fewer training samples per epoch. As expected, the baseline method achieves the highest NMI (86.08%), since it utilizes the complete dataset, whereas in the data-aware and *Random* models, approximately 50% of the images are discarded on average. The lowest NMI for the data-aware method (80.79%) is observed at $\theta_s = 0.09$. Additionally, higher θ_s values result in a reduction in training time, since fewer samples are processed per epoch. The shortest recorded training time (50.14 seconds) is obtained with the data-aware and *Random* methods at $\theta_s = 0.09$, whereas the *Base* model exhibits the longest execution time (54.62 seconds).

In general, the data-aware approach reduces training time around 10%, while suffering a 1.13% loss of NMI compared to the baseline. The *Random* model also improves the training time and decreases NMI because of its sample rejection approach. However, the *Random* method is less effective than the data-aware method as the *Random* method produces a mean value of NMI which is 3.14% lower than the data aware method. This shows how effective the proposed data-aware technique is compared to purely random rejection techniques.

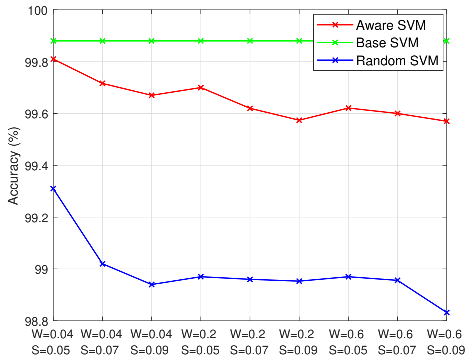


Figure 11: SVM Model test accuracy for the MNIST dataset, over different θ_w and θ_s thresholds.

Table 1: Performance evaluation of the DNN clustering

	$\theta_s = 0.05$	$\theta_s = 0.07$	$\theta_s = 0.09$
Data-aware (NMI)	84.4%	82.86%	80.79%
Data-aware (Training time)	$50.94s$	$49.7s$	$48.4s$
Base (NMI)	86.08%	86.08%	86.08%
Base (Training time)	$55.62s$	$55.62s$	$55.62s$
Random (NMI)	81.42%	79.66%	77.53%
Random (Training time)	$50.94s$	$49.7s$	$48.4s$
Image drop-off rate	42%	48%	60%

3.5.3 SVM Performance

Figure 11 outlines the SVM model's test accuracy for different combinations of the θ_w and θ_s thresholds. The model was trained for 200 epochs on the MNIST dataset. We use W in place of θ_w and S in place of θ_s to save space. The data-aware approach achieves 99.81% test accuracy at $\theta_w = 0.04$ and $\theta_s = 0.05$. When $\theta_w = 0.6$ and $\theta_s = 0.09$, the accuracy drops to the lowest value of 99.57%. On average, the data-aware approach is less accurate than the baseline by 0.22%. The baseline SVM model which does not apply θ_w or θ_s, maintains accuracy of 99.88% for all configurations. The effects that θ_w and θ_s have on accuracy are observable; larger θ_w values reduce the duration of the warm-up phase, which causes performance to decrease,

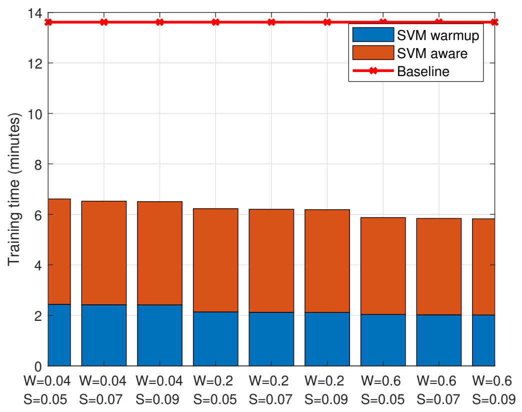


Figure 12: SVM model execution time for *Base* and data-aware implementations.

while larger θ_s values reduces the amount of training tuples per epoch which impacts accuracy. Regarding the random model, results indicate that it suffers a more pronounced accuracy drop relative to the data-aware baseline, especially when the sample pruning during training is more aggressive (e.g., $\theta_w = 0.6$, $\theta_s = 0.09$). The average accuracy loss of the random model is 0.66% relative to the data-aware approach.

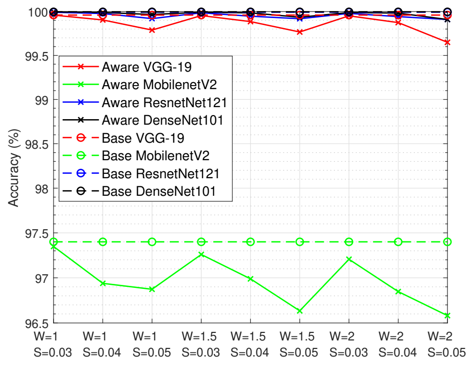
Training time for the SVM model while using the MNIST dataset is displayed in Figure 12. The total training time of the model is classified as warm-up training time and data-aware training time. The models are trained over 200 epochs, and as expected, increasing θ_w (denoted as W) decreases the number of epoch warm-ups which decreases the warm-up training time. Additionally, increasing θ_s (denoted as S) leads to more rejected tuples per epoch leading to further reduction in training time. The *Base* model's training time, however, is not affected from θ_w and θ_s. When $\theta_w = 0.6$ and $\theta_s = 0.09$, the fastest training time of 5.82 minutes is achieved with the data-aware method. Meanwhile, the slowest execution time in the data aware configuration occurs at $\theta_w = 0.04$ and $\theta_s = 0.05$, which is 6.61 minutes. The data-aware method's average training duration is 6.19 minutes which provides 7.41 minute speedup over the *Base*, accelerating the procedure by concluding in less than half of the time, compared to the Base method.

3.5.4 DNN image classification performance

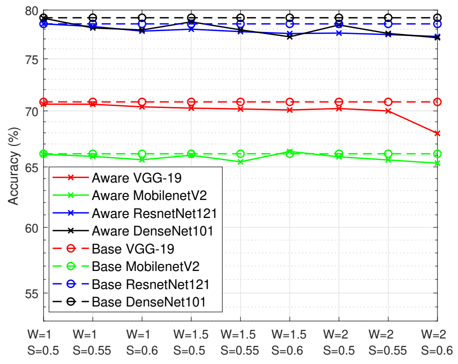
The test accuracy for the CIFAR-10 dataset across different θ_w and θ_s values is shown in Figure 13a. The DNN models undergo training for 200 epochs with a batch size of 128. Among the tested models, DenseNet121 demonstrates the highest resilience, as the data-aware methodology causes only a minimal accuracy drop of 0.08% even at $\theta_w = 2$ and $\theta_s = 0.05$. For lower thresholds like $\theta_w = 1$ and $\theta_s = 0.03$, the impact on ResNet101 is nearly negligible, reducing accuracy by just 0.00087%. In contrast, MobileNetv2 is the less resilient model in terms of performance affected by the data-aware approach, experiencing a reduction in accuracy of 0.81% with high threshold values and 0.051% when lower thresholds are used. The average accuracy reduction per model is as follows: VGG-19 = 0.1%, MobileNetv2 = 0.43%, ResNet101 = 0.04%, and DenseNet121 = 0.02%. It is safe to assume that the data-aware method negative impact is bearable across all models with an average decrease of 15%.

After the completion of 200 training epochs, accuracy results for CIFAR-100 are shown in Figure 13b. CIFAR-100 is comprised of 100 classes compared to the 10 in CIFAR-10, making the classification task more complex. Therefore, accuracy is lower for all models. The general pattern is consistent with CIFAR-10; higher values of θ_s lead to a faster decrease in accuracy. The highest accuracy recorded for the data-aware approach is 78.54%, achieved by ResNet101 with $\theta_w = 1$ and $\theta_s = 0.5$, while the lowest accuracy is 65.31%, observed with MobileNetv2 at $\theta_w = 2$ and $\theta_s = 0.6$. MobileNetv2 emerges as the best-performing model with an average accuracy decrease of just 0.33%. ResNet101 is the next best-performing one with an average drop of 0.74%. Last are VGG-19 with 0.80% and DenseNet121 with 1.16%. The CIFAR-100 results suggest that even in more difficult classification tasks, the data-aware methodology does not affect the results significantly. On average, the accuracy reduction across all experiments is 0.76%.

The comparisons between the *Base* implementation and data-aware implementation regarding the time taken to train the DNN models are spotlighted in figures 14a, and 14b. The details of the outcome illustrate that the data-aware strategy is far more superior than the *Base* approach in terms of efficiency in training. As for the CIFAR-10 dataset, the data-aware method achieves the speedup factors of: $1.7x$ for VGG-19, $1.7x$ for MobileNetv2, $1.68x$ for ResNet101, and $1.71x$ for DenseNet101. Likewise, for the training on the CIFAR-100 dataset, the methodology is able to reduce the time taken to train the model by $1.47x$ for VGG-19, $1.51x$ for MobileNetv2, $1.46x$ for ResNet101 and $1.72x$ for DenseNet101. The data aware method, on average, over both data sets reduces the time taken in training the models by $1.62x$. This proves that the methodology is effective at speeding the learning process



(a) CIFAR-10 Dataset



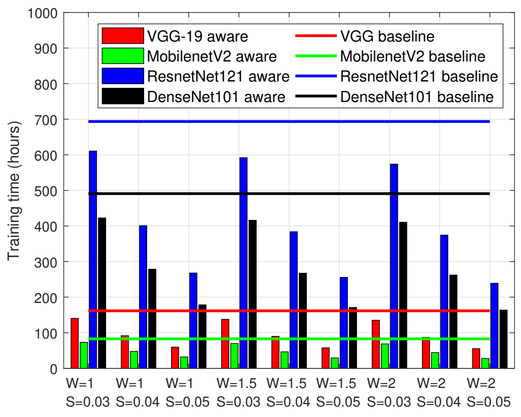
(b) CIFAR-100 Dataset

Figure 13: DNN Model accuracy for the CIFAR datasets, over different θ_w and θ_s thresholds.

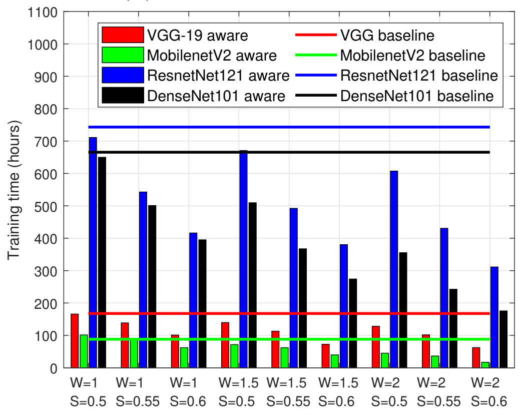
up. For the DenseNet101 model trained on the CIFAR-10, the training time is highly decreased by a stunning $2.99x$ when the parameters $\theta_w = 2$ and $\theta_s = 0.05$ are set. Overall, a higher value for the θ_s parameter value achieves lower training times which greatly improves the effectiveness and flexibility of the proposed method.

3.5.5 Performance Comparison

In this subsection, we analyze the performance of the proposed data-aware methodology relative to an analogous approach which disregards an equivalent portion of data samples in the course of model training.. Specifically,



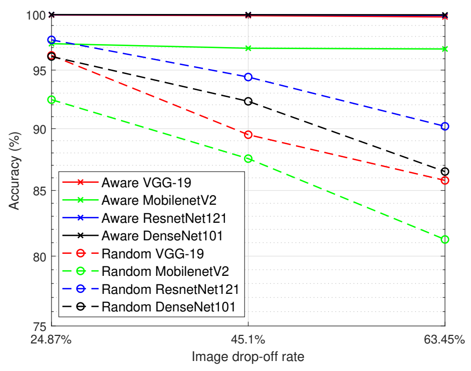
(a) CIFAR-10 Dataset



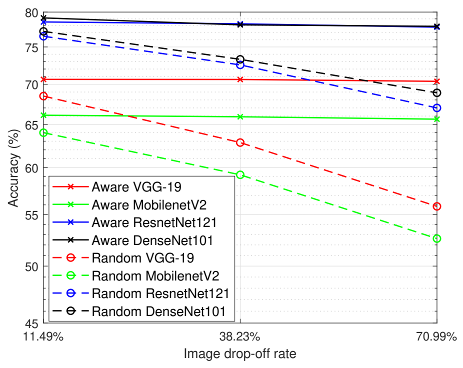
(b) CIFAR-100 Dataset

Figure 14: DNN model execution time using the CIFAR datasets, for baseline and data-aware implementations.

we assess the aforementioned *Random* method which eliminates data points uniformly at random instead of leveraging the structured selection process of the data-aware technique. As a result, sample rejection occurs randomly at each training epoch. Figures 15a and 15b present the comparative results between the data-aware and random methods. For CIFAR-10, as shown in Figure 15a, the data-aware approach consistently outperforms the random technique, achieving an average accuracy improvement of 8.38%. In certain cases, such as with the VGG-19 model, the proposed methodology attains a 13.99% higher accuracy, with the performance gap widening as the image drop-off rate increases. Notably, the data-aware technique maintains high ac-



(a) CIFAR-10 Dataset



(b) CIFAR-100 Dataset

Figure 15: Data-aware methodology accuracy comparison over the Random method, for different training sample drop-off rates, using the CIFAR datasets.

curacy even when 63.45% of the data points are omitted from training. The difference between the data-aware and random techniques becomes more evident as task complexities increase. This is clearly visible with the results of the data-aware method on the CIFAR-100 in Figure 15b, where the data-aware method yielded an average improvement of 6.72% in comparison to the random method. With respect to the model and image drop-off rate, there was an improvement of accuracy from 1.94% to 14.53%. Most significantly, there were no instances where the random method outperformed the data-aware method for both CIFAR-10 and CIFAR-100.

3.5.6 Discussion and lessons learned

In this subsection, we reflect on the lessons learned from the experimentation processes and assess what problems might emerge when implementing the suggested methodology.

- **Supported data types**: The data-aware approach defined in this document is aimed exclusively at supervised learning techniques using image-based data.
- **Application to models requiring large datasets for convergence**: DNNs with a high number of parameters typically demand substantial amounts of training data. If such models are trained with insufficient data, there is a risk that the loss function may stagnate, preventing further improvements in accuracy. In order to solve this, appropriate adjustments of the θ_w and θ_s parameters would have to be done. Decreasing θ_w permits the model to train for a greater number of epochs before implementing the data-aware strategy, which improves convergence but prolongs training time. A higher value of θ_s can be used in order to counterbalance this since it reduces training time while imposing a very small cost in accuracy.
- **Risk of overfitting due to scarce training data**: Overfitting is one of the more likely problems for DNNs which have an excessive number of parameters, especially when the training dataset is not very large. It is indicated when the loss function is minimized, obtaining high training accuracy with low eventual testing accuracy. Our proposed methodology is based on selectively removing training samples, so it poses the danger of overfitting in large models. By reducing the θ_s parameter, this challenge can be partially solved by making sure that more samples are kept for training which reduces the odds of overfitting.To offset the increase in training time, θ_w may be increased to facilitate faster training.

3.6 Summary and Key Findings

The enhancements in performance with respect to ML/DL training within EC were especially visible in resource-limited situations. Our approach towards data management within EC unlocked new possibilities. For example, spatially aware management techniques along with loss function minimization were able to maintain model quality at a fraction of the training time. The most notable key points are as follows:

- **The achieved training time reduction was 50%**: This improvement stems from selective management processes for various datasets, which removed inefficient data processing during training.
- **Model accuracy was maintained**: The high model performance with low resource expenditure during shortened training times was sufficient to enable resource-efficient high-quality predictions.
- **Adaptability to constraints**: The proposed methodologies enabled surrogated edge nodes to execute real-time analytics on devices with severely limited computational capabilities.

4 Transfer Learning in Edge Computing

4.1 Context and Motivation

With TL, models developed for a specific set of tasks can be adapted for different tasks, which results in a reduction in the time and resource expenditure for training. This is useful in EC because the data are usually limited and so are the resources. However, TL on the edge has its own set of challenges, such as dealing with heterogeneous data sources and ensuring model performance in dynamic environments. The ability to adapt models to new tasks as rapidly as possible is crucial in EC because devices need to operate over data in real time with computing resources limitations. The available TL mechanisms do not cater to the specificity of the edge systems, and therefore, are inefficient. This research proposes an uncertainty driven model to improve TL for edge environments, filling the gaps of handling diverse data distributions and efficient model reuse across devices. The goal is to enable faster, more adaptive learning for a wide range of edge applications.

4.2 Related Work

TL techniques are increasingly being researched for their potential to improve model accuracy and reduce training times in EC environments, addressing the challenges of dynamic data and limited computational resources identified in the introduction. The division of TL stems from the similarity of the feature spaces between the source and the target task domains. It is usually divided into two types. If the feature spaces are the same, it is referred to as homogeneous TL. On the other hand, when feature spaces differ, it is referred to as heterogeneous TL.

4.2.1 Homogeneous Transfer Learning

In the setting of homogeneous TL, four primary categories are distinguished based on the mode of transfer: instance-based, feature-based, parameter-based, and relational-based [83]. In instance-based TL, weights are assigned to instances in the source domain by their marginal distributional differences from the target domain, making them reusable for training the target domain. The authors in [84] describe a multi source domain adaptation framework (2SW-MDA) that uses a weighting strategy to evaluate each source's relevance. 2SW-MDA accounts for marginal and conditional probability distribution gaps between source and target domains. A target model is generated using weighted source samples and the labeled target data. Methods based on features seek to minimize the divergence in marginal distributions between domains. A framework for dimensionality reduction that maps data into a latent space for domain adaptation is proposed in [85]. These methods employ unsupervised and semi-supervised feature extraction techniques to bridge domain distribution differences.

Parameter-based strategies facilitate knowledge transfer by leveraging shared model parameters across several source domains to construct a more effective model for the target domain. In [84], an alternative multi-source domain adaptation framework (CP-MDA) is introduced, which uses a weighted approach to correct conditional probability distribution gaps between the domains. This framework can combine models from several source domains to label the unlabeled target data, to subsequently build a better target model that contains the newly labeled and the pre-existing labeled target samples. [86] comes up with the idea of a bellwether domain, which acts as the most efficient predictive source for all other domains and enables knowledge transfer through a quality predictor constructed using data from the bellwether domain. In [87], the authors build a theoretical framework on parameter transfer for linear regression models. The results show that transfer effectiveness for a new input vector depends on its eigenbasis representation with respect to the model parameters. In addition, they propose a statistical test to assess whether a tuned model has a lower quadratic prediction risk than a base target model for a given input.

TL which relies on relations focuses on capturing structural relations between the source and target domains for knowledge transfer. In [88], the authors propose a language-bias-based transfer learning algorithm (LTL) that permits cross-domain transfer. LTL transfers logical rules by detecting predicates in the source domain that have relational counterparts in the target domain. The transferred rules are refined using theory refinement methods. In [89], an instance-based and a parameter-based transfer approach is blended in a single model. The adaptive transfer Gaussian process (AT-GP) model uses a transfer kernel that quantifies the resemblance of different tasks by modeling the correlation of function outputs (labels) across tasks. Depending on the task similarity, AT-GP shares parameters within the kernel function together with relevant data from the source task to the target task.

4.2.2 Transfer Learning in Edge Computing

Recently, there has been a focus on applying TL in EC. Hypothesis TL attempts to explain the trade-off between accuracy and network traffic in [90], where patterns are drawn from data produced by distributed devices. A privacy-preserving, TL-enabled, CNN framework for 5G industrial edge networks is proposed in [91]. The paper defines a joint energy and latency optimization problem. The authors solve it through a decomposition approach, which breaks the problem into an uploading decision sub-problem and a wireless bandwidth allocation sub-problem. A TL-based EC framework for home health monitoring is developed in [92]. This approach uses a pretrained CNN that is subsequently transferred to edge devices by performing fine-tuning with few labeled samples. The authors in [93] present FT-IoMT Health, a gradient reduction federated random variance algorithm designed to improve communication in mobile EC contexts. This system allows for the confidential aggregation of data across multiple domains with the use of privacy guarantees. In addition, TL is applied to reduce the negative impact of different detection methods on performance.

An automated classification system for date fruits utilizing DL techniques is proposed in [94], where pretrained models are adapted to improve the classification precision. The work in [95] provides a proactive caching strategy (LECC) for reducing transmission expenditure. Content popularity is estimated using TL, and the estimate is used to formulate and solve the proactive caching optimization problem. In [96], grouped TL is described as a non-linear mixed-integer programming problem with the objective of minimizing costs over an extended period. [97] describes a collaborative multi-object tracking system that uses TL at the edge. In addition, [98] describes Cartel, a system meant for collaborative learning in edge clouds. Cartel performs adaptive model sharing among edge nodes to enable rapid responses to changes in expected statistical value characteristics. For efficient TL, Cartel uses metadata-based techniques that minimize the need for large data transfers between edge nodes.

4.2.3 Existing literature gap

Despite the increasing research on TL, particularly its applications in EC, current literature predominantly addresses theoretical frameworks and broad categorizations such as instance-based, feature-based, parameter-based, and relational-based approaches. However, these studies often overlook the practical challenges associated with dynamically changing environments, typical of EC. There is a lack of comprehensive mechanisms to effectively handle the uncertainty and variability in data distributions across edge devices. Moreover, existing methods largely depend on external datasets with potentially different statistical distributions, leading to sub-optimal model performance when applied to edge environments. The need for robust methodologies that can detect and leverage similar statistical distributions in real-time to optimize TL processes remains insufficiently explored.

4.3 Problem Statement

Building on the formulations in Section 1.6, each input data vector x (independent variable) is associated with an output label y (dependent variable). Each edge node n_i randomly selects a sample of M data vectors and computes their mean vector m_i and variance vector v_i at different epochs. If a notable deviation is detected between the distributions of the local dataset and an incoming data batch (which is then incorporated into the dataset), the values of m_i and v_i are updated. The mean vector m_i has d dimensions and consists of the mean value for each feature: $m_i = (m_{i_1}, m_{i_2}, \ldots, m_{i_d})$. Similarly, the variance vector v_i has d dimensions, containing the variance values for each feature: $v_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_d})$. Each node transmits its mean and variance vectors to its peers whenever there is an update in the dataset information. Upon receiving a new data batch, the node n_i computes the mean (mb), variance (vb) vectors from a randomly selected subset of M data vectors from the batch. Consequently, we obtain mean/variance vectors for both the existing dataset and the new data batch. The vector mb contains the mean value for each feature: $mb = (mb_1, mb_2, \ldots, mb_d)$. Similarly, the variance vector vb includes the variance values: $vb = (vb_1, vb_2, \ldots, vb_d)$. Figure 16 illustrates this process visually.

In specific instances, there may be a point in time after which new data batches cannot be obtained or are too costly. For instance, consider a healthcare institution that wants to sort patients form different hospitals according to how likely they are to develop a certain illness. Medical experts have to provide data labels, however, once a reasonable sized labeled dataset is gathered, the organization can use a trained model to predict labels for new patient records that need attention. Another case comes into play when data with labels is not plentiful, or is completely absent. Take the case of a dataset

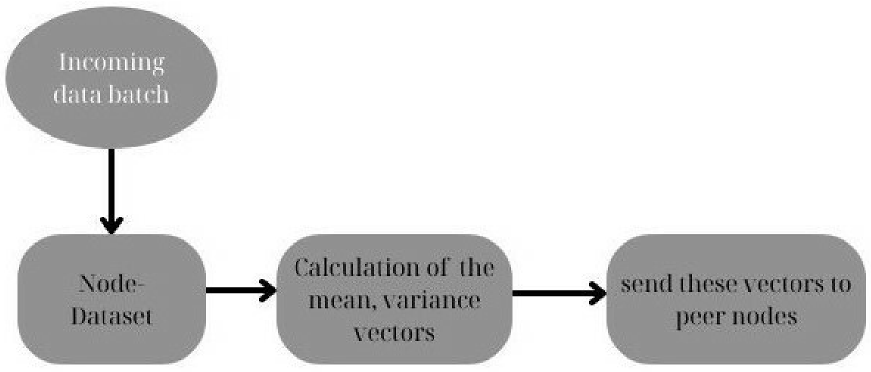

Incoming data batch
Node-Dataset
Calculation of the mean, variance vectors
send these vectors to peer nodes

Figure 16: Fundamental components of this setting

with traffic patterns for different cities where the label of interest is whether there is blockage during certain times of the day. Assume that New York's dataset is completely labeled, and Paris' dataset has only a few labeled samples. Training a model on Paris data would not be helpful, instead it would be better to label the Paris dataset for use with a model trained on New York data because traffic patterns across cities are similar.

However, a critical factor to consider is whether the incoming data batch shares a similar probability distribution with the node's dataset. If their distributions differ significantly, labeling the new batch using the local trained model would likely result in poor accuracy. A more effective strategy would be to use a model from another node that has a dataset with a similar probability distribution to the incoming data batch. After labeling, the model can be retrained with the newly incorporated data, making it more robust to a wider range of inputs. Conversely, if the new data batch follows a similar probability distribution as the node's dataset, local labeling can be performed without retraining the model. Since data originating from the same distribution has minimal impact on further improving the model, retraining may not be necessary.

The core challenge addressed in this study is to determine when to transfer knowledge (i.e., when to exchange models) and from which peer node to fetch the knowledge (i.e., selecting the most suitable peer). Consequently, it is essential to quantify the similarity (or dissimilarity) between different datasets, as well as between a dataset and a newly received data batch.

In this effort, domain adaptation is not considered, and its exploration is left for future research. To support our approach, we make the following assumptions:

- The samples drawn from both datasets and incoming data batches are assumed to be independent.
- Every feature in the input vectors (x) is drawn from a continuous probability distribution.
- Every data batch contains a finite set of (M) input vectors.
- All datasets share the same feature space.

4.4 Uncertainty-Driven Transfer Learning Mechanism

In this section, we outline the mechanisms for TL in an EC environment. Our objective is to establish a method for evaluating multi-dimensional distribution-based similarity and dissimilarity between two datasets or between a dataset and an incoming data batch. Each dimension of the input vectors within a dataset or data batch is assumed to follow a distinct probability distribution. The true parameters of the population distribution from which dataset samples originate are typically unknown and cannot be directly measured. However, if a dataset contains a sufficiently large historical record, its mean and variance can be estimated to approximate the underlying distribution. Each sample is considered a random draw from this distribution. Given a lower-bound sample size, the Central Limit Theorem allows for the use of approximate Z-tests [99], F-tests [100], or the Kolmogorov-Smirnov test [101] to assess distributional similarities and differences across each dimension of a dataset and a data batch, or between two datasets.

Z-tests and F-tests are applied due to their computational efficiency, as they rely only on mean and variance vectors. The Kolmogorov-Smirnov test, on the other hand, requires two data samples from the entities being compared and in scenarios where datasets from peer nodes must be analyzed, a sample must be transmitted to the local node for comparison. Despite this necessity, the Kolmogorov-Smirnov test remains computationally lightweight compared to full dataset comparison methods. To further optimize the process, we first apply Z- and F-tests to detect significant distributional differences, thereby minimizing unnecessary data exchanges. The cost associated with manually labeling a dataset or a portion of data grows proportionately to its size, while the workload on each edge node varies as tasks are completed or newly allocated. To address these ambiguities, we apply fuzzy logic along with distribution-based similarity measures in an effort to minimize labeling costs as well as system-wide computing ones.

4.4.1 Similarity Based on Multidimensional Distributions

The Kolmogorov-Smirnov two-sample test (K-S two-sample test) is a non parametric statistical test that evaluates whether two independent one-dimensional probability distributions differ significantly [101]. We apply this test to determine if an incoming data batch follows a distribution similar to a node's dataset or if two datasets exhibit comparable statistical properties. To perform this analysis, a random sampling process is conducted whenever two datasets or a dataset and a data batch need to be compared. From each entity, we extract a subset of M data vectors. Each dimension of these sampled vectors is treated as an independent random variable, denoted as $(P_1, P_2, \ldots, P_d)$ for the first sample P and $(Q_1, Q_2, \ldots, Q_d)$ for the second sample Q. The Cumulative Distribution Function (CDF) of a random variable V_i is defined as:

$$F(\alpha) = P(V_i \leq \alpha), \tag{22}$$

which represents the probability that V_i takes a value less than or equal to α. Since the true distribution is unknown, we approximate it using empirical distributions. The empirical CDF for a sample of size M is given by:

$$F_M(\alpha) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}(V_i \leq \alpha), \tag{23}$$

where $(V_1, V_2, \ldots, V_M)$ represent the sampled data points. To quantify distributional similarity across all dimensions, we compare the empirical CDFs of the two samples, F_M^1 and F_M^2, using the Kolmogorov-Smirnov test statistic $S_{i,M}$:

$$S_{i,M} = \max_g \left| F_M^1(g) - F_M^2(g) \right|, \tag{24}$$

where g belongs to the set of all observed values, i.e., $g \in \{P_{i_1}, P_{i_2}, \ldots, P_{i_M}\} \cup \{Q_{i_1}, Q_{i_2}, \ldots, Q_{i_M}\}$, with $i = 1, \ldots, d$.

Definition 7. *A dataset and a data batch are considered to share a similar distribution along the i-th dimension if the Kolmogorov-Smirnov statistic satisfies the condition $S_{i,M} \geq \beta$, where β is a predefined threshold.*

To generalize this assessment across multiple dimensions, we aggregate the results from all dimensions into a similarity function.

Definition 8. *The overall multidimensional similarity between two datasets, P and Q, is computed as:*

$$sim(P, Q) = \frac{\sum_{i=1}^{d} \mathbb{I}(S_{i,M} \geq \beta)}{d}, \tag{25}$$

where d represents the total number of dimensions.

The proposed similarity function adheres to the following fundamental properties:

- **Reflexivity**: $sim(P, P) = 1$
- **RSymmetry**: $sim(P, Q) = sim(Q, P)$
- **RBoundedness**: $0 \leq sim(P, Q) \leq 1$

Definition 9. *If the computed multidimensional similarity $sim(P, Q)$ meets or exceeds a predefined threshold γ, then the two datasets are considered to originate from similar distributions, i.e., $sim(P, Q) \geq \gamma$.*

4.4.2 Dissimilarity Based on Multidimensional Distributions

In scenarios where an incoming data batch exhibits a distribution distinct from the dataset residing at the node, or when comparing datasets from different nodes, performing Kolmogorov-Smirnov tests between a local node and all peers can lead to excessive latency. This is because transferring $N-1$ data samples to the local node incurs significant communication overhead. To mitigate this issue, we leverage the mean and variance vectors of peer nodes, which are already stored at the local node. Instead of transferring entire data samples, we conduct Z-tests to compare mean values and F-tests to assess variance differences between the local node and its peers. These statistical tests form the basis of a dissimilarity function that aids in identifying nodes with significantly different distributions.

Definition 10. *Given two samples P and Q, characterized by mean vectors mp and mq, variance vectors vp and vq, and a sample size M, the Z-statistic for each dimension is computed as:*

$$z(i) = \frac{mp_i - mq_i}{\sqrt{\frac{vp_i}{M} + \frac{vq_i}{M}}}, \quad i = 1, \ldots, d \tag{26}$$

The null hypothesis H_0 states: "There is no significant difference in the mean values for the i-th dimension," while the alternative hypothesis H_1 asserts: "A difference exists in the mean values for the i-th dimension." The statistical significance level p is determined based on the application's tolerance for error. From this value, we derive the corresponding critical value z_{score}. If the computed Z-statistic exceeds z_{score}, we accept the alternative hypothesis. Otherwise, we do not reject the null hypothesis due to insufficient evidence.

Definition 11. *Given two samples P and Q with variance vectors vp and vq, respectively, and sample size M, the F-statistic for each dimension is computed as:*

$$f(i) = \frac{vp_i}{vq_i}, \quad i = 1, \ldots, d \tag{27}$$

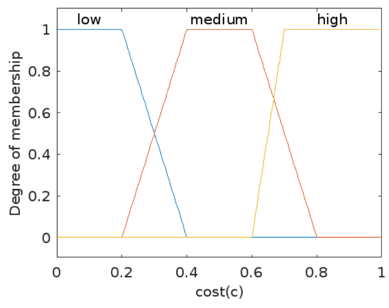
The null hypothesis H_0 states: "There is no significant difference in the variance values for the i-th dimension," whereas the alternative hypothesis H_1 claims: "A significant variance difference exists for the i-th dimension." The significance level p is used to determine the critical value f_{score}. If the computed F-statistic exceeds f_{score}, we accept the alternative hypothesis. Otherwise, the null hypothesis remains inconclusive due to insufficient evidence.

Definition 12. *Considering the computed Z- and F-statistics for all dimensions of two samples, along with their respective critical values at significance level p, the multidimensional distribution-based dissimilarity is formulated as:*

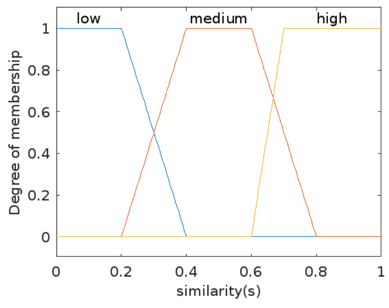
$$dissim(P, Q) = \frac{\sum_{i=1}^{d} \mathbb{I}(z(i) > z_{score} \lor f(i) > f_{score})}{d} \tag{28}$$

This dissimilarity function satisfies the following properties:

- **Reflexivity**: $dissim(P, P) = 0$



(a)



(b)

Figure 17: Fuzzy membership functions representing the input variables: cost (a) and similarity (b)

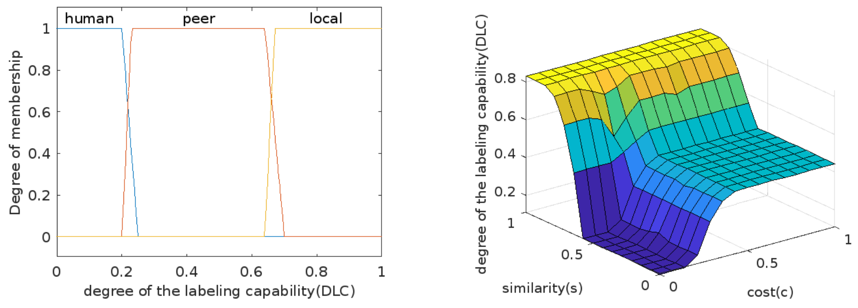
- **Symmetry**: $dissim(P, Q) = dissim(Q, P)$
- **Boundedness**: $0 \leq dissim(P, Q) \leq 1$

Definition 13. *Given the computed dissimilarity measure $dissim(P, Q)$ and a predefined threshold γ, the two datasets are considered to have significantly different distributions if:*

$$dissim(P, Q) > 1 - \gamma \tag{29}$$

4.4.3 Fuzzy Technique for Labeling and Training

For each incoming data batch containing unlabeled instances, incurs a labeling cost if human experts are required. According to above calculations, the data batch exhibits a similarity degree, ranging from 0 to 1, with the local node's dataset. In cases where a dataset primarily consists of unlabeled data, the similarity is assigned a value of zero, and the corresponding human expert labeling cost remains applicable. To mitigate this cost, we can leverage a model trained on a similar dataset available at a peer node. To determine the labeling approach, we introduce a fuzzy logic-based mechanism that takes into account two parameters: labeling cost and similarity (input variables). The linguistic values (fuzzy sets) for both cost and similarity are categorized as (low, medium, high). These input values influence the final decision-making process. The membership function for each fuzzy set value is trapezoidal and is illustrated in Figure 17.



(a) (b)

Figure 18: Fuzzy membership functions representing the output variable DLC (a) and the inferred DLC score (b) based on the input variables

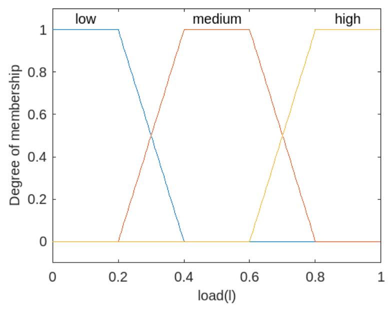
The output variable, termed *Degree of Labeling Capability* (DLC), determines the optimal method for labeling the unlabeled data (human expert, local model, or peer model). When the labeling cost is low and similarity is low/medium, a human expert performs the labeling to ensure high accuracy at minimal cost. If similarity is high, labeling is conducted using the local model, achieving acceptable accuracy without incurring labeling costs. Finally, when the labeling cost is medium/high and similarity is low/medium, the peer model is used to balance accuracy and cost efficiency, considering that optimal accuracy cannot be attained at a low cost. For defuzzification, we employ the centroid method, which extracts a single numerical value from the aggregated fuzzy set, converting the fuzzy inference results into a precise decision. Based on this value and the DLC membership function, the optimal labeling strategy is determined. The membership function is depicted in Figure 18.

The fuzzy rules used to infer DLC are outlined in Table 2.

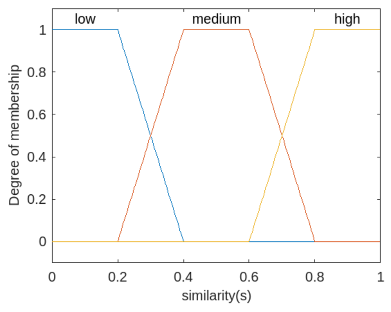
Another critical point of attention is the integration of the labeled data batch into the local dataset. If a data batch has a significantly different distribution from the local dataset, the model must be retrained upon its incorporation. Additionally, when a dataset containing predominantly unlabeled data is labeled, a new model must be trained. However, if the local node is already handling a high computational load, immediate training may lead to system failures or increased response times. In such cases, postponing training and delaying the integration of the data batch into the dataset is

Table 2: Set of fuzzy logic rules used to determine the output variable DLC, based on input variables cost and similarity

Rule	c (cost)	s (similarity)	DLC (Degree of Labeling Capability)
1	low	low	human
2	low	medium	human
3	low	high	local
4	medium	low	peer
5	medium	medium	peer
6	medium	high	local
7	high	low	peer
8	high	medium	peer
9	high	high	local



(a)

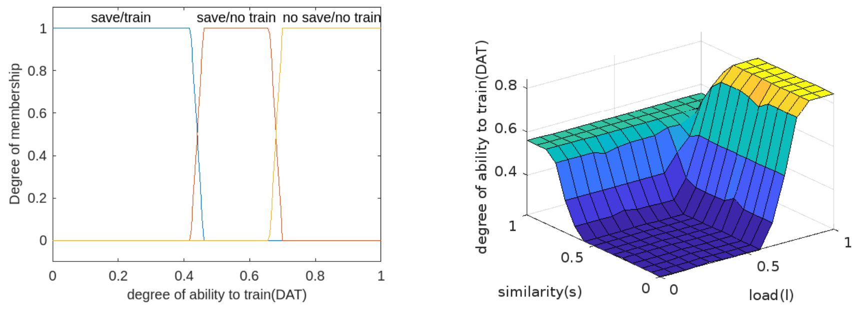


(b)

Figure 19: Fuzzy membership functions representing the input variables: load(a) and similarity (b)

preferable. To regulate the training process, we introduce a second fuzzy logic-based mechanism, which considers two input parameters: *computational load* and *similarity*. The linguistic values (fuzzy sets) for both input variables are categorized as (low, medium, high). The membership functions for these fuzzy sets are trapezoidal, as defined in Figure 19.

The output variable, termed *Degree of Ability to Train* (DAT), determines whether the data batch should be integrated into the dataset and whether the model should be retrained. When computational load is low/medium and similarity is low/medium, the data batch is incorporated into the dataset, and the model is retrained, as significant distribution differences exist and the



(a) (b)

Figure 20: Fuzzy membership functions representing the output variable DAT (a) and the inferred DAT score (b) based on the input variables

local node can handle the additional load. When similarity is high, the data batch is saved in the dataset, but retraining is skipped, as the distribution differences are minimal. Conversely, if the computational load is high and similarity is low/medium, the data batch is neither saved in the dataset nor used for retraining, as the node cannot manage the additional processing at that moment. The centroid method is used for defuzzification, extracting a single numerical value from the aggregated fuzzy set and converting the fuzzy inference results into a precise decision. Based on this value and the DAT membership function, the optimal training strategy is determined. The membership function is depicted in Figure 20.

The fuzzy rules used to infer DAT are outlined in Table 3.

The proposed fuzzy logic mechanisms govern both the labeling process for incoming data batches and the subsequent training process following data labeling.

4.4.4 Mechanism for Transfer Learning

To better explain the integration processes of the previous three subsections into a coherent whole, a diagram has been constructed in Figure 21 that shows their interelationships and interactions. Algorithm 4 describes the provided mechanism for TL, which deals with the best approach for the labeling and training activities. In cases where datasets are entirely unlabeled, the similarity is set to zero. For incoming data batches that lack labels, we perform a Kolmogorov-Smirnov test to compute their similarity. Subsequently, the first fuzzy mechanism is triggered to determine the most suitable

Table 3: Set of fuzzy logic rules used to determine the output variable DAT, based on input variables load and similarity

Rule	l (load)	s (similarity)	DAT (Degree of Ability to Train)
1	low	low	save/train
2	low	medium	save/train
3	low	high	save/no train
4	medium	low	save/train
5	medium	medium	save/train
6	medium	high	save/no train
7	high	low	no save/no train
8	high	medium	no save/no train
9	high	high	save/no train

labeling approach based on cost and similarity. If the outcome suggests labeling by a peer node, we conduct Z-tests and F-tests to compute dissimilarity between the dataset or data batch in the local node and datasets from peer nodes. Nodes with significant distribution differences are excluded. For the remaining nodes, Kolmogorov-Smirnov tests are conducted to assess similarity. The peer node with the first acceptable similarity score is selected (to minimize computation), and its model is used for labeling the unlabeled dataset or data batch. Algorithm 5 is a function invoked within Algorithm 4. In particular, using the calculated similarity and the local node's workload, the second fuzzy mechanism is activated to determine the optimal training strategy for the model in the local node.

4.5 Experiments and Results

4.5.1 Setup and Performance Metrics

We simulate an EC environment consisting of ten nodes using Python. To evaluate the performance of our TL mechanism, we conduct a series of experiments using three key performance metrics in comparison with a baseline approach that does not utilize TL. We start with tackling how well our approach is achieving labeling accuracy as a percentage, pla. It quantifies the effectiveness in labeling datasets, or a batch of data that was previously unlabeled. When the pla value is close to 100, the model succeeds in labeling the

Algorithm 4 Transfer Learning Mechanism

```
Input: N, d, beta, gamma
Output: labeling of the unlabeled data-batches, datasets and training of
the models.
for every unlabeled data-batch/dataset do
    if data-batch then
        input labeling cost
        sample M vectors from the data-batch(P) and local dataset(Q)
        calculate $mp, vp, mq, vq$ vectors
        for $i = 1, \ldots, d$ do
            calculate $S_{i,M}$ based on (1)
        end for
        calculate $sim(P, Q)$ based on (2)
    else if dataset then
        sample M vectors from the dataset(P)
        $sim(P, Q) = 0$
        input labeling cost
    end if
    FuzzyLogicSystem1(cost, similarity)
    if DLC$\leq DLC_{human}$ then
        label by human expert
    end if
    if DLC$\leq DLC_{local}$ then
        label with local model
    end if
    if DLC$\leq DLC_{peer}$ then
        for every peer node do
            for i=1,...,d do
                calculate $z(i), f(i)$ based on (3), (4)
            end for
            calculate $dissim(P, Q)$ based on (5)
            if $dissim(P, Q) > 1 - \gamma$ then
                prune away the peer node
            end if
        end for
        for the remaining nodes do
            input sample of M vectors (Q)
            for $i = 1, \ldots, d$ do
                calculate $S_{i,M}$ based on (1)
            end for
            calculate $sim(P, Q)$ based on (2)
            if $sim(P, Q) \geq \gamma$ then
                break
            end if
        end for
        input load
```

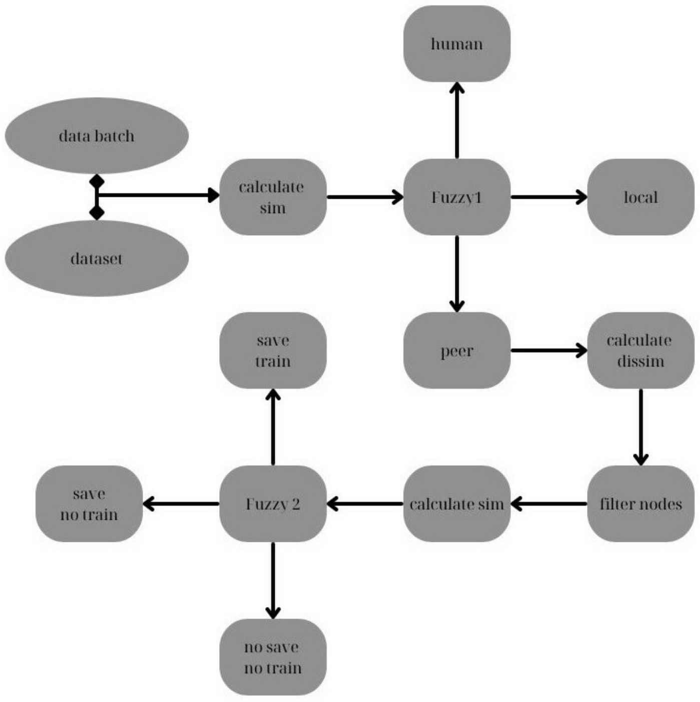


Figure 21: Diagram depicting the TL Mechanism

data. On the other hand, if pla is nearing 0, that signifies failure to label the data. Next we consider the percentage of cost saving (pcs) which indicates how well our mechanism is able to minimize labeling costs. pcs is a ratio of the total cost after TL is applied, to the initial cost without TL. With TL, if the pcs is 100, costs are completely minimized, whereas, if it is 0, it means no cost savings were achieved. We then analyze the percentage of load saving (pls) which has similar meaning as the savings in computational load. Like the previous case, pls is defined as a ratio of the final computational load to the initial computational load. If pls is 100, it means that the the computational load is fully optimized, whereas, if it is 0, there are no savings in terms of computational load.

The experiments are carried out using three synthetic datasets generated via Python with a fixed random seed to ensure reproducibility. Each dataset consists of six continuous input features and a binary output feature (0 or 1). Each input feature follows a unique Gaussian distribution, and the output feature is derived from the input values. The mean and variance parameters

Algorithm 5 Fuzzy-Based Training Decision Function

```
Input: load, similarity
Output: Decision on training of the models and incorporation of the new data.
FuzzyLogicSystem2(load, similarity)
if DLC≤ $DLC_{save/train}$ then
    save the new data and train the model
end if
if DLC≤ $DLC_{save/notrain}$ then
    save the new data and don't train the model
end if
if DLC≤ $DLC_{nosave/notrain}$ then
    don't save the new data and don't train the model
end if
```

of these distributions are set to ensure significant statistical differences between the datasets. These synthetic datasets are then divided among edge nodes and used to generate data batches. Each node's dataset and all data batches originate from a single original dataset. The ML model employed in each node is Logistic Regression, where the hyperparameter 'solver' was set to liblinear. The remaining hyperparameters such as penalty, max_iter, etc. are kept at their default values, i.e., L2 penalty with a weighting of coefficients set to 1.0, 100 respectively . In this setting, three out of the ten datasets within the edge system are left unlabeled, with each of these datasets being a subset of one of the original synthetic source datasets. The unlabeled data batches are randomly assigned throughout the system. Our mechanism is evaluated based on its ability to correctly identify the most appropriate source for TL when needed. Additionally, we assess the overall impact of our mechanism in reducing both labeling costs and computational load within the system.

We extend our evaluation by comparing our approach with Direct Inductive Transfer Learning (DITL) [102], a model that employs importance-weighted TL for regression under dataset shift. DITL assigns weights to instances from the source data to improve predictions for the target dataset, requiring entire datasets to be exchanged for training. In contrast, our Distribution Based Transfer Learning (DBTL) mechanism only requires a small sample exchange for TL. Another key distinction is that in DITL, the source and target datasets must be predefined, whereas DBTL autonomously identifies the most appropriate dataset for TL based on statistical distribution analysis. Additionally, DITL necessitates training a new model, while DBTL leverages an already trained model. We compare these models based on their ability to correctly label/classify the three unlabeled datasets in the system, excluding data batches. Performance is evaluated using standard classification metrics, including accuracy, precision, recall, and F1-score. Finally, we conduct a case study to compare our model with the approach presented in [103], which leverages knowledge transfer from synthetic data to minimize the need for real data. Since no specific denomination is given, we will denote this model as DBO (Detecting Building Occupancy), based on the title of the referenced paper.

Case Study: The Ministry of Civil Protection aims to predict occupancy levels in four buildings (A, B, C, D) to facilitate timely responses in the event of an earthquake. Buildings A and B have a capacity of 200 occupants, while Buildings C and D can accommodate up to 600 occupants. Sensor data, including temperature, humidity, and CO2 levels, are collected over time in each building. Buildings A and C are equipped with occupancy sensors to track human presence, whereas Buildings B and D lack such sensors. Each building collects its sensor data through an edge node and can train predictive models. The trained models from Buildings A and C, where occupancy data are available, could be utilized for TL to estimate occupancy levels in Buildings B and D, provided that their statistical distributions are similar. The datasets representing sensor data from all four buildings are generated using Energy+, without any preprocessing. We analyze the performance of our method in contrast to the DBO model with regard to occupancy estimation for Buildings B and D. Data batches, as in the previous experiment, are ignored. The evaluation is performed using the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) metrics typical for regression analysis. The predictive model employed is Support Vector Regression (SVR). Two SVR models are trained using data from Buildings A and C (where occupancy information is provided). Furthermore, to comply with the DBO model, datasets are merged by combining data for one week from Building C with Building A and the other way around. This enables the training of two SVR models using the DBO approach which incorporates data from various sources.

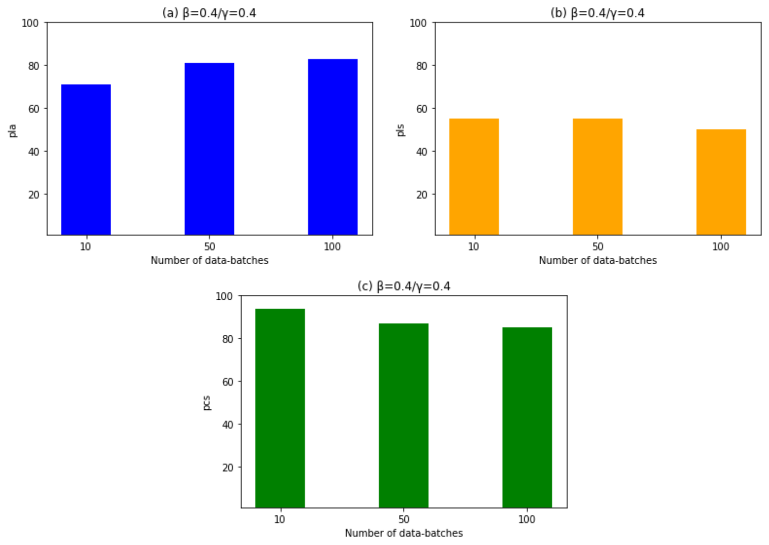


Figure 22: Assessment of pla, pcs, pls ($\beta = 0.4, \gamma = 0.4$)

4.5.2 Performance Evaluation

The following Figures present experimental results for different values of the similarity thresholds. In Figures 22, 23, 24 we observe that the proposed mechanism attains higher pla when $\beta = 0.8, \gamma = 0.8$. This suggests that increasing the similarity thresholds positively impacts pla. In contrast, pcs and pls appear largely unaffected by the increase in the aforementioned thresholds. Across all scenarios, our method consistently achieves pla between 71% and 95%, pcs between 85% and 94% and pls between 50% and 57%. When γ, β are set to low values we observe lower values in accuracy. This is due to the fact that the mechanism may select the first dataset with minimal acceptable similarity to transfer knowledge from, which could lead to knowledge transfer from a less suitable source. On the other hand, cost saving and load saving remain stable regardless of the threshold values. The labeling cost is zero whenever a dataset or a data batch are labeled by the model in the local node or the model of a peer node, regardless of the accuracy. In the second case every time a data-batch is being saved without re-training the model, the additional load is zero.

Table 4 presents a comparison between the performance of the proposed methodology on data-batches and on full datasets, considering the accuracy (%) metric (pla).

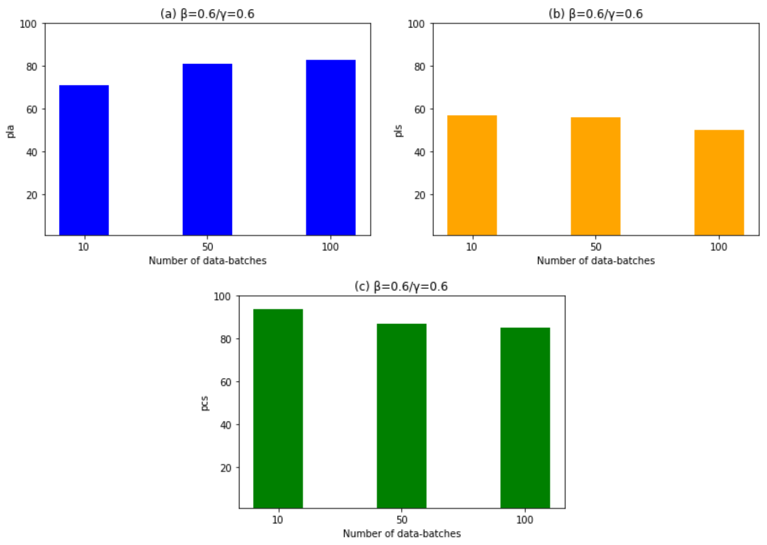


Figure 23: Assessment of pla, pcs, pls $(\beta = 0.6, \gamma = 0.6)$

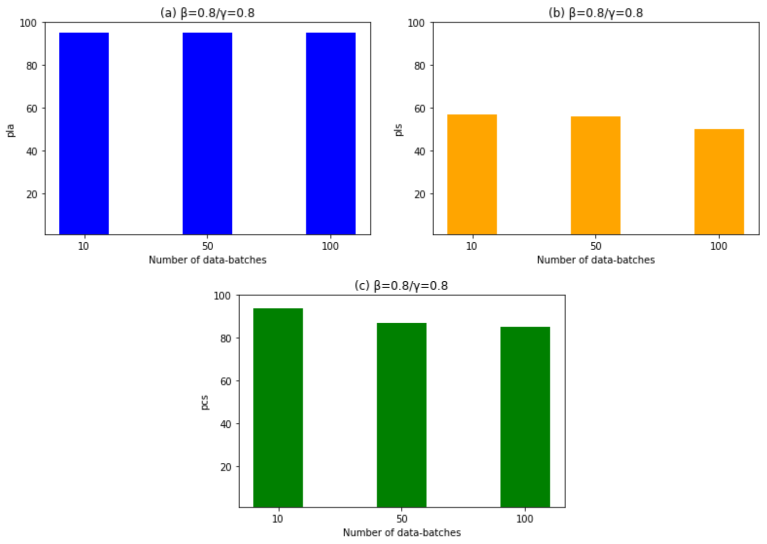


Figure 24: Assessment of pla, pcs, pls $(\beta = 0.8, \gamma = 0.8)$

Table 4: Accuracy (%) comparison between the proposed approach applied to data-batches and its performance on full datasets

Threshold values	3 datasets	10 data batches	50 data batches	100 data batches
$(\beta = \gamma = 0.4)$	80	71	81	83
$(\beta = \gamma = 0.6)$	80	71	81	83
$(\beta = \gamma = 0.8)$	96.5	95	95	95

Table 5: Evaluation of DITL and DBTL across different β, γ threshold values, focusing on their ability to accurately label/classify unlabeled datasets

Metric	DITL	DBTL $(\beta = \gamma = 0.4)$	DBTL $(\beta = \gamma = 0.6)$	DBTL $(\beta = \gamma = 0.8)$
Accuracy(%)	51	65	95	95
Precision(%)	67	67	100	100
Recall(%)	35	48	90	90
F1 score(%)	46	55	95	95

Table 5 reports the outcomes of the DITL versus DBTL comparison for different values of the similarity thresholds. Across all scenarios the proposed approach (DBTL) consistently outperforms DITL. As previously noticed, low values of γ, β lead to lower values in all metrics. Once again, this is due to the fact that the mechanism may select the first dataset with minimal acceptable similarity to transfer knowledge from, which could lead to knowledge transfer from a less suitable source. DBTL proves advantageous and in real setups where datasets with similar distribution are distributed among various nodes of the system.

Table 6 presents the comparison between DBO and DBTL considering building B for different values of the similarity thresholds, while Table 7 shows the corresponding results for building D. In all cases DBTL consistently outperforms DBO. As noted and interpreted previously, low values of γ, β lead to lower values in all metrics.

Table 6: Evaluation of DBO and DBTL across different β, γ threshold values, focusing on their ability to accurately label/predict unlabeled datasets (Building B)

Metric	DBO	DBTL	DBTL
	(1 week)	($\beta = \gamma = 0.3$)	($\beta = \gamma = 0.6$)
RMSE	204.37	182.11	162.51
MAE	200.73	178.01	157.90

Table 7: Evaluation of DBO and DBTL across different β, γ threshold values, focusing on their ability to accurately label/predict unlabeled datasets(Building D)

Metric	DBO	DBTL	DBTL
	(1 week)	($\beta = \gamma = 0.3$)	($\beta = \gamma = 0.6$)
RMSE	190.37	152.45	151.54
MAE	180.47	125.30	120.10

4.6 Summary and Key Findings

In this section, we explored an uncertainty-driven TL framework that was considered in order to tackle the unique challenges presented by heterogeneous edge environments. The fuzzy mechanism for labeling and distribution-based similarity detection improved the efficiency of model reuse across different edge devices. Outcomes include:

- **Faster adaption to different edge devices:** Pre-trained models are adapted with less effort across different tasks and hence, extensive multi-task retraining is not required.
- **Managing diverse as well as complex data:** The framework ensured the transfer of knowledge even in the presence of complex data sources by using multi-dimensional similarity based on distribution.
- **Less computation and communication:** The TL mechanism is well suited for resource constrained edge devices.

5 Task Management

5.1 Context and Motivation

Effective task management is a cornerstone part of distributed IoT and EC systems. A method for allocating tasks to nodes is considered optimal if it results in the best usage of the resources and meets the deadlines with the lowest amount of latency. Unfortunately, the ever changing IoT, EC environments' data sources and network conditions, combined with the limited resources, increases the difficulty of solving this particular problem. With the scaling of these systems, adaptive task management becomes increasingly important. Existing efforts in task management ignore data and concept drift, leading to sub-optimal resource usage and increased task failures. This effort aims to address this problem by developing an adaptive drift-driven mechanism that responds swiftly to changes in data and system conditions. Thus, enhancing system performance and resilience, providing a solution to the growing complexity task management problem in distributed systems.

5.2 Related Work

Out of the many challenges IoT and EC systems face, task management is arguably one of the most challenging ones. There has been some work related to task management in the context of Cloud systems [104–107]. However, there seems to be a recent focus [108–116] on moving away from centralized task management and towards a more decentralized approach where the management is handled at the edge infrastructure level. Doing this would enable data to be processed at a closer distance, thus lowering the response time. Given the limited computing power of edge devices, resource allocation strategies have to be very effective in order to maintain a reasonable level of service. [117] presents a full analysis of task management algorithms for Cloud and IoT infrastructures.

5.2.1 Static versus Dynamic Management

The methods of task management, in general, can be divided into two categories: static and dynamic strategies. Static task management [106], [118–120] works exclusively with prior knowledge of the state of the system, ignoring the real-time status of the nodes within the network. Over time, as data keeps accumulating, the statistical characteristics of each node change and, thus, the state of the system changes. Conversely, dynamic management [104], [111], [113], [114], [121–123] follows the current state of the system and, therefore, makes changes to task scheduling in real time. Dynamic management can be further subdivided into online mode, where tasks are processed upon arrival, and batch mode, in which tasks have to be accumulated within some predefined time period before they are acted upon [124].

5.2.2 Scheduling Algorithms and Optimization Models

In IoT and EC context, nodes work in extremely flexible conditions with unbalanced availabilities of resources and require constant status updates. To provide a near real-time response to an incoming task, the available resources of the target node need to be evaluated with high precision prior to the scheduling decision [116]. In order to achieve a balance between a range of different metrics which include task completion time, energy consumption, and load balancing, multi-objective optimization models are utilized to improve the overall scheduling efficiency.

5.2.3 Heuristic and Proactive Task Allocation

A three-stage heuristic approach for task allocation combined with an Integer Linear Programming (ILP) model is proposed in [125]. A task allocation management approach based on a genetic algorithm is introduced in [126], which attempts to optimize accuracy, energy consumption, and overall efficiency. The use of micro-data centers and intelligent gateways has been investigated to improve the efficiency of task allocation further in [127]. [128] uses a swarm based optimization approach for efficient task scheduling. In [129], a proactive task allocation framework is presented with a decision-making model to determine the optimal node assigned to each task. This framework employs unsupervised ML methods for local decision-making. Also, [130] provides a demand-driven task management model where a Long Short-Term Memory (LSTM) network is used to study task demand distribution and forecast workload trends.

5.2.4 Smart Gateways and Micro-Data Centers

Smart gateways and micro-data centers greatly improve the efficiency of task management [127]. In [128], an optimization strategy based on swarm intelligence is deployed to increase management efficiency. As explained in [131], user-defined tasks are subject to pre-processing at the gateway node, before tasks are divided into spatial and temporal sub-tasks. Those sub-tasks are either allocated to appropriate nodes or collectively processed by several nodes to improve system performance. A task allocation strategy, ETSI [131], has been introduced to assign tasks to the most relevant nodes in order to optimize execution workflows.

5.2.5 Existing literature gap

Even with new improvements in task management as well as scheduling algorithms, the available literature neglects to focus on the consequences of data and concept drift for automation execution in both static and dynamic management models. And, while models focus on optimization resources alongside execution time, energy usage, and load balancing, they fail to consider the ever changing set of data and the structural changes to its statistical properties over time that lead to sub-optimal task allocation and increased failure rates. Furthermore, insufficient amount of attention is dedicated to the correlation between tasks, which is vital for proper resource allocation and efficient scheduling. Considerable attention should be given to the design of adaptive mechanisms capable of responding to data and concept drift, while incorporating tasks correlation so that task scheduling is done efficiently and effectively in highly dynamic resource constrained environments like EC.

5.3 Problem Statement

Expanding on the previous formulations in Section 1.6, each node continuously acquires data from its connected IoT devices. The node n_i acts as a data aggregator, collecting and processing incoming information for further analysis. The reporting interval (td) is determined by the application's requirements. For time-sensitive applications, frequent updates may be necessary, although excessive reporting can significantly deplete the energy of IoT devices. At every reporting instance, n_i computes statistical aggregates such as the mean and variance for each feature within the incoming data. Consequently, each node maintains statistical records as two lists, m_i and v_i, where each entry corresponds to a specific reporting period. These lists continuously expand as new data arrive. The statistical aggregates for reporting period j are represented as $m_{ij} = (m_{ij_1}, m_{ij_2}, \ldots, m_{ij_d}), \quad v_{ij} = (v_{ij_1}, v_{ij_2}, \ldots, v_{ij_d})$, where m_{ij} is a d-dimensional vector storing mean values and v_{ij} contains variance values for each feature. For simplicity, the subscript i, which indicates the dataset index, is omitted in the subsequent description.

Each node is responsible for handling a queue q_i of computational tasks, where the queue length is denoted as lq_i. These tasks may involve operations such as data filtering, time-series analysis, and training or inference of ML models. Given that edge devices have limited processing capabilities compared to the Cloud infrastructure, it is crucial to manage workloads effectively to avoid system overload. Additionally, minimizing response time is critical to ensure efficient task execution. Each node is assigned a processing capacity P_i, while each task has a computational size sz $(sz_1, sz_2, \ldots, sz_{lq_i})$. The cumulative size of all tasks currently allocated to a node, referred to as the node task size (nts_i), quantifies the computational workload. Typical tasks include:

- **Analyzing time-series data** to identify patterns and trends in wind speed fluctuations.
- **ML model inference** using pre-trained DL models on newly received data.

Definition 14. *For a node n_i, its computational load is defined as:*

$$ld_i = \frac{nts_i}{P_i}. \tag{30}$$

Definition 15. *A task t_s with size sz_s can be assigned to a node n_i if:*

$$\frac{nts_i + sz_s}{P_i} \leq \zeta, \tag{31}$$

, where ζ represents a predefined threshold for load balancing.

In an IoT-EC setting, data distributions constantly evolve due to environmental factors and changes in user behavior. This causes fluctuations in the data collected by edge devices. Some examples include:

- **Data drift**: A humidity sensor registering variations in measurements throughout the day due to temperature changes.
- **Concept drift**: An ML model in a surveillance system showing reduced accuracy as a result of changing lighting conditions.

Such drifts affect data reliability, necessitating adaptive task scheduling strategies to maintain performance. To address these issues, the system may:

- Transfer tasks to a peer node with a dataset exhibiting similar distribution characteristics when compared to the local node.
- Retrain ML models to adjust to evolving data patterns.
- Postpone ML inference tasks until retraining is completed, particularly in cases of concept drift.

The proposed approach operates under the following assumptions:

- Each node n_i independently handles task execution and workload distribution.
- Tasks are scheduled without preemptive prioritization.
- Tasks operate independently without dependencies.
- Execution times vary based on the application and processing requirements.

5.4 Adaptive Task Management Approach

This section details our methodology on how to detect data related drifts in the EC environment as well as how to find peer nodes with similar dataset distributions. Also, we propose a task management mechanism that decides to suspend, offload, or update/retrain ML models based on detected drifts, dataset resemblance, and resource availability.

5.4.1 Data Drift

Data drift [132] is the process of data gradually altering in terms of statistical attributes, or distribution over a time. This is very common with sensor data and datasets used to train ML models. Drifting data can be extremely harmful to tasks' execution as well as the performance of applications on the edge. If not addressed, this will lead to lower accuracy of ML models and higher latencies in executing data processing tasks. For this reason, it is important to detect data drift as soon as possible and act accordingly.

To detect deviations in data behavior, statistical methods examine distribution properties and evolving patterns. Such methods are particularly effective for the observation of modifications in historical and live streaming data. The sliding window method [133] is a commonly used approach for drift detection, which is based on the changes in the mean and variance of a dataset. This method continuously tracks the statistical aggregates (mean and variance) of either an entire dataset or individual features within a fixed-size moving window of recent data. When the mean or variance within this window deviates beyond a predefined threshold, it signals a data drift.

For a window of size W, capturing the last W reporting periods, we define threshold values Δ_μ and Δ_{σ^2} to represent the tolerated deviation levels for mean and variance, respectively. These thresholds determine the acceptable range of change before drift is flagged and can vary based on the specific application. At the start of a monitoring phase, baseline values for mean and variance, denoted as μ_{base} and v_{base}, are established using the initial W reporting periods. As new data arrive, the oldest data points are discarded to maintain a constant window size. At each reporting interval, updated mean and variance values (μ_{new}, v_{new}) are computed and compared against the established baselines. The d-dimensional vectors μ_{base} and v_{base} are computed as follows:

$$\mu_{base_l} = \frac{\sum_{j=1}^{W} m_{ij_l}}{W}, \quad v_{base_l} = \frac{\sum_{j=1}^{W} v_{ij_l}}{W}, \quad l = 1, \ldots, d \tag{32}$$

Similarly, for the reporting period k, the updated values μ_{new} and v_{new} are computed as:

$$\mu_{new_l} = \frac{\sum_{j=k-W+1}^{k} m_{ij_l}}{W}, \quad v_{new_l} = \frac{\sum_{j=k-W+1}^{k} v_{ij_l}}{W}, \quad l = 1, \ldots, d \tag{33}$$

Whenever a data drift is detected, the baselines μ_{base} and v_{base} are updated with the newly computed values μ_{new} and v_{new}.

Definition 16. *A data drift for the l-th dimension is identified if the deviation between μ_{new} and μ_{base}, or between v_{new} and v_{base}, exceeds the respective predefined thresholds Δ_μ and Δ_{σ^2}:*

$$|\mu_{new_l} - \mu_{base_l}| \geq \Delta_\mu \vee |v_{new_l} - v_{base_l}| \geq \Delta_{\sigma^2} \tag{34}$$

Definition 17. *A Boolean function $\mathbb{I}(c)$ is defined to evaluate a given condition c:*

$$\mathbb{I}(c) = \begin{cases} 0, & \textit{if } c \textit{ is False} \\ 1, & \textit{if } c \textit{ is True} \end{cases} \tag{35}$$

Definition 18. *A node is considered to be experiencing data drift when the fraction of features that exhibit deviations beyond the defined thresholds ($\Delta_\mu, \Delta_{\sigma^2}$) surpasses a predefined parameter α:*

$$\frac{\sum_{l=1}^{d} \mathbb{I}(|\mu_{new_l} - \mu_{base_l}| \geq \Delta_\mu \vee |v_{new_l} - v_{base_l}| \geq \Delta_{\sigma^2})}{d} \geq \alpha \tag{36}$$

Algorithm 6 outlines our data drift detection process. Specifically, for each reporting interval, it calculates the updated statistical aggregates μ_{new} and v_{new} for all features and compares them against their respective baselines μ_{base} and v_{base}. If the deviation for any feature exceeds its defined threshold (Δ_μ or Δ_{σ^2}), data drift is flagged for that feature. If the proportion of affected features surpasses α, then data drift is considered to have occurred for node i during reporting period k. Algorithm 6 serves as a helper function within Algorithm 7.

Algorithm 6 Data drift detection

```
Input: $\mu_{base}, v_{base}, \Delta_\mu, \Delta_{\sigma^2}, m_i, v_i, \alpha, k, d, W$
Output: indicator for detection of data drift
sum = 0
for $l = 1, \ldots, d$ do
    calculate $\mu_{new_l}, v_{new_l}$ based on (4)
    if $|\mu_{new_l} - \mu_{base_l}| \geq \Delta_\mu \vee |v_{new_l} - v_{base_l}| \geq \Delta_{\sigma^2}$ then
        sum++, data drift for the $l$-th dimension
    end if
    if $\frac{sum}{d} \geq \alpha$ then
        Algorithm 6 ←1
    else
        Algorithm 6 ←0
    end if
end for
```

5.4.2 Concept Drift

Concept drift [134] depicts the phenomenon where the performance of the model declines over time with no observable change in the distribution of input data. This occurs when the relationships between input features and the target feature shift, while the features' statistical properties are stable in time. One reason for this is the temporal evolution in the target variable, which gradually changes over time. For example, when predicting stock prices, the economic factors affecting stock movements may change dynamically, impacting the models' predictive accuracy. Likewise, in the case of seasonal concept drift, some models may not generalize well to another season, even if the feature distribution remains the same. Applications driven by user behavior, like recommendation systems, e-commerce platforms, and even social media analytics, tend to undergo evolution in user preferences over a period of time which changes the underlying relationships on which the model was trained initially. Also, is it possible that the statistical distribution of input features remains the same, while each feature's significance in predicting the output changes. In such case, the predictive performance of the model will decrease. This problem is particularly important in EC environments where real time processing is expected.

In order to prevent performance degradation, concept drift detection and mitigation must be done in a timely manner. In an edge node, monitoring concept drift consists of keeping track of an ML model's accuracy over time and determining if there are any changes in the relationship between features and the target variable. The performance metric needed for assessment is model-specific: classification models are focused on accuracy and precision, regression models depend on RMSE and R-squared (R^2), while clustering models center their attention to Normalized Mutual Information (NMI). Monitoring drift can be accomplished through the use of mathematical or statistical techniques, among the most effective ones are control charts, specifically the Exponentially Weighted Moving Average (EWMA) [135]. This approach focuses on recent observations and so is capable of detecting shifting performance levels at an early stage. For node i in reporting period j, EWMA is calculated recursively as follows:

$$EWMA(i,j) = \lambda \cdot pm_{ij} + (1 - \lambda) \cdot EWMA(i,j-1) \tag{37}$$

where pm_{ij} represents the performance metric value for node i at reporting period j, and λ is the smoothing factor, typically set within the unity interval. A higher λ assigns greater weight to recent observations. The initial condition is $EWMA(i, 1) = pm_{i1}$. To detect performance drift, control limits must be established. These thresholds define significant deviations from expected performance and can be adjusted based on application sensitivity. One or two control limits can be defined, depending on the desired level of sensitivity. The first step in setting these limits is computing the variance of the EWMA values, which is also defined recursively:

$$Var(i, j) = \lambda \cdot (pm_{ij} - EWMA(i, j))^2 + (1 - \lambda) \cdot Var(i, j - 1) \tag{38}$$

where λ follows the same smoothing principle as in EWMA, and $Var(i, 1)$ is initialized at zero. The upper and lower control limits (ucl, lcl) for the EWMA chart, defining acceptable performance deviations for node i at reporting period j, are given by:

$$ucl(i, j) = EWMA(i, 1) + \delta \cdot \sqrt{Var(i, j)} \tag{39}$$

$$lcl(i, j) = EWMA(i, 1) - \delta \cdot \sqrt{Var(i, j)} \tag{40}$$

where δ is a multiplier that determines the sensitivity of drift detection.

Definition 19. *Given a node i and reporting period j, concept drift is identified if:*

$$EWMA(i, j) > ucl(i, j) \lor EWMA(i, j) < lcl(i, j) \tag{41}$$

When drift is detected, the reference value $EWMA(i, 1)$ is updated with the newly computed $EWMA(i, j)$ to ensure continuous adaptation.

5.4.3 Drift-Based Task management Mechanism

The execution of tasks within a certain node may be drastically affected by data and concept drift. A typical strategy used to cope with drift in ML models is to refresh or modify the models whenever a significant shift in the dataset occurs. Yet, constant re-training places a great burden on the system's processing capacity. This creates a trade-off between model effectiveness and resources, The balance between system resources and model execution accuracy needs to be adequately resolved. In circumstances where drift is noticed, and a node's computational load is considerably high, the ML model inference may be postponed. Nevertheless, for some crucial applications such as health monitoring [136], any significant change to the data calls for action which allows little time to postpone processing. To alleviate computational strain on an overloaded node, tasks related to data mining and analysis can either be postponed or offloaded to a suitable peer node. The decision on whether to delay or offload tasks depends on each task's execution tolerance and urgency. Tasks that do not require immediate execution may be deferred until the model is retrained, while others can be migrated to an alternative edge node. The selection of a peer node is based on two criteria, the similarity of its dataset to the pre-drift dataset of the initial node, and its available computational resources. This highlights the necessity of a mechanism capable of identifying nodes with comparable data distributions within the system.

In this Section, we discuss the practical application of the distribution-based similarity concepts introduced in Section 4.4.1 to the task management process, particularly in environments experiencing data and concept drift. Effective task management in EC environments requires not only the identification of nodes with available resources but also the selection of nodes where the data distribution is most similar to the current task's requirements. The multidimensional distribution-based similarity measures, as discussed in Section 4.4.1, are utilized here to identify the most suitable nodes for task offloading. By comparing the empirical CDFs of the current data batch with those of potential target nodes, we ensure that tasks are allocated to nodes where the data distribution is similar. This approach minimizes the impact of data and concept drift by continuously adapting the task management strategy to the evolving data landscape.

To efficiently manage tasks, the Kolmogorov-Smirnov test is employed to detect significant differences in distributions between the data associated with incoming tasks and the data stored on potential nodes. Nodes with the most similar data distributions are prioritized for task offloading, thereby optimizing task execution and maintaining model accuracy. The task management mechanism operates in real-time, continuously monitoring data distributions across nodes. When a new task arrives, the system quickly computes the similarity between the task's data and the data distributions on available nodes. If a significant drift is detected, the system dynamically reallocates tasks to nodes with more suitable data distributions. This ensures that tasks are processed by nodes that are most likely to yield accurate and efficient outcomes. For instance, consider a scenario where an edge node is responsible for processing tasks related to real-time analytics in a smart city environment. If a node detects a shift in traffic patterns (indicating concept drift), it can offload some tasks to another node with a data distribution more reflective of the new conditions. This proactive management reduces the likelihood of processing delays or inaccuracies, improving the overall performance of the system.

Algorithm 7 outlines our task management mechanism. Specifically, for each reporting period and every node, we first identify potential data drifts (dd) using Algorithm 6 and detect concept drifts (pd) through the procedure detailed in 5.4.2. If either data drift or concept drift is observed, we evaluate whether the node can accommodate the additional computational load introduced by model retraining. When a node does not have enough resources, we search for peer nodes with matching data distributions as outlined in 5.4.3. We also make sure that no node within the system surpasses ζ (computational load threshold) by distributing the workload to one or several of the determined peer nodes. Model retraining is then conducted to ensure optimal accuracy levels are continually maintained.

5.5 Experiments and Results

5.5.1 Configuration and Performance Metrics

We simulate the described EC scenario, consisting of ten (10) nodes, using Python. Our simulation framework consists of a collection of objects representing the available nodes. These objects implement functionalities that align with the execution of the previously described algorithms, governing the nodes' operations. All nodes are assumed to have identical computational resources, including processing capacity, memory, and network capabilities. To evaluate the performance of our DBTM mechanism, we conduct two experimental scenarios using the following performance metrics. For benchmarking, we compare our mechanism against the commonly used First Come First Served (FCFS) [137] and Longest Job First (LJF) [138] task management approaches within an EC context.

The first metric under examination is the load (ld) of each node throughout the execution process. More precisely, we focus on the maximum load experienced by a node during task execution. As noted earlier, a predefined threshold determines the acceptable load level based on the application re-

Algorithm 7 Drift-Based Task management

```
Input: $N$, $\Delta_\mu, \Delta_{\sigma^2}, d, W, td, sz_{rt}, \alpha, \beta, \gamma, \delta, \zeta$
Output: management of tasks
for every node $i = 1, \ldots, N$ do
    Initialize the lists $m_i, v_i, pm_i$
    $EWMA(i, 1) = pm_{i1}$
    $Var(i, 1) = 0$
    Calculate $\mu_{base}, v_{base}$ based on (3)
end for
for every period k do
    for every node $i = 1, \ldots, N$ do
        update $\mathbf{m_i}, \mathbf{v_i}, pm_{ij}$
        calculate $\mu_{new}, v_{new}$ based on (4)
        $dd \leftarrow$ Algorithm1( $\mu_{base}, v_{base}, \Delta_\mu, \Delta_{\sigma^2}, m_i, v_i, \alpha, k, d, W$)
        calculate $EWMA(i, j), Var(i, j)$ based on (8), (9)
        calculate $ucl(i, j), lcl(i, j)$ based on (10), (11)
        $pd \leftarrow 0$
        if $EWMA(i, j) > ucl(i, j) \lor EWMA(i, j) < lcl(i, j)$ then
            $pd \leftarrow 1$
        end if
        if $dd$ or $pd$ then
            if $\frac{nts_i + sz_{rt}}{P_i} > \zeta$ then
                identify similar nodes based on (13), (14)
                offload tasks until $\frac{nts_i + sz_{rt}}{P_i} \leq \zeta$
            end if
            retrain the model(s)
        end if
    end for
end for
```

quirements. When the load remains within this threshold, our mechanism effectively manages tasks without risking system failures. However, exceeding the threshold indicates inefficient task management, potentially leading to system failures where nodes cannot execute their assigned workloads. The second metric we analyze is the accuracy of ML models deployed in the nodes over time. Specifically, we monitor the minimum accuracy achieved by the models during execution. If accuracy remains stable, our mechanism successfully detects data drifts and retrains models accordingly. Conversely, if accuracy deteriorates, this suggests that our mechanism fails to adapt to data drifts by retraining models as needed.

Table 8 presents the parameters used in both experimental scenarios. Each node employs Support Vector Regression (SVR) models. SVM-based models have been widely adopted in concept drift detection research [139–142]. Regarding SVR hyperparameters, we set 'lib-linear' as the solver while leaving all other parameters at their default values. Additionally, the sliding window size W is set to 3, corresponding to three reporting periods/epochs.

Table 8: Parameter definitions used in the evaluation of DBTM

Symbol	Meaning
Δ_μ	Mean value threshold
Δ_{σ^2}	Variance value threshold
α	Data drift threshold
λ	Smoothing parameter
δ	Multiplier
β	Similarity threshold (dimensions)
γ	Similarity threshold
ζ	Load threshold

For the first experiment, we compare DBTM against FCFS and LJF using load and accuracy metrics. We generate synthetic datasets to simulate an EC system comprising ten (10) nodes. The dataset consists of three input features and a binary output (0/1). The model retraining size (sz_{rt}) is set to 0.5. We perform experiments under different parameter configurations (Cases 1-4), as listed in Table 9. The dataset dimensions are set to $d = 3$, and the load threshold is defined as $\zeta = 0.9$.

Table 9: DBTM configuration cases used in synthetic evaluation

	Δ_μ	Δ_{σ^2}	α	λ	δ	β	γ	ζ
Case 1	3	0.5	0.9	0.7	2	0.5	0.5	0.9
Case 2	3	0.5	0.9	0.6	1.5	0.8	0.8	0.9
Case 3	2.5	0.2	0.9	0.7	2	0.8	0.8	0.9
Case 4	2.5	0.2	0.9	0.6	1.5	0.5	0.5	0.9

For the second experiment, we compare DBTM with FCFS and LJF using load and accuracy metrics. This time, we utilize a real-world dataset[2] to assess model performance in a practical setting. The EC environment consists of nine (9) nodes, and the dataset is evenly split into nine portions to represent data distributed across the nodes. The dataset comprises nine (9) input features (id, date, day, period, nswprice, nswdemand, vicprice, vicdemand, transfer) and one output feature (class). The id, date, day, and period attributes are omitted. The model retraining size (sz_{rt}) is set to 0.6. Similar to the first experiment, we test different parameter configurations (Cases 5-8), as shown in Table 10. The dataset dimensions are set to $d = 5$, with a load threshold of $\zeta = 0.9$.

Table 10: DBTM configuration cases used in real-world evaluation

	Δ_μ	Δ_{σ^2}	α	λ	δ	β	γ	ζ
Case 5	0.2	0.1	0.9	0.8	2	0.5	0.5	0.9
Case 6	0.2	0.1	0.9	0.6	1.5	0.8	0.8	0.9
Case 7	0.4	0.2	0.9	0.8	2	0.5	0.5	0.9
Case 8	0.4	0.2	0.9	0.6	1.5	0.8	0.8	0.9

The datasets used in both experiments are described below:

- **Synthetic Dataset (Experiment 1)**: Designed to simulate an EC environment with ten nodes. Each dataset contains three input features and one binary output. The dataset is intentionally generated to exhibit data and concept drifts to evaluate the DBTM mechanism's effectiveness.

[2]https://www.kaggle.com/datasets/gauravduttakiit/electricity-prices-in-new-south-wales

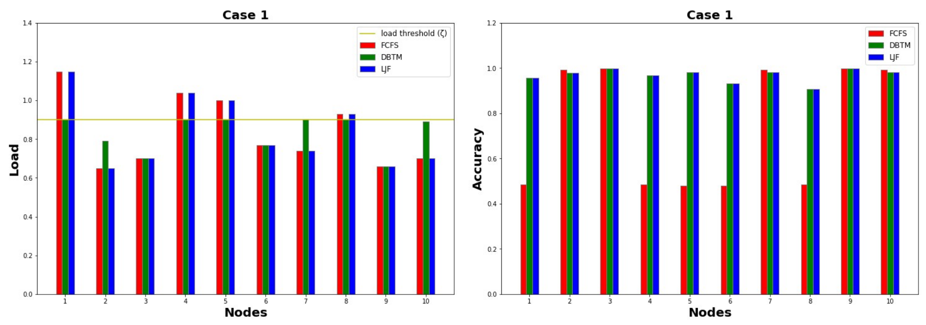


Figure 25: Load (ld) and accuracy comparisons (Case 1)

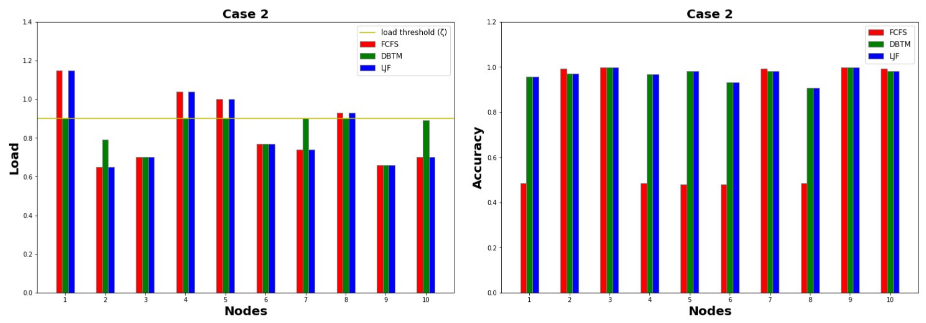


Figure 26: Load (ld) and accuracy comparisons (Case 2)

- **Electricity Pricing Dataset (Experiment 2)**: A real-world dataset consisting of nine input features and one output feature (class). It is divided into nine equal portions, simulating an EC environment with real data that may experience concept drift.

5.5.2 First Experimental Scenario

The following Figures illustrate the findings from the first experiment.

Figures 25, 26, 27, and 28 illustrate that DBTM successfully maintains node load at or below the threshold ζ, while simultaneously ensuring high accuracy levels for the ML models (ranging from 0.908 to 0.998). Conversely, FCFS and LJF fail to keep the load of multiple nodes (e.g., nodes 1, 4, 5, and 8) within the defined threshold ζ. Additionally, FCFS is unable to preserve accuracy levels, as nodes 1, 4, 5, 6, and 8 exhibit accuracy scores as low as 0.48. Meanwhile, LJF achieves the same accuracy results as DBTM, which is expected given that LJF prioritizes large tasks, with model retraining

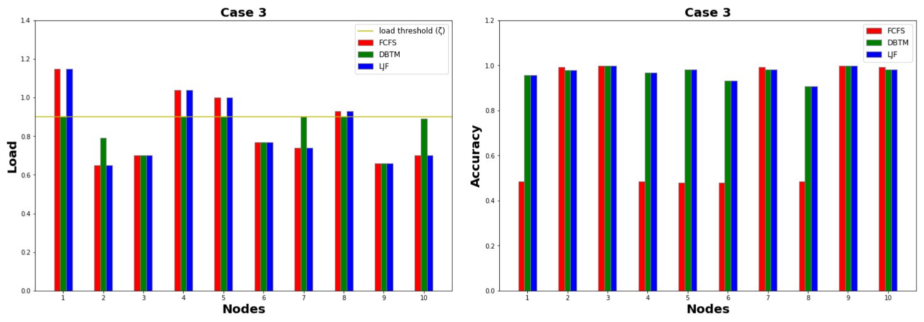


Figure 27: Load (ld) and accuracy comparisons (Case 3)

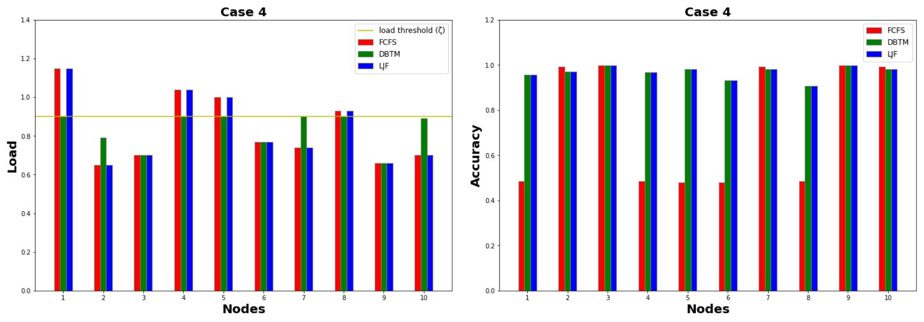


Figure 28: Load (ld) and accuracy comparisons (Case 4)

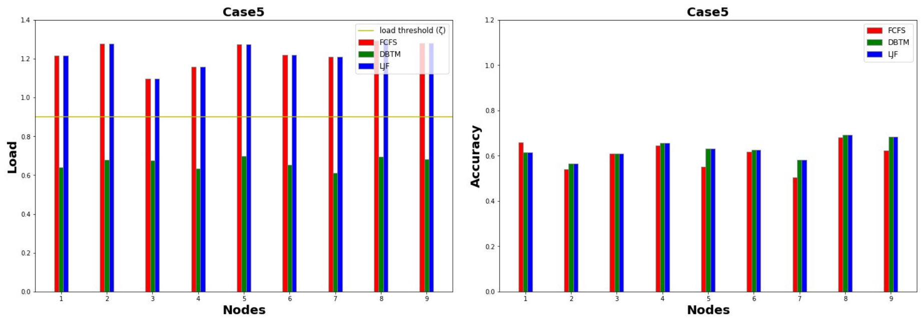


Figure 29: Load (ld) and accuracy comparisons (Case 5)

being the most computationally intensive task. As a result, LJF ensures that model updates occur first, preventing accuracy degradation. However, in most cases, LJF exhibits higher load levels compared to DBTM. Overall, in this experimental setting, DBTM significantly outperforms both FCFS and LJF, effectively balancing both load management and model accuracy.

5.5.3 Second Experimental Scenario

Figures 29, 30, 31, and 32 demonstrate that DBTM successfully regulates node load within the threshold ζ while maintaining model accuracy at levels comparable to or exceeding the other two methods. In contrast, both FCFS and LJF fail to control node load within the predefined threshold across all nodes. Additionally, FCFS is unable to sustain accuracy levels, as all nodes experience accuracy values lower than or equal to those achieved by the other methods. As observed in the first experimental scenario, LJF attains the same accuracy performance as DBTM. These findings highlight that DBTM effectively balances task-related computational load while ensuring high accuracy in knowledge production, offering a robust and efficient alternative to traditional task management approaches in EC environments.

5.6 Summary and Key Findings

Accurate and reliable performance is enhanced without being impacted by performance degradation through the use of a drift detection mechanism that ensures the model adapts to changes dynamically. Our mechanism is capable of detecting shifts in data distribution by monitoring changes in the mean and variance of the data with the sliding window approach. This means that

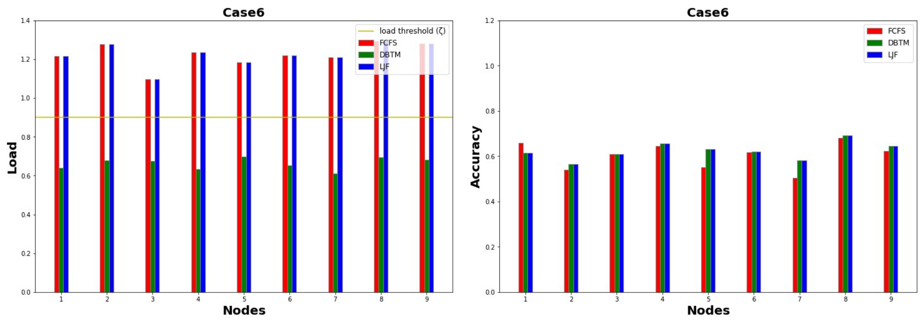


Figure 30: Load (ld) and accuracy comparisons (Case 6)

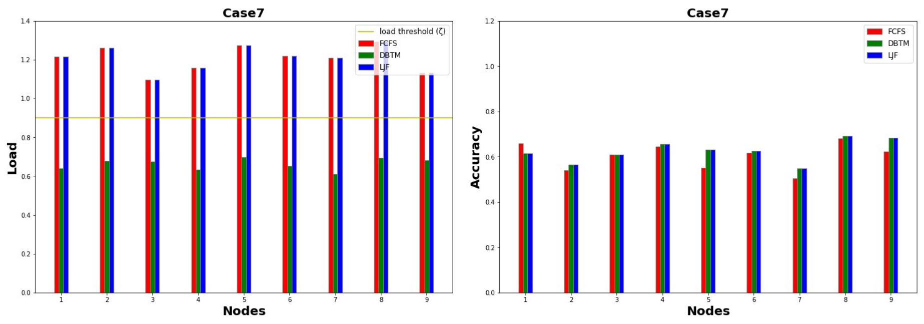


Figure 31: Load (ld) and accuracy comparisons (Case 7)

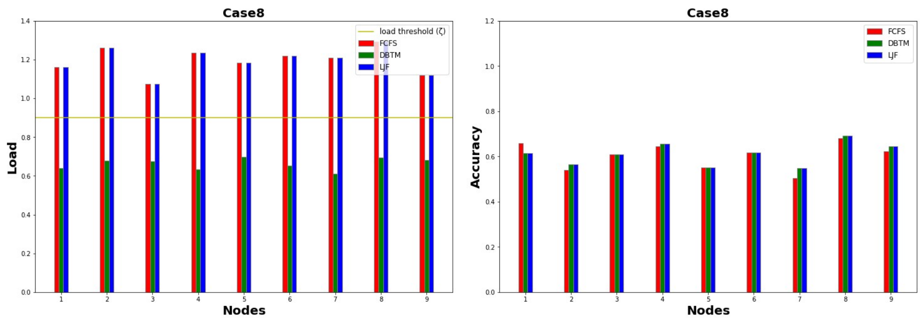


Figure 32: Load (ld) and accuracy comparisons (Case 8)

the models can be retrained in a timely manner to ensure that they remain accurate over time, even when the data being fed to the model continuously changes. The system keeps track of the model's performance using EWMA, and when certain minimum thresholds are breached, the model is retrained. These drifts indicate that feature-relationship with the target variable has changed. The drift detection mechanism is robust, and helps in ensuring optimal model performance and reliability under different conditions. This kind of reliability would not be possible by switching to a different ML algorithm lacking adaptiveness.

Our strategy utilizes the Kolmogorov-Smirnov test for identifying distribution-based similarity and EWMA monitoring for performance assessment. This combination enables effective offloading of tasks and retraining of models while sustaining system performance. In the previous Section we demonstrated our approach's efficacy in mitigating model performance deterioration alongside resource wastage through drift detection analysis across different contexts. The following items illustrate the effects on the predictive system:

- **Enhanced Model Versatility**: The system is able to achieve and sustain high predictive accuracy by monitoring and adapting to data and concept drifts. This type of flexibility is essential to dynamic systems where the most relevant data and their interrelations may shift unexpectedly.

- **Optimal Use of Resources**: Informed by drift detection, the task management mechanism is able to intelligently offload the tasks to the nodes with higher similarity in data distribution. This balances the computational workload and warrants that the tasks are processed by models relevant to the data, thereby optimizing system performance.

- **Less Severe Model Degradation**: Proactively retraining models in response to detected drifts reduces the performance degradation that comes from exposure to changing data. This approach ensures the model's accuracy and reliability.

- **Validation through Experiments**: The performed experiments indicate that the inclusion of these drift solutions within DBTM execution mechanism results in better performance than traditional task management systems (FCFS and LJF). In particular, DBTM demonstrated lower computational loads and higher accuracy across nodes in both synthetic and real-world datasets.

By integrating these advanced drift detection and management strategies, the predictive system sustains optimal performance and reliability, even in dynamically changing data environments. This evaluation highlights the significance and influence of effectively handling both data and concept drifts in contemporary predictive frameworks.

6 Task Scheduling

6.1 Context and Motivation

In EC systems, tasks must be scheduled across multiple devices to ensure effi- cient use of resources and timely execution. Scheduling tasks becomes increasingly complicated when taking into account different task requirements such as network limitations, data availability, or computation demands. Complexity also continues to grow alongside an increase in the number of tasks and edge devices. The success of real-time, distributed applications within edge environments heavily relies on effective scheduling of tasks. Many current methods of scheduling focus on the node capabilities and available tasks, but fail to consider their interrelationship which leads to reduced resource efficiency and increased latency. This research attempts to tackle the absence of intelligent scheduling mechanisms that consider node resources alongside task and data correlation for effective execution of tasks in distributed systems. Thus enhancing the performance and scalability of EC frameworks.

6.2 Related Work

Task offloading is becoming an increasingly researched topic, especially for tasks that require considerable computation and need to be sent to more powerful remote nodes for processing [143]. The process of deciding how and when to offload tasks is profoundly complex and affected by multiple factors [144]. This is particularly important in situations where a node has inadequate data or processing power in comparison to what the task's execution requires [145–147].

6.2.1 Optimization Models and Algorithms

ALTO is an adaptive learning-based algorithm that seeks to optimize task offloading by minimizing the average associated delay [148]. The Mobile Edge Computing (MEC) paradigm has recently emerged with distributed cloud computing resources accessible at the edge of the network, allowing for timely services and applications with minimal latency for users [149]. Offloading tasks to MEC has been shown to be an effective method in alleviating the computational load on mobile devices and reducing service latency, especially for high computation tasks [150]. In [150], an offloading algorithm is proposed which incorporates task dependencies, both temporally and across multiple tasks to minimize the amount of transmitted tasks within the network. Likewise, in [151], a multi-user offloading framework is developed which incorporates two different types of resource allocation problems as part of a cost minimization model. The authors present a heuristic solution that jointly optimizes the allocation of radio and computational resource distribution.

6.2.2 Multi-Agent Systems and Caching Strategies

The management of task offloading using multi-agent systems is examined in [152], where decision making is integrated across three interrelated tables to make the allocation process more efficent. The main considerations are task importance, network congestion, and resource availability. Offloading techniques may be aided with caching mechanisms as indicated in [153]. Through the storage and sharing of task information and resource requirements, nodes are able to fetch previously computed results, thus avoiding unnecessary computations and conserving energy.

6.2.3 Sum-Cost Delay Model

In [94], the task offloading issue is framed in the context of EC within a sum cost delay model. In this work, an optimal binary offloading decision is firstly formulated, and Reinforcement Learning (RL) is subsequently applied to obtain the final allocation policy.

6.2.4 Game Theory Models

In [154], task offloading is analyzed using game theory as a double auction game. The authors study the Bayes–Nash Equilibrium (BNE) in order to derive a suitable pricing policy. Moreover, a Stackelberg game is used to model the relationships between nodes involved in task trading with the aim of improving resource allocation related decision making. The proposed framework is particularly suited for blockchain-related applications.

6.2.5 Existing literature gap

Although task offloading and advanced management processes have been researched extensively, most available solutions still center around resource allocation optimization as well as minimizing execution delays without fully considering real time decision making complexities in EC systems. Most task offloading frameworks are frequently based on a set of policies or heuristics that do not respond to constantly changing system parameters like variable workload volumes, network traffic, and diverse available resources. Furthermore, while some studies consider multi-agent collaboration and caching strategies, they frequently treat tasks independently, overlooking inter-task dependencies that could influence the overall efficiency of task execution. This can lead to suboptimal task scheduling, where decisions are made based on individual tasks rather than holistic system performance. The relationships between tasks (where one task's execution impacts the execution efficiency of other tasks) is frequently overlooked, but could potentially be significant for resource optimization. Furthermore, most strategies based on game theory and optimization assume ideal conditions, or use a model that is too costly in terms of resources and time to be useful for EC. Though some approaches based on RL and auction theory have been developed for task offloading, their applicability in highly mobile and resource-limited situations is still a question.

This gap highlights the need for more adaptive task offloading frameworks that integrate real-time system monitoring, lightweight decision-making mechanisms, and task dependency awareness to enhance offloading efficiency in EC. Our work addresses this by proposing a dynamic, context-aware offloading strategy that optimally distributes tasks across nodes while considering real-time system constraints and task interdependencies

6.3 Problem Statement

Building upon the formulations presented in Section 1.6, each node n_i calculates statistical features such as mean, variance and range of values for each feature of the particular local dataset. These features are summarized by d-dimensional vectors: (Node Feature Variance - NFV, Node Feature Mean - NFM, Node Range of Values - NROV), where:

$$NFV_i = [NFV_i[1], NFV_i[2], \ldots, NFV_i[d]]$$

$$NFM_i = [NFM_i[1], NFM_i[2], \ldots, NFM_i[d]]$$
$$NROV_i = [NROV_i[1], NROV_i[2], \ldots, NROV_i[d]]$$

for node i. Here, each $NFV_i[j], NFM_i[j], NROV_i[j]$ corresponds to the variance, mean, and range of values for the j-th feature within n_i's dataset. This cluster is employed for performing user defined activities like executing SQL queries, undertaking ML model training and carrying out other data processing activities. Every task has accompanying data requirements to its execution and may have other limitations like dependence on other tasks, execution order or preference, and so forth. Nevertheless, these dependencies and preferences pose a challenge to manage, which is beyond the coverage of the present work and is reserved for the subsequent research endeavors. Each node n_i has a processing capability denoted by P_i and a queue q_i where incoming tasks are placed for execution, represented as:

$$q_i = [t_1, t_2, \ldots, t_s]$$

where s is the maximum queue size.

Examples of such tasks include:

- An SQL query retrieving data records where, for instance, humidity values fall within the range of 20% to 50%.
- An ML task training a regression model on local data where the pressure variance ranges from 1 to 3 and the mean wind speed lies between 20 km/h and 50 km/h.

Fundamentally, any computational task involving data processing can be treated as a resource-consuming operation on the node. The requirements of each task are represented by d-dimensional vectors: (task feature variance - tfv, task feature mean - tfm, task range of values - trov), where:

$$tfv_j = [tfv_j[1], tfv_j[2], \ldots, tfv_j[d]]$$

$$tfm_j = [tfm_j[1], tfm_j[2], \ldots, tfm_j[d]]$$
$$trov_j = [trov_j[1], trov_j[2], \ldots, trov_j[d]]$$

for task j. Each element in these vectors represents an interval of desired values for a particular feature. If no specific requirement exists for a given feature, the corresponding value in the vector is set to zero. Additionally, each task has a size sz:

$$sz = [sz_1, sz_2, \ldots, sz_s]$$

which quantifies its computational demand on the system.

Tasks are scheduled for execution at predefined scheduling periods, the duration of which varies depending on the application requirements. Real-time applications, for instance, may necessitate shorter scheduling periods to ensure responsiveness. If a task arrives during a scheduling period, it must wait until the next scheduling round before being considered for execution. At the start of each scheduling period, nodes update their statistical data and accumulated task size: node task size (nts), which is computed as the sum of all pending and executing task sizes within the node. This updated information is then sent to the scheduler.

In instances where a node is fully engaged with execution, it may pause other activities temporarily in order to calculate relevant statistical information. The scheduler receives the information along with the requirements of the tasks and then determines which nodes can perform each task. The next step is the clustering of similar tasks which is then followed by the final scheduling. The following assumptions are considered in our approach:

The following assumptions are considered in our approach:

- Each node n_i independently executes tasks and manages its workload.
- Execution of tasks is based on a non-preemptive scheduling policy, i.e. tasks do not have any predefined order of priorities.
- Tasks are self sufficient and do not rely on any other tasks for completion.

6.4 Task scheduling Approach

In this section, we outline the methodology for identifying suitable nodes for task execution and establishing task correlations. By leveraging these insights, we construct an optimized scheduling mechanism that takes into account the computational capacity of each node.

6.4.1 Detecting the Suitability of Nodes

For each task, the scheduler determines the appropriate nodes for execution by analyzing their statistical data and represents their availability using an N-dimensional vector:

$$su_j = [su_j[1], su_j[2], \ldots, su_j[N]]$$

where $su_j[i]$ is set to 1 if node n_i qualifies for executing task j, otherwise, it is set to 0.

Definition 20. *A node n_i is deemed suitable for executing task t_j if the statistical attributes $NFV_i, NFM_i, NROV_i$ align with the task's requirements $tfv_j, tfm_j, trov_j$, ensuring that the intersection of all non-zero values in the pairs $(tfv_j - NFV_i)$, $(tfm_j - NFM_i)$, $(trov_j - NROV_i)$ is not an empty set.*

This definition serves as a criterion for determining whether a node's dataset meets the requirements of a specific task at the feature level. The suitability evaluation function, applied to each feature dimension, is defined as follows:

$$s(j,i,l) = \begin{cases} 1, & \text{if } (tfv_j[l] \cap NFV_i[l] \neq \emptyset \vee tfv_j[l] = 0) \\ & \wedge(tfm_j[l] \cap NFM_i[l] \neq \emptyset \vee tfm_j[l] = 0) \\ & \wedge(trov_j[l] \cap NROV_i[l] \neq \emptyset \vee trov_j[l] = 0) \\ 0, & \text{otherwise} \end{cases} \tag{42}$$

The overall suitability vector su_j is then computed by aggregating the feature-wise suitability indicators:

Definition 21. *Given a task t_j with specified requirements $tfv_j, tfm_j, trov_j$ and a node n_i characterized by $NFV_i, NFM_i, NROV_i$, the suitability vector is determined as:*

$$su_j[i] = \begin{cases} 1, & \textit{if } \sum_{l=1}^{d} s(j,i,l) = d \\ 0, & \textit{otherwise} \end{cases}, i = 1, 2, \ldots, N$$

This function assesses whether a node's dataset meets the task's requirements across all feature dimensions.

We assume that task requirements for each feature are random variables following a Bernoulli distribution, where an existing requirement is assigned probability p and the absence of a requirement is assigned probability $1 - p$. In scenarios where dataset access is restricted, as in federated learning, we model statistical attributes as independent normal distributions:

$$NFV_i[l] \sim \mathcal{N}(\mu_1 i[l], \sigma_1 i[l]^2), \quad NFM_i[l] \sim \mathcal{N}(\mu_2 i[l], \sigma_2 i[l]^2)$$

$$NROV_i[l][1] \sim \mathcal{N}(\mu_3 i[l], \sigma_3 i[l]^2), \quad NROV_i[l][2] \sim \mathcal{N}(\mu_4 i[l], \sigma_4 i[l]^2)$$

Similarly, task requirements are defined as intervals $tfv_j[l] = [c, d]$, $tfm_j[l] = [e, f]$, $trov_j[l] = [g, h]$.

The expected value computations for feature-wise suitability and the overall suitability vector are outlined in Lemmas 1 and 2. To ensure clarity and completeness, supplementary calculations and detailed proofs are provided in the Appendix Section 8.

Lemma 1. *The expected suitability indicator for a feature is given by:*

$$\begin{aligned} \mathbf{E}(s(j,i,l)) = {} & P[tfv_j[l] \cap NFV_i[l] \neq \emptyset \vee tfv_j[l] = 0] \\ & \cdot P[tfm_j[l] \cap NFM_i[l] \neq \emptyset \vee tfm_j[l] = 0] \\ & \cdot P[trov_j[l] \cap NROV_i[l] \neq \emptyset \vee trov_j[l] = 0] \end{aligned} \tag{43}$$

Lemma 2. *The expected value of the overall suitability vector is given by:*

$$\mathbf{E}(su_j[i]) = \prod_{l=1}^{d} \mathbf{E}(s(j,i,l)) \tag{44}$$

Definition 22. *A node n_i is deemed suitable for executing task t_j if:*

$$\mathbf{E}(su_j[i]) \geq \gamma \tag{45}$$

where γ is a predefined threshold that determines suitability.

This formulation allows different applications to define varying tolerance levels for task execution based on their accuracy and performance requirements. The procedure for determining suitable nodes for all tasks is summarized in Algorithm 8. The input consists of d-dimensional vectors $tfv_j, tfm_j, trov_j$ for all tasks $(j = 1, \ldots, k)$ and the statistical information

$NFV_i, NFM_i, NROV_i$ for all nodes ($i = 1, \ldots, N$). Each statistical parameter must include the respective mean and variance for accurate computations. For each dimension of the task requirements, we compute the expected suitability indicator using Lemma 1 and the overall suitability using Lemma 2. The suitability of nodes is determined based on these calculations and compared against a predefined threshold γ. The final output is a $k \times N$ matrix, where each element represents the suitability vector for a given task and node.

Algorithm 8 Suitability

```
Input: t=($t_1, t_2, \ldots, t_k$), $trov_j, tfv_j, tfm_j, NROV_i, NFV_i, NFM_i, r, \gamma$,
mean and variance for every random variable
Output: su
for j=1, j≤ k, j++ do
    for i=1, i≤ N, i++ do
        $P = 1$
        for l=1, l≤ $d$, l++ do
            calculate $\mathbf{E}(s(j, i, l))$ based on lemma 1
            $P = P * \mathbf{E}(s(j, i, l))$
        end for
        if $P \geq \gamma$ then su[j][i] = 1
        else su[j][i] = 0
        end if
    end for
end for
```

6.4.2 Task Requirement Overlap and Correlation

We examine the scenario in which multiple tasks require access to similar datasets. Consider, for example, an SQL query that extracts temperature values within the range of 0 to 30 degrees Celsius, and a separate regression model that is tested using temperature values spanning from 10 to 20 degrees Celsius. Furthermore, some tasks may require identical datasets, such as training and testing an ML algorithm with temperature data that falls within the range of [20,30] degrees Celsius.

Tasks that exhibit a significant degree of similarity in data-related requirements can be grouped together based on this similarity. Scheduling correlated tasks on the same node enhances efficiency by minimizing redundant disk accesses. When two tasks require identical datasets, the results computed for the first task may be leveraged for the second, effectively reducing the overall system's computational burden. In cases where tasks share overlapping data related demands, the intermediate results from one task can be utilized to reduce the computation needed for the second. Consequently, the execution time for the second task will be shorter than if task correlation was not considered. Additionally, executing correlated tasks on the same node can improve TL capabilities, enabling knowledge transfer from one task to another. This facilitates a more efficient learning process by leveraging knowledge obtained from a previously completed task to enhance the performance of a different but related task.

Definition 23. *Given two intervals $int_1 = [a, b]$ and $int_2 = [c, d]$ with $a < b$ and $c < d$, the overlap between them is defined as:*

$$ov(int_1, int_2) = \frac{\max(0, \min(b, d) - \max(a, c))}{b - a} \tag{46}$$

The function $ov(int_1, int_2)$ yields a value in the range $[0, 1]$, representing the percentage of overlap between the two intervals, normalized by the length of the first interval.

Definition 24. *Given two tasks t_q and t_w, each with d feature dimensions and respective requirement vectors $v_q = (vq_1, vq_2, \ldots, vq_d)$ and $v_w = (vw_1, vw_2, \ldots, vw_d)$, the correlation between the two tasks is calculated as:*

$$corr(v_q, v_w) = \frac{\sum_{l=1}^{d} \frac{ov(vq_l, vw_l) + ov(vw_l, vq_l)}{2}}{d} \tag{47}$$

This measures the degree of similarity between the requirement vectors across all feature dimensions.

Definition 25. *Two requirement vectors v_q and v_w are classified as correlated if:*

$$corr(v_q, v_w) \geq \theta \tag{48}$$

where θ is a predefined threshold. Otherwise, they are considered non-correlated.

Algorithm 9 details the procedure for computing task correlation, which serves as the algorithm's output. The input consists of two requirement vectors. The correlation computation proceeds as follows: first, the overlap is determined for each feature dimension. If neither task has a specified value for a particular feature, the overlap is set to zero. In cases where both tasks specify values for the feature, the overlap is computed using Equation (46). Finally, the correlations are aggregated across all feature dimensions, and the overall correlation score is determined using Equation (47).

Algorithm 9 Detection of Correlation

```
Input: $v_q, v_w, d$
Output: $corr(v_q, v_w)$
for l=1, l≤ $d$, i++ do
    if $v_q[l] \neq 0$ & $v_w][l] \neq 0$ then
        calculate $ov(v_q[l],v_w[l])$, $ov(v_w[l],v_q[l])$ based on (1)
    else
        $ov(v_q[l],v_w[l])=0$, $ov(v_w[l],v_q[l])=0$
    end if
end for
calculate corr($v_q$,$v_w$) based on (2)
```

Finally, two tasks t_q and t_w are classified as correlated if their respective r-dimensional requirement vectors $tfv_q, tfm_q, trov_q$ for t_q and $tfv_w, tfm_w, trov_w$ for t_w satisfy the following condition:

$$corr(tfv_q, tfv_w) \geq \theta \wedge corr(tfm_q, tfm_w) \geq \theta \wedge corr(trov_q, trov_w) \geq \theta$$

where θ is a predefined correlation threshold.

Algorithm 10 is employed to identify and group correlated tasks, producing an output vector $cg = (cg_1, cg_2, \ldots, cg_k)$. This k-dimensional vector represents the assigned correlation group for each task, where cg_i takes values in the range $(1, l)$, with l denoting the total number of identified correlated groups. The procedure begins by initializing the first group with the first task and then identifying all tasks correlated with it using Algorithm 9. These tasks are then assigned to the same group. The process continues by selecting the next ungrouped task, assigning it to a new group, and repeating the correlation detection until every task is categorized into a group.

Algorithm 10 Correlated groups

```
Input: t=($t_1, t_2, \ldots, t_k$), trov, tfv, tfm, $\theta$
Output: cg, $l$
for h=1, h$\leq$ k, h++ do
    cg[h] = 0
end for
current=1
while 0 in cg do
    for h=1, h$\leq$ k, h++ do
        if cg[h]=0 then
            cg[h] = current
            break
        end if
    end for
    for d=1, d$\leq$ k, d++ do
        if cg[d]=0 then a = corr(trov[h],trov[d]), b = corr(tfv[h],tfv[d]), c
= corr(tfm[h],tfm[d])
            if ($a \geq \theta$ & $b \geq \theta$ & $c \geq \theta$) then
                cg[d] = current
            end if
        end if
    end for
    current++
end while
$l$ = current -1
```

6.4.3 Task Scheduling Mechanism

During each scheduling period, a queue of k tasks $(t_1, t_2, \dots, t_k)$ is maintained, with each task characterized by its requirements $(tfv_1, \dots, tfv_k)$, $(tfm_1, \dots, tfm_k)$, $(trov_1, \dots, trov_k)$ and the respective computational size $(sz_1, sz_2, \dots, sz_k)$. Let l denote the number of correlated task groups. For each correlated group, the intersection of the su_j vectors determines the set of 'suitable' nodes capable of executing the entire group. The intersected vector for each correlated group is given as:

$$cgint_j = (cgint_j[1], cgint_j[2], \dots, cgint_j[N]), \quad j = 1, 2, \dots, l$$

where $cgint_j[i] = 1$ if node n_i is suitable for executing the j-th correlated group. The complete set of intersected vectors is represented by the $l \times N$ matrix $cgint = (cgint_1, cgint_2, \dots, cgint_l)$.

From this matrix, we extract:

- The vector $op = (op_1, op_2, \dots, op_l)$, where op_j represents the number of suitable nodes (*options*) for the j-th correlated group.
- The vector $ncg = (ncg_1, ncg_2, \dots, ncg_N)$, where each ncg_i represents the number of correlated groups associated with node n_i.

Given that each node possesses a processing capacity P_i and has accumulated a task load nts_i from prior scheduling periods, the current load of each node can be computed accordingly. Refer to Definitions 14 and 15 for formal descriptions of computational load and task execution criteria, as these principles apply consistently throughout the document. The final task scheduling is executed by assigning correlated tasks to suitable nodes while ensuring that each node's load remains within allowable limits. The scheduling methodology comprises the following steps: 1) Determine the node which has the least number of correlated groups (ignoring zero values); 2) From those groups, determine which one has the smallest available count of node options (once more, ignoring zero values); 3) Begin by assigning the highest possible task size in attempts to maximize the number of tasks assigned to a node while ensuring the node load remains constantly under the defined threshold; 4) Update the correlation groups, options, intersected vectors, and number of correlated groups (cg, op, cgint, ncg); 5) Repeat these steps until one of the following outcomes is reached: The op vector is all zeros, indicating no further scheduling based on correlation is possible. The ncg vector is all zeros indicating nodes have no further correlated tasks they are able to execute. This procedure is detailed in Algorithm 11. Once correlated tasks have been scheduled, the remaining tasks (those not assigned within correlated groups) are scheduled separately based on their original suitability vectors (su), prior to intersection. This process is outlined in Algorithm 12, where tasks are allocated to available nodes without consideration for correlation.

In the case of allocation for unscheduled tasks, the process focuses on assigning the largest and the smallest computing sizes to the node that first available node that is suitable. This max-min strategy mitigates the phenomenon of executing large tasks only, or executing small tasks only, which can cause starvation of the tasks and can increase the response times. It is important to note that our scheduling mechanism does not maintain a detailed record of all computations performed by individual tasks. This approach would require a significant amount of communication overhead and possible data relocation costs. Instead, tasks are grouped based on correlation and allocated to nodes capable of executing them so that reuse of computational work is feasible. Also, our approach does not impose strong limitations on the evaluation time for the correlated groups. Instead, a node keeps the results of all the computations related to a certain task until all the tasks belonging to the corresponding correlated group are completed.

6.5 Experiments and Results

We evaluate the performance of our Correlation Adaptive Task Scheduling (CATS) mechanism through two experimental scenarios, using three key performance metrics. Our approach is compared against widely used task scheduling algorithms in EC, including First Come First Served (FCFS) [137], MAX-MIN [155], Longest Job First (LJF) [138], and Shortest Job First (SJF) [156]. Initially, we assess the *percentage of task failures* (ptf), which indicates how often tasks are assigned to nodes that cannot execute them. A task is considered a failure when it must be rescheduled due to an unsuitable allocation. The ptf metric is defined as:

$$ptf = \frac{\text{Number of task failures}}{\text{Total number of tasks}}$$

When $ptf \to 0$, tasks are successfully assigned to suitable nodes, while $ptf \to 100$ indicates frequent task failures.

Algorithm 11 Task scheduling

```
Input: t=$(t_1, t_2, \ldots, t_k)$, sz=$(sz_1, sz_2, \ldots, sz_k)$, cg=$(cg_1, cg_2, \ldots, cg_k)$, $P = (P_1, P_2, \ldots, P_N)$, nts=$(nts_1, nts_2, \ldots, nts_k)$,su=$(su_1, su_2, \ldots, su_k)$ ,$\beta$
Output: Scheduling of tasks to the suitable nodes based on correlation
for $j = 1, j \leq l, j++$ do
    cgint[j] = intersection of all $su_i$ vectors with $cg[i] == j$
end for
for $j = 1, j \leq l, j++$ do
    op[j]=sum(cgint[j])
end for
for $a = 1, a \leq N, a++$ do
    ncg[a]= 0
    for $j = 1, j \leq l, j++$ do
        if $cgint[j][a] == 1$ then
            ncg[a]++
        end if
    end for
end for
$count_1 = sum(op)$
$count_2 = sum(ncg)$
while $count_1 \neq 0$ & $count_2 \neq 0$ do
    find the index j of min (non-zero) value in ncg
    for $(j = 1,\ i \leq l,\ j++)$ do
        if $cgint_j[i] == 1$ then
            from all related correlated groups to the node i find the index g
of
            the one that has minimum (non-zero) options
        end if
    end for
    for $(h = 1,\ h \leq k,\ h++)$ do
        if $cg[h] == g$ then
            from all tasks in the g correlated group find the index f of the
one
            with maximum size
            if $(nts[i] + sz[f])/P[i] \leq \beta$ then
                assign task f to node j, $cg[h] == 0$
                $nts[i] = nts[i] + sz[f]$
            end if
            from all tasks in the g correlated group find the index z of the
one
            with minimum size
            if $(nts[i] + sz[z])/P[i] \leq \beta$ then
                assign task f to node j, $cg[h] == 0$
                $nts[i] = nts[i] + sz[z]$
            end if
        end if
```

Algorithm 12 Remaining tasks scheduling

```
Input: t=$(t_1, t_2, \ldots, t_k)$, sz=$(sz_1, sz_2, \ldots, sz_k)$, cg=$(cg_1, cg_2, \ldots, cg_k)$, $P = (P_1, P_2, \ldots, P_N)$, nts=$(nts_1, nts_2, \ldots, nts_k)$,su=$(su_1, su_2, \ldots, su_k)$ ,$\beta$
Output: Scheduling of the remaining tasks to the suitable nodes
______________________________________
$count_1 = sum(cg)$
$count_2 = 0$
while $count_2 \neq count_1$ & $count_1 \neq 0$ do
    for $(e = 1,\ e \leq k,\ e++)$ do
        if cg[e]$\neq$ 0 then
            from all unscheduled tasks find the index d of the one with
            maximum size
            for $i = 1, i \leq N, i++$ do
                if $su[d][i] == 1$ & $(nts[i] + sz[d])/P[i] \leq \beta$ then
                    assign task d to node i
                    $count_1$-=1
                    $cg[e] == 0$
                    $nts[i] = nts[i] + sz[d]$
                    break
                end if
            end for
            from all unscheduled tasks find the index c of the one with
            minimum size
            for $i = 1, i \leq N, i++$ do
                if $su[c][i] == 1$ & $(nts[i] + sz[c])/P[i] \leq \beta$ then
                    assign task d to node i
                    $count_1$-=1
                    $cg[e] == 0$
                    $nts[i] = nts[i] + sz[c]$
                    break
                end if
            end for
        end if
    end for
    $count_2 += 1$
end while
```

We also evaluate the *percentage of tasks scheduled based on correlation* (*pbc*), which measures our mechanism's efficiency in grouping correlated tasks. The *pbc* metric is defined as:

$$pbc = \frac{\text{Number of tasks scheduled based on correlation}}{\text{Total number of tasks}}$$

A value of $pbc \to 100$ implies that most correlated tasks are allocated to the same node, reducing redundant processing. Conversely, when $pbc \to 0$, task correlation is not leveraged for scheduling.

The third metric we examine is the *load of the system* (*load*), which is defined as:

$$load = \frac{\text{Total scheduled task size}}{\text{Total computational capacity of the system}}$$

By setting a threshold β for each node and the overall system, we ensure that task allocation does not exceed processing limits. Higher load values, while remaining below the threshold, indicate that more tasks are efficiently scheduled. Our simulation is implemented in Python and models an EC environment where each node periodically updates the scheduler with its current load and dataset statistics (mean, variance, range of values). These updates are minimal in terms of communication overhead. Additionally, execution time for each task is not explicitly considered in the evaluation. The primary goal is to group correlated tasks onto the same node to maximize computational reuse, striking a balance between increased energy consumption for some nodes and reduced overall energy demand across the system.

6.5.1 First Experimental Scenario (Setting)

In the first experiment, we compare CATS against FCFS and MAX-MIN using the ptf and *pbc* metrics. The environment consists of ten (10) edge nodes processing synthetic data. We vary the number of tasks across different trials: $k \in \{10, 50, 100\}$. Task sizes (sz) range from 1 to 5, and node computational capacity (P_i) varies between 50 and 100. The dataset dimensions are set to $r = 3$, and the load threshold is $\beta = 0.9$. Task requirements exist with probability $p \in \{0.5, 0.8\}$, while suitability (γ) and correlation (θ) thresholds are configured as follows:

$$\gamma \in \{0.5, 0.8\}, \quad \theta \in \{0.5, 0.8\}$$

The FCFS and MAX-MIN scheduling algorithms do not use these parameters, as they are specific to CATS.

6.5.2 Second Experimental Scenario (Setting)

In the second experiment, we compare CATS against LJF and SJF using the *load* and ptf metrics. Here, we use a real dataset[3] to assess our model's effectiveness in a practical setting. The dataset includes six input features: transaction date, house age, distance to the nearest MRT station, number of convenience stores in the living circle on foot, latitude, and longitude. The transaction date, latitude, and longitude are excluded from our experiments. The objective is to predict property values using a Linear Regression model trained on data grouped by age categories (e.g., [0-5], [6-15]).

The experimental setup consists of ten (10) edge nodes, each managing a portion of the dataset. We vary the number of tasks: $k \in \{10, 50, 100\}$. Task sizes (sz) range from 1 to 5, and node computational capacity (P_i) varies between 50 and 100. The dataset dimensions are set to $r = 4$, and the load threshold remains $\beta = 0.9$. Since all tasks in this scenario require data from a specific age group, the requirement probability is set to $p = 1$. The suitability and correlation thresholds are:

$$\gamma \in \{0.7, 0.9\}, \quad \theta \in \{0.7, 0.9\}$$

These parameters are not utilized in the LJF and SJF scheduling models.

6.5.3 Experimental Setup Summary

Table 11 provides a summary of the parameter values used in both experimental scenarios.

6.5.4 First Experimental Scenario (Performance Assessment)

The outcomes of our initial experimental setup are provided in the figures below along with the results for the chosen parameter values and task quantity. Our model achieves a task failure rate of zero, as shown in Figures 33, 34, which illustrates the scheduling of all tasks based on their correlation. Conversely, FCFS and MAX-MIN methodologies' failure rates hover around

[3]https://archive.ics.uci.edu/ml/datasets/Real+estate+valuation+data+set

Table 11: Unified parameter configuration across all evaluation scenarios for CATS

Parameter	Description	Value(s)
r	Number of dataset dimensions	3 (Experiment 1), 4 (Experiment 2)
p	Probability of task requirements	0.5/0.8 (Experiment 1), 1 (Experiment 2)
γ	Suitability threshold	0.5/0.8 (Experiment 1), 0.7/0.9 (Experiment 2)
θ	Correlation threshold	0.5/0.8 (Experiment 1), 0.7/0.9 (Experiment 2)
β	Load threshold	0.9
N	Number of edge nodes	10
P_i	Computational capacity per node	50-100
k	Total number of tasks	{10, 50, 100}
sz	Task size range	1-5

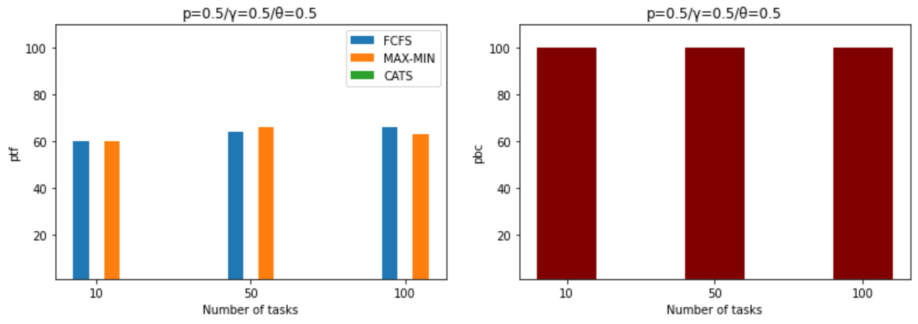


Figure 33: Comparison of ptf and evaluation of pbc ($p = 0.5, \gamma = 0.5, \theta = 0.5$)

60%, which indicates a lack of competent task allocation. The observed performance gap showcases the effectiveness of our method, which intelligently clusters correlated tasks and assigns them to appropriate nodes, yielding the best results for this scenario.

Figures 35 and 36 demonstrate that our model maintains a task failure rate between 17% and 30%, while scheduling tasks based on correlation in the range of 60% to 80%. Conversely, FCFS and MAX-MIN show a failure rate close to 80%, reinforcing their inefficiency in handling tasks. Notably, increasing γ leads to higher failure rates in our approach, as stricter suitability requirements limit the number of eligible nodes.

Figures 37 and 38 present results for an increased p value ($p = 0.8$). Once again, our model achieves zero task failures and schedules all tasks based on correlation. In contrast, FCFS and MAX-MIN continue to exhibit a failure

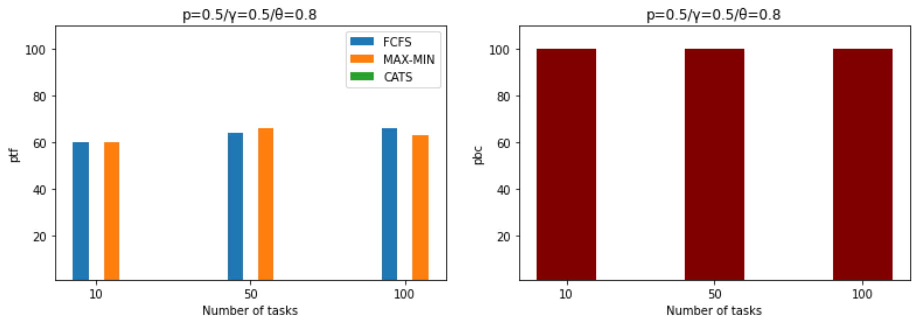


Figure 34: Comparison of ptf and evaluation of pbc $(p = 0.5, \gamma = 0.5, \theta = 0.8)$

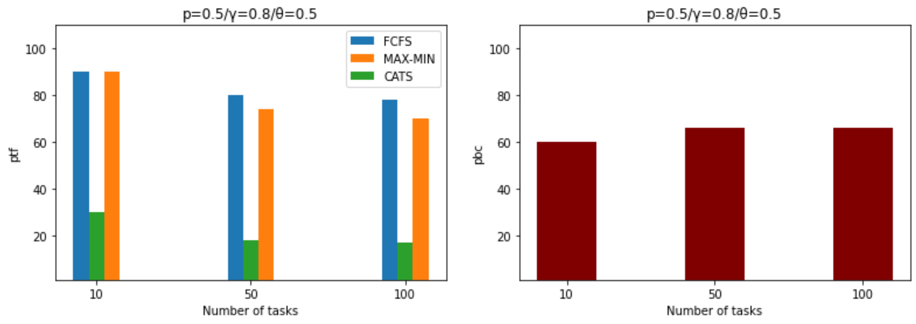


Figure 35: Comparison of ptf and evaluation of pbc $(p = 0.5, \gamma = 0.8, \theta = 0.5)$

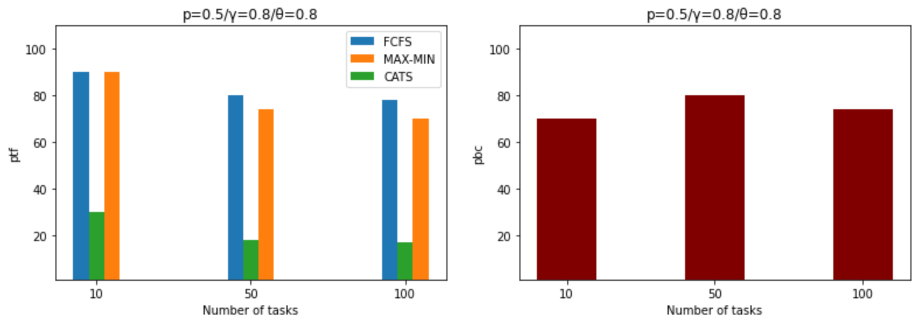


Figure 36: Comparison of ptf and evaluation of pbc $(p = 0.5, \gamma = 0.8, \theta = 0.8)$

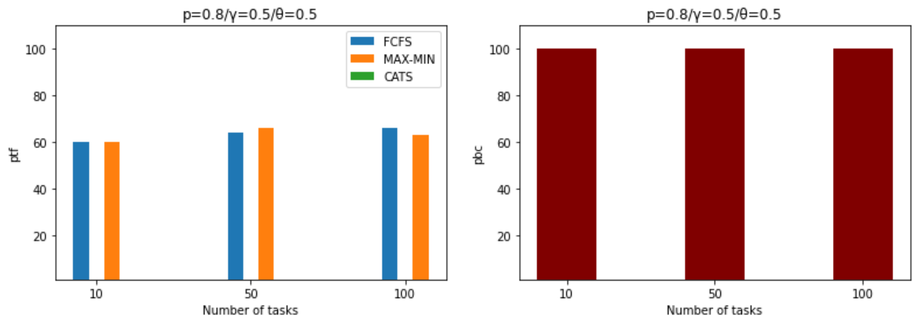


Figure 37: Comparison of ptf and evaluation of pbc $(p = 0.8, \gamma = 0.5, \theta = 0.5)$

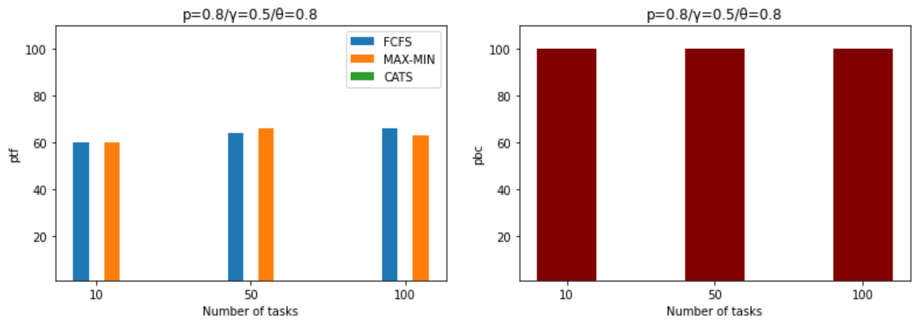


Figure 38: Comparison of ptf and evaluation of pbc $(p = 0.8, \gamma = 0.5, \theta = 0.8)$

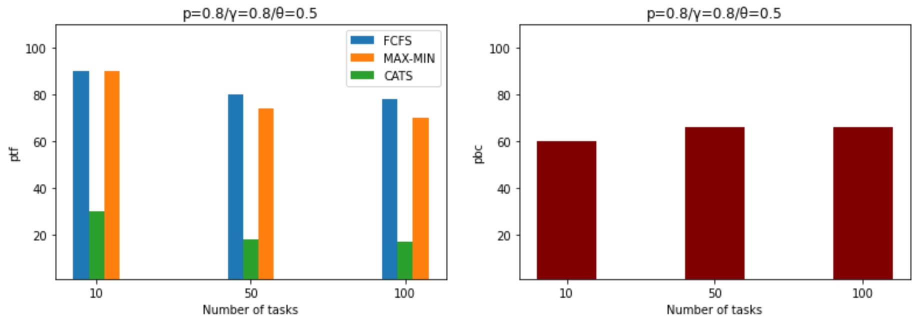


Figure 39: Comparison of ptf and evaluation of pbc ($p = 0.8, \gamma = 0.8, \theta = 0.5$)

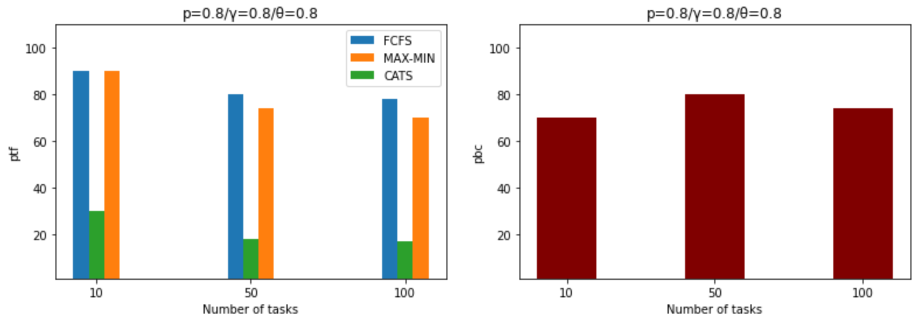


Figure 40: Comparison of ptf and evaluation of pbc ($p = 0.8, \gamma = 0.8, \theta = 0.8$)

rate of approximately 60%, highlighting their persistent inefficiencies.

Figures 39 and 40 show that when both p and γ are increased to 0.8, our model maintains a task failure rate in the range of 17% to 30%, while still ensuring a scheduling percentage based on correlation between 60% and 80%. Meanwhile, FCFS and MAX-MIN continue to demonstrate approximately 80% task failures. These results indicate that while p and θ do not significantly impact performance, γ plays a crucial role. Increasing γ results in more task failures due to fewer suitable nodes for execution. Moreover, our model shows improved performance as the number of tasks increases, unlike the other two methods, which sometimes exhibit an increasing failure rate as task volume rises.

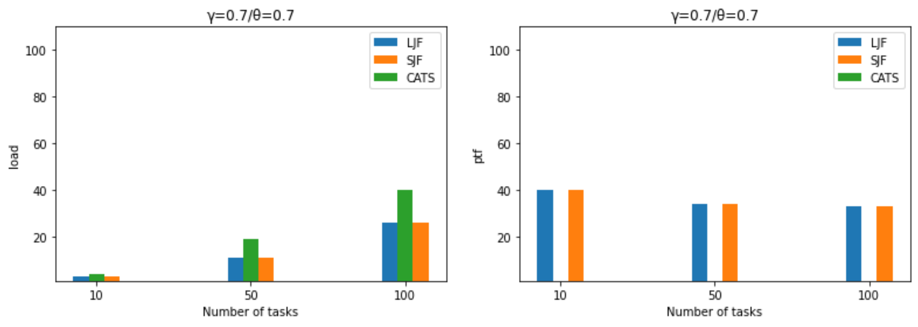


Figure 41: Comparison of *load* and *ptf* $(\gamma = 0.7, \theta = 0.7)$

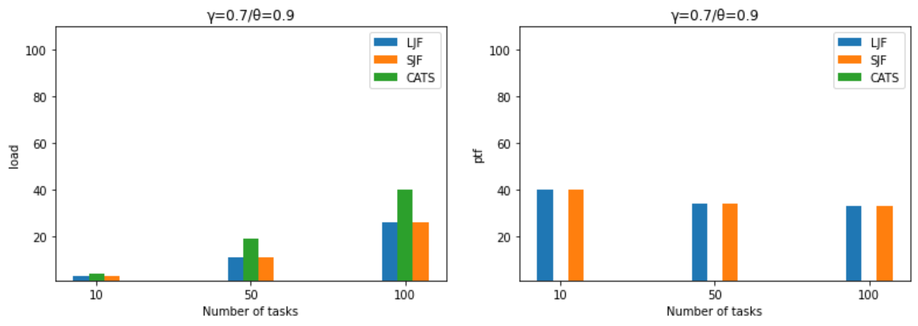


Figure 42: Comparison of *load* and *ptf* $(\gamma = 0.7, \theta = 0.9)$

6.5.5 Second Experimental Scenario(Performance Assessment)

The results of the second experiment, evaluating our model under a different dataset and task allocation scenario, are depicted in the figures below. Figures 41 and 42 show that our model achieves consistently higher load distribution while maintaining lower task failure rates. In contrast, LJF and SJF fail to keep the failure rate low across different task volumes. Additionally, for cases with only ten tasks, a low system load is observed, as expected.

Figures 43 and 44 confirm that our model consistently outperforms LJF and SJF by maintaining higher load efficiency and minimizing task failures. Even with stricter suitability thresholds, at least 95% of tasks are successfully scheduled to appropriate nodes. Meanwhile, LJF and SJF experience higher failure rates. Observing the trends across all cases, the effect of θ is negligible when the remaining parameters are fixed. The main influencing factor is γ, which, when increased, leads to a drop in system load and a rise in task failures due to a reduction in suitable nodes for task execution. Overall, the

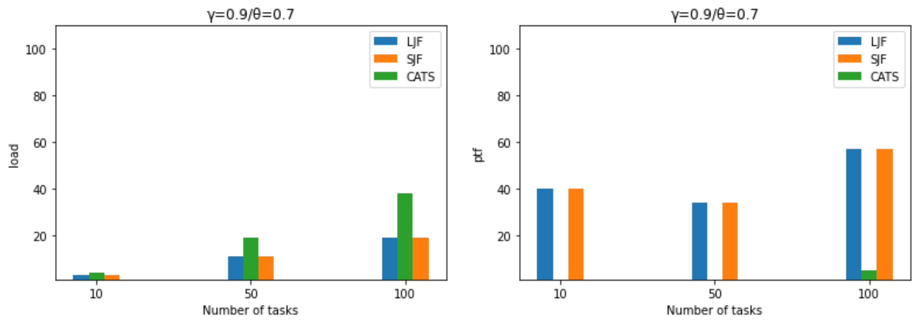


Figure 43: Comparison of $load$ and ptf ($\gamma = 0.9, \theta = 0.7$)

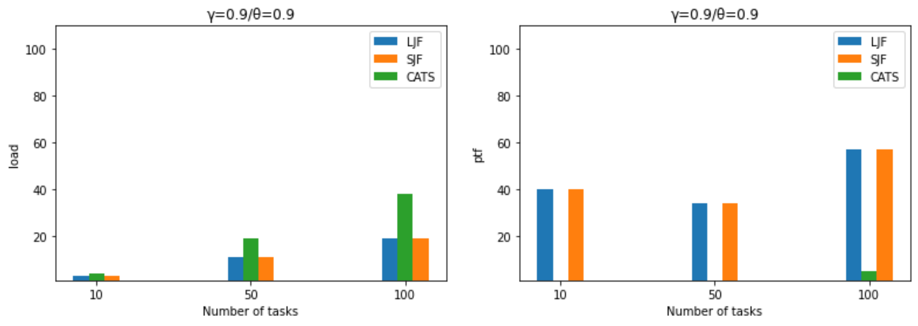


Figure 44: Comparison of $load$ and ptf ($\gamma = 0.9, \theta = 0.9$)

results across both experimental scenarios clearly demonstrate the superiority of our model over traditional scheduling approaches, ensuring balanced load distribution and minimal task failures.

6.6 Summary and Key Findings

The mechanism we implemented was an innovative effort in the assignment of tasks through node suitability and the correlation between tasks. The model employed multi-agent systems and a sum-cost delay evaluation model, which improved load balancing and reduced task failures significantly. The important outcomes are as follows:

- **Task failure minimization**: The scheduling model achieved optimal output from all nodes resulting in over 95% of tasks being executed even in cases of heavy load.
- **Improved load balancing**: With an appropriate load threshold, performance improvements on task completion time were recorded, ensuring balanced load distribution across all nodes.
- **Robustness**: The model was robust and effective in both small and large scale task management settings.

7 Conclusion-Prospects

7.1 Conclusion

This Thesis has provided a significant contribution to the fields of IoT, EC, CC and PC, particularly in addressing critical challenges related to efficient data processing, adaptive ML, and task management in resource-constrained environments. By focusing on the development of novel frameworks and algorithms, this work has enhanced the efficiency, scalability, and adaptability of distributed systems operating under dynamic conditions. One of the central contributions is the cluster-based similarity extraction method, which optimizes data grouping and knowledge sharing in distributed datasets. This method enables edge nodes to identify similar datasets using a synopsis-based approach, facilitating more effective data processing and federated learning across clusters. By focusing on statistical similarity over time, the approach not only minimizes transmission costs but also improves resource allocation and predictive analytics. Another key contribution is the development of a data and resource-aware incremental learning framework, which effectively balances the trade-off between resource constraints and the need for continuous ML model updates. This approach minimizes training time by dynamically rejecting data that does not significantly contribute to the model's performance. As demonstrated in the experimental evaluation, this technique reduced training times by up to 50% while maintaining acceptable accuracy, making it particularly useful for EC nodes with limited computational resources.

Additionally, the research includes advancements in TL within the edge ecosystem, proposing a fuzzy-logic-based method for distribution-based similarity detection. This method addresses the challenge of applying TL in environments where data distributions vary across nodes. By leveraging multi-dimensional distribution-based similarity, the proposed framework enhances decision-making processes related to task offloading and knowledge transfer. The thesis also introduces a drift-based task management mechanism, which addresses the problem of data and concept drift in EC environments. This mechanism adapts task execution strategies based on shifts in data distribution, improving task offloading decisions and optimizing resource utilization across edge nodes. This dynamic approach allows for greater accuracy in task execution while ensuring that computational resources are used efficiently, particularly in environments where data patterns evolve over time.

Finally, the correlation-adaptive task scheduling algorithm provides a novel way to manage tasks in EC settings by considering the correlation between task requirements. This approach reduces task failures and optimizes resource utilization by grouping related tasks at the same node, thereby improving overall system performance. The algorithm is particularly effective in minimizing the need for redundant computation and enhancing the reuse of resources.

Overall, this thesis makes substantial strides in developing more robust, scalable, and efficient distributed systems capable of handling the increasing demands of IoT and pervasive applications.

7.2 Prospects for Future Research

Despite the significant progress made in this thesis, there are still numerous avenues for future exploration. As the complexity of IoT and EC environments continues to grow, further research is needed to refine and extend the contributions made in this work.

7.2.1 Edge-Focused Federated Learning Enhancements

The current similarity extraction framework offers a solid foundation for federated learning in edge clusters, but future research could extend this work by incorporating more advanced privacy-preserving techniques, such as differential privacy. Further research into enhancing communication efficiency and reducing the computational overhead of federated learning in edge environments would be highly valuable, particularly as the number of IoT devices continues to grow.

7.2.2 Energy-Efficient Resource Management Strategies

Energy efficiency remains a critical area of research in EC environments. Future work could explore more sophisticated energy-aware task scheduling algorithms that balance the need for real-time processing with energy conservation. This research could focus on optimizing energy consumption at the edge while maintaining low latency and high accuracy, particularly in environments with constrained energy resources.

7.2.3 Adaptive Real-Time Systems for Edge Environments

Future research could explore the development of more advanced adaptive real-time systems capable of dynamically adjusting their operations in response to changing environmental conditions. Incorporating real-time data processing and anomaly detection capabilities into adaptive systems could improve the resilience and reliability of EC systems, particularly in mission-critical applications such as autonomous vehicles and healthcare.

7.2.4 Advanced Data-Aware Incremental Learning Frameworks

Although the current data-aware incremental learning framework has been shown to significantly reduce training times while maintaining accuracy, there is potential for refinement. Future work could explore the integration of more advanced adaptive learning techniques, such as meta-learning, to optimize model updates in response to abrupt changes in data distributions. Moreover, research into balancing the trade-off between accuracy and resource efficiency in highly volatile environments could yield more efficient learning frameworks.

7.2.5 Enhanced Transfer Learning for Dynamic Environments

Future research should focus on further enhancing TL mechanisms to handle even more diverse and dynamic environments. While this thesis introduced a fuzzy-logic-based TL mechanism for handling distribution-based similarity, more sophisticated methods could be developed to adapt to environments where data distributions are highly volatile or noisy. Incorporating hybrid approaches that combine RL with TL may further improve the adaptability of ML models in EC environments.

7.2.6 Cross-Domain and Heterogeneous Transfer Learning

While this thesis focused primarily on homogeneous TL, future research should investigate cross-domain and heterogeneous TL methods that allow knowledge to be transferred across domains with different feature spaces. This research could explore more complex scenarios where edge nodes need to transfer knowledge between tasks that operate in entirely different domains, such as from image classification to time-series prediction.

7.2.7 Scalable Task Offloading with Predictive Capabilities

Future work could build upon the drift-based task management mechanism by incorporating predictive analytics to anticipate resource bottlenecks before they occur. Predictive offloading strategies that proactively distribute tasks based on anticipated changes in data distribution or node availability could enhance the performance and reliability of EC systems. Additionally, deeper integration of predictive modeling techniques may help optimize energy consumption and reduce latency in time-sensitive applications.

7.3 Closing Remarks

The contributions of this Thesis provide a foundation for further advancements in distributed systems, particularly in IoT and EC environments. By addressing critical challenges in data processing, ML, task management, and resource utilization, this work moves the field closer to achieving more efficient and scalable systems capable of handling complex workloads in dynamic environments. The prospects outlined here offer a roadmap for future research that will continue to push the boundaries of what is possible in distributed computing, ensuring that these systems can meet the demands of the ever-expanding IoT ecosystem.

8 Appendix

8.1 Proof of Lemma 1

We provide the proof for Lemma 1.
As mentioned before $tfv_i[l], tfm_i[l], trov_i[l], NFV_j[l], NFM_j[l], NROV_j[l]$ are independent between them, so we can calculate the initial probability separately for each pair of statistical measure (variance, mean and range of values).

$\mathbf{E}(f(i,j,l)) = 1 * [P(f(i,j,l) = 1] + 0 * [P(f(i,j,l) = 0] = [P(f(i,j,l) = 1] = P[tfv_i[l] \cap NFV_j[l] \neq \emptyset \vee tfv_i[l] = 0] * P[tfm_i[l] \cap NFM_j[l] \neq \emptyset \vee tfm_i[l] = 0] * P[trov_i[l] \cap NROV_j[l] \neq \emptyset \vee trov_i[l] = 0]$

To avoid confusion, we perform further calculations separately. Each task requirement has a probability p of existing. This leads to the following:

$P[tfv_i[l] \cap NFV_j[l] \neq \emptyset \vee tfv_i[l] = 0] = p * P[tfv_i[l] \cap NFV_j[l] \neq \emptyset] + (1-p) = p * P[NFV_j[l] \in [c,d]] + (1-p) = p * P[c \leq NFV_j[l] \leq d] + (1-p) = p * (P[NFV_j[l] \leq d] - P[NFV_j[l] < c]) + (1-p)$

$P[tfm_i[l] \cap NFM_j[l] \neq \emptyset \vee tfm_i[l] = 0] = p * P[tfm_i[l] \cap NFM_j[l] \neq \emptyset] + (1-p) = p * P[NFM_j[l] \in [e,f]] + (1-p) = p * P[e \leq NFM_j[l] \leq f] + (1-p) = p * (P[NFM_j[l] \leq f] - P[NFM_j[l] < e]) + (1-p)$

$P[trov_i[l] \cap NROV_j[l] \neq \emptyset \vee trov_i[l] = 0] = p * P[trov_i[l] \cap NROV_j[l] \neq \emptyset] + (1-p)$

where
$P(trov_i[l] \cap NROV_j[l] \neq \emptyset) = P[[g,h] \cap [NROV_j[l][1], NROV_j[l][2]] \neq \emptyset) = 1 - P([g,h] \cap [NROV_j[l][1], NROV_j[l][2]]) = \emptyset) = 1 - P((NROV_j[l][2] < g) \cup ([NROV_j[l][1] > h)) = 1 - (P(NROV_j[l][2] < g) + P((NROV_j[l][1] > h)) = 1 - (P(NROV_j[l][2] < g) + 1 - P((NROV_j[l][1] \leq h)) = 1 - P(NROV_j[l][2] < g) - 1 + P((NROV_j[l][1] \leq h)) = P((NROV_j[l][1] \leq h)) - P(NROV_j[l][2] < g)$

8.2 Proof of Lemma 2

We provide the proof for Lemma 2.
$\mathbf{E}(su_i[j]) = 1*P(\sum_{l=1}^{r} f(i,j,l) = r) + 0*P(\sum_{l=1}^{r} f(i,j,l) \neq r) = P(\sum_{l=1}^{r} f(i,j,l) = r) = P(f(i,j,1) = 1 \cap f(i,j,2) = 1 \cap \ldots \cap f(i,j,r) = 1) = \prod_{l=1}^{r} P(f(i,j,l) = 1) = \prod_{l=1}^{r} \mathbf{E}(f(i,j,l))$

Bibliography

[1] Gionis, Aristides, Piotr Indyk, and Rajeev Motwani. ”Similarity search in high dimensions via hashing.” Vldb. Vol. 99. No. 6. 1999.

[2] Johnson, Jeff, Matthijs Douze, and Hervé Jégou. ”Billion-scale similarity search with GPUs.” IEEE Transactions on Big Data 7.3 (2019): 535-547.

[3] Kirkpatrick, James, et al. ”Overcoming catastrophic forgetting in neural networks.” Proceedings of the national academy of sciences 114.13 (2017): 3521-3526.

[4] Aljundi, Rahaf, Klaas Kelchtermans, and Tinne Tuytelaars. ”Task-free continual learning.” Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

[5] Shin, Hoo-Chang, et al. ”Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning.” IEEE transactions on medical imaging 35.5 (2016): 1285-1298.

[6] Devlin, Jacob, et al. ”Bert: Pre-training of deep bidirectional transformers for language understanding.” Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers). 2019.

[7] Mao, Hongzi, et al. ”Resource management with deep reinforcement learning.” Proceedings of the 15th ACM workshop on hot topics in networks. 2016.

[8] Zhou, Zhi, et al. ”Edge intelligence: Paving the last mile of artificial intelligence with edge computing.” Proceedings of the IEEE 107.8 (2019): 1738-1762.

[9] Moustakas, T., Kolomvatsos, K. (2024). Cluster based similarity extraction upon distributed datasets. Cluster Computing, 27(3), 2917-2929, https://doi.org/10.1007/s10586-023-04116-5.

[10] Moustakas, T., Tziouvaras, A., Kolomvatsos, K. (2024). Data and resource aware incremental ML training in support of pervasive applications. Computing, 106(11), 3727-3753, https://doi.org/10.1007/s12530-023-09548-3.

[11] Moustakas, T., Kolomvatsos, K. (2024). Homogeneous transfer learning for supporting pervasive edge applications. Evolving Systems, 15(4), 1179-1195, https://doi.org/10.1007/s00607-023-01192-8.

[12] Moustakas, T., Kolomvatsos, K. (2024). Drift-based task management in support of pervasive edge applications. Internet of Things, 27, 101277, https://doi.org/10.1016/j.iot.2024.101277.

[13] Moustakas, T., Kolomvatsos, K. (2023). Correlation adaptive task scheduling. Computing, 105(11), 2459-2486, https://doi.org/10.1007/s00607-024-01338-2.

[14] Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. Computer Networks, 54(15), 2787-2805. https://doi.org/10.1016/j.comnet.2010.05.010

[15] Statista (2023). Number of IoT Connected Devices Worldwide 2010-2030. Available at: https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/

[16] Bytebeam. (2022). A Brief History of Internet of Things (IoT). Available at: https://www.linkedin.com/posts/bytebeam_technology-iot-internetofthings-activity-6976474016467349504-1iJ0

[17] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), 1645-1660.

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

[19] Satyanarayanan, M. (2017). The emergence of edge computing. Computer, 50(1), 30-39.

[20] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality

for delivering computing as the 5th utility. Future Generation computer systems, 25(6), 599-616.

[21] White, T. (2012). Hadoop: The definitive guide. " O'Reilly Media, Inc.".

[22] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In 2nd USENIX workshop on hot topics in cloud computing (HotCloud 10).

[23] IBM. What is cluster computing? https://www.ibm.com/topics/cluster-computing

[24] Alvarez-Melis, David, and Nicolo Fusi. "Geometric dataset distances via optimal transport." Advances in Neural Information Processing Systems 33 (2020): 21428-21439.

[25] Lopez-Paz, David, and Maxime Oquab. "Revisiting classifier two-sample tests." arXiv preprint arXiv:1610.06545 (2016).

[26] Muandet, Krikamol, et al. "Kernel mean embedding of distributions: A review and beyond." Foundations and Trends® in Machine Learning 10.1-2 (2017): 1-141.

[27] Stolte, Marieke, et al. "Methods for quantifying dataset similarity: a review, taxonomy and comparison." Statistic Surveys 18 (2024): 163-298.

[28] Zhang, Dongyu, et al. "Classification of pulse waveforms using edit distance with real penalty." EURASIP Journal on Advances in Signal Processing 2010 (2010): 1-8.

[29] Keogh, Eamonn and Pazzani, Michael. 'Derivative Dynamic Time Warping'. First SIAM International Conference on Data Mining. 1. 10.1137/1.9781611972719.1, 2002.

[30] H. Li and C. Wang, 'Similarity Measure Based on Incremental Warping Window for Time Series Data Mining', in IEEE Access, vol. 7, pp. 3909-3917,doi: 10.1109/ACCESS.2018.2889792, 2019.

[31] Lei Chen, M. Tamer Özsu, and Vincent Oria.'Robust and fast similarity search for moving object trajectories'. In Proceedings of the 2005

ACM SIGMOD international conference on Management of data (SIGMOD '05). Association for Computing Machinery, New York, NY, USA, 491–502. https://doi.org/10.1145/1066157.1066213, 2005.

[32] M. Vlachos, G. Kollios and D. Gunopulos, 'Discovering similar multidimensional trajectories', Proceedings 18th International Conference on Data Engineering, pp. 673-684, doi: 10.1109/ICDE.2002.994784, 2002.

[33] Joan Serra, Josep Ll. Arcos, 'An Empirical Evaluation of Similarity Measures for Time Series Classification', in Knowledge-Based Systems, Jan 2014.

[34] Ratanamahatana, Chotirat and Keogh, Eamonn. 'Three Myths about Dynamic Time Warping Data Mining'. Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005. 10.1137/1.9781611972757.50, 2005.

[35] Tak-chung Fu,'A review on time series data mining,Engineering Applications of Artificial Intelligence',Volume 24, Issue 1,pages 164-181, ISSN 0952-1976,https://doi.org/10.1016/j.engappai.2010.09.007, 2011.

[36] Mörchen, Fabian.'Time series feature extraction for data mining using DWT and DFT', 2003.

[37] Yi-Leh Wu, Divyakant Agrawal, and Amr El Abbadi. 'A comparison of DFT and DWT based similarity search in time-series databases'. In Proceedings of the ninth international conference on Information and knowledge management (CIKM '00). Association for Computing Machinery, New York, NY, USA, 488–495. https://doi.org/10.1145/354756.354857, 2000.

[38] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, Teh Ying Wah, 'Time-series clustering – A decade review',Information Systems,Volume 53,Pages 16-38,ISSN 0306-4379, 2015.

[39] Wang, Xiaozhe and Smith-Miles, Kate and Hyndman, Rob. 'Characteristic-Based Clustering for Time Series Data'. Data Min. Knowl. Discov.. 13. 335-364. 10.1007/s10618-005-0039-x, 2006.

[40] Jun Liu1,, Jaaved Mohammed1, James Carter1, Sanjay Ranka1, Tamer Kahveci1 and Michael Baudis. 'Distance-based clustering of CGH data'

Vol. 22 no. 16 2006, pages 1971–1978 BIOINFORMATICS ORIGINAL PAPER doi:10.10, May 16, 2006.

[41] Gibbons, Phillip and Matias, Yossi,'Synopsis Data Structures for Massive Data Sets', June 1999

[42] Kolomvatsos, K., Anagnostopoulos, C., Koziri, M., Loukopoulos, T., 'Proactive and Time-Optimized Data Synopsis Management at the Edge', IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE), 2020.

[43] Cormode, G., 'Current Trends in Data Summaries', ACM SIGMOD Record, Volume 50, Issue 4, 2021, pp. 6–15.

[44] Siddique, A. B., et al., 'Comparing Synopsis Techniques for Approximate Spatial Data Analysis', PVLDB, 12(11): 1583–1596, 2019.

[45] Kontaxakis, A., et al., 'A Synopses Data Engine for Interactive Extreme-Scale Analytics', in Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020, pp. 2085–2088.

[46] Poepsel-Lemaitre, R., et al., 'In the Land of Data Streams where Synopses are Missing, the One Framework to Bring Them All', PVLDB, 14(10): 1818–1831, 2021.

[47] Zhao, Z., et al., 'Efficient Join Synopsis Maintenance for Data Warehouse', In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, https://doi.org/10.1145/3318464.3389717.

[48] Kim, A., et al., 'Summarizing Hierarchical Multidimensional Data', IEEE 36th International Conference on Data Engineering (ICDE), 2020, pp. 877–888.

[49] Vollmer, M., et al., 'Informative Summarization of Numeric Data', In 31st International Conference on Scientific and Statistical Database Management (SSDBM), 2019.

[50] Personnaz, A., et al., 'EDA4SUM: Guided Exploration of Data Summaries', PVLDB, 15(12): 3590–3593, 2022.

[51] Kolomvatsos, Kostas and Christos Anagnostopoulos. “A Proactive Management Scheme for Data Synopses at the Edge.” ArXiv abs/2107.10558 n. pag, 2021.

[52] Yeh, M. Y., Wu, K. L., Yu, P. S., and Chen, M. S., 'PROUD: a probabilistic approach to processing similarity queries over uncertain data streams'. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (pp. 684-695), 2009.

[53] Cormode, Graham and Garofalakis, Minos and Haas, Peter and Jermaine, Chris, 'Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. Foundations and Trends in Databases' 4. 1-294. 10.1561/1900000004, 2012.

[54] Sedgwick, Philip. "Pearson's correlation coefficient." Bmj 345 (2012).

[55] Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. International journal of forecasting, 22(4), 679-688.

[56] Scalabrini Sampaio, Gustavo; Vallim Filho, Arnaldo R.d.A.; Santos da Silva, Leilton; Augusto da Silva, Leandro.' Prediction of Motor Failure Time Using An Artificial Neural Network'. Sensors 2019, 19, 4342. DOI: 10.3390/s19194342, 2019.

[57] S. De Vito, E. Massera, M. Piga, L. Martinotto, G. Di Francia.'On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario', Sensors and Actuators B: Chemical, Volume 129, Issue 2, Pages 750-757, ISSN 0925-4005, 2008.

[58] Stephan Matzka, 'Explainable Artificial Intelligence for Predictive Maintenance Applications', Third International Conference on Artificial Intelligence for Industries (AI4I 2020), (in press) , 2020.

[59] Aggarwal, C.C., Yu, P.S. 'A Survey of Synopsis Construction in Data Streams'. In: Aggarwal, C.C. (eds) Data Streams. Advances in Database Systems, vol 31. Springer, Boston, MA,2007.

[60] Johannes Gehrke, Flip Korn, and Divesh Srivastava. 2001. 'On computing correlated aggregates over continual data streams'. SIGMOD Rec. 30, 2 (June 2001).

[61] Chen, Guangyong, et al. "Rethinking the usage of batch normalization and dropout in the training of deep neural networks." arXiv preprint arXiv:1905.05928 (2019).

[62] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. pmlr, 2015.

[63] Inoue, Hiroshi. "Multi-sample dropout for accelerated training and better generalization." arXiv preprint arXiv:1905.09788 (2019).

[64] Jiang, Angela H., et al. "Accelerating deep learning by focusing on the biggest losers." arXiv preprint arXiv:1910.00762 (2019).

[65] Yang, Naisen, et al. "Accelerating the training process of convolutional neural networks for image classification by dropping training samples out." IEEE Access 8 (2020): 142393-142403.

[66] Sun, Xu, et al. "meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting." International Conference on Machine Learning. PMLR, 2017.

[67] Zhang, Jiong, Hsiang-Fu Yu, and Inderjit S. Dhillon. "Autoassist: A framework to accelerate training of deep neural networks." Advances in Neural Information Processing Systems 32 (2019).

[68] Song, Zhuoran, et al. "Approximate random dropout for DNN training acceleration in GPGPU." 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019.

[69] Wu, Shuang, et al. "$L1$-norm batch normalization for efficient training of deep neural networks." IEEE transactions on neural networks and learning systems 30.7 (2018): 2043-2051.

[70] Nandini, Gangi Siva, AP Siva Kumar, and K. Chidananda. "Dropout technique for image classification based on extreme learning machine." Global Transitions Proceedings 2.1 (2021): 111-116.

[71] Katharopoulos, Angelos, and François Fleuret. "Not all samples are created equal: Deep learning with importance sampling." International conference on machine learning. PMLR, 2018.

[72] Xiang, Zhengliang, et al. "An active learning method combining deep neural network and weighted sampling for structural reliability analysis." Mechanical Systems and Signal Processing 140 (2020): 106684.

[73] Li, D., et al., 'AdaError: An Adaptive Learning Rate Method for Matrix Approximation-based Collaborative Filtering', in Proc. of the 2018 World Wide Web Conference, April 2018, pp. 741–751

[74] Felsenstein, J., 'Evolutionary trees from DNA sequences: a maximum likelihood approach', J. Mol. Evol., 1981, 17:368–376.

[75] Anderson, E., "The Species Problem in Iris," in Annals of the Missouri Botanical Garden, vol. 23, pp. 457-509, 1936.

[76] Deng, L., "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.

[77] Simonyan, S., Zisserman, A., "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[78] Sandler, M., et al., "Mobilenetv2: Inverted residuals and linear bottlenecks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510-4520, 2018.

[79] He, K., et al., "Deep residual learning for image recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016.

[80] Huang, G., et al., "Densely connected convolutional networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4700-4708, 2017.

[81] Krizhevsky, A., et al., "The CIFAR-10 Dataset," Available: https://www.cs.toronto.edu/ kriz/cifar.html.

[82] Krizhevsky, A., "Learning Multiple Layers of Features from Tiny Images," Technical Report TR-2009, University of Toronto, Toronto, 2009.

[83] Weiss K, Khoshgoftaar T, Wang DD (2016) A survey of transfer learning. J Big Data . https://doi.org/10.1186/s40537-016-0043-6.

[84] Chattopadhyay R, Sun Q, Fan W, Davidson I, Panchanathan S, Ye J (2012) Multisource domain adaptation and its application to early detection of fatigue. ACM Trans. Knowl. Discov. Data 6, 4, Article 18, 26 pages. https://doi.org/10.1145/2382577.2382582

[85] Pan SJ, Tsang I, Kwok J, Yang Q (2011) Domain Adaptation via Transfer Component Analysis, in IEEE Transactions on Neural Networks, vol. 22, no. 2, pp. 199-210, doi: 10.1109/TNN.2010.2091281.

[86] Krishna R, Menzies T, Fu W (2016) "Too much automation? The bellwether effect and its implications for transfer learning," 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE),pp. 122-131.

[87] Obst D, Ghattas B, Claudel S, Cugliari J, Goude Y,Oppenheim G (2022) Improved linear regression prediction by transfer learning. Comput. Stat. Data Anal. 174. https://doi.org/10.1016/j.csda.2022.107499

[88] Kumaraswamy R, Odom P, Kersting K, Leake D, Natarajan S (2015) "Transfer Learning via Relational Type Matching," 2015 IEEE International Conference on Data Mining, pp. 811-816, doi: 10.1109/ICDM.2015.138.

[89] Cao B, Pan SJ, Zhang Y, Yeung DY, Yang Q (2010) Adaptive transfer learning. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI'10). AAAI Press, 407–712.

[90] Valerio L, Passarella A, Conti M (2016) Accuracy vs. traffic trade-off of learning IoT data patterns at the edge with hypothesis transfer learning, 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), Bologna, Italy, pp. 1-6, doi: 10.1109/RTSI.2016.7740634.

[91] Yang B, Fagbohungbe O et al (2022), "A Joint Energy and Latency Framework for Transfer Learning Over 5G Industrial Edge Networks," in IEEE Transactions on Industrial Informatics, vol. 18, no. 1, pp. 531-541, doi: 10.1109/TII.2021.3075444.

[92] Sufian A, You C, Dong M (2021) A Deep Transfer Learning-based Edge Computing Method for Home Health Monitoring, 2021 55th Annual

Conference on Information Sciences and Systems (CISS), Baltimore, MD, USA, pp. 1-6, doi: 10.1109/CISS50987.2021.9400321.

[93] Zheng X, Shah SBH, Ren X, Li F, Nawaf L, Chakraborty C, Fayaz M, Rani S (2021) Mobile Edge Computing Enabled Efficient Communication Based on Federated Learning in Internet of Medical Things. Wirel. Commun. Mob. Comput. 2021 . https://doi.org/10.1155/2021/4410894

[94] Hossain MS, Muhammad G, Amin SU (2018) Improving consumer satisfaction in smart cities using edge computing and caching: A case study of date fruits classification, Future Generation Computer Systems, Volume 88, 2018, Pages 333-341, ISSN 0167-739X, https://doi.org/10.1016/j.future.2018.05.050.

[95] Hou T, Feng G, Qin S, Jiang W (2017) Proactive Content Caching by Exploiting Transfer Learning for Mobile Edge Computing, GLOBECOM 2017 - 2017 IEEE Global Communications Conference, Singapore, pp. 1-6, doi: 10.1109/GLOCOM.2017.8254636.

[96] Yuan Y, Jiao L, Zhu K, Lin X, Zhang L (2022) AI in 5G: The Case of Online Distributed Transfer Learning over Edge Networks, IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, London, United Kingdom, pp. 810-819, doi: 10.1109/INFOCOM48880.2022.9796779.

[97] Sun H, Chen Y, Aved A, Blasch E (2020) Collaborative Multi-Object Tracking as an Edge Service using Transfer Learning, 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Cuvu, Fiji, pp. 1112-1119, doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00146.

[98] Daga H, Nicholson P, Gavrilovska A, Lugones D (2019) Cartel: A System for Collaborative Transfer Learning at the Edge. In Proceedings of the ACM Symposium on Cloud Computing (SoCC '19). Association for Computing Machinery, New York, NY, USA, 25–37. https://doi.org/10.1145/3357223.3362708

[99] Welch BL (2023) On the Z-Test in Randomized Blocks and Latin Squares. Biometrika, vol. 29, no. 1/2, 1937, pp. 21–52. JSTOR, https://doi.org/10.2307/2332405.

[100] Sanderson E, Windmeijer F (2016) A weak instrument F-test in linear IV models with multiple endogenous variables, Journal of Econometrics, Volume 190, Issue 2, Pages 212-221, ISSN 0304-4076, https://doi.org/10.1016/j.jeconom.2015.06.004.

[101] Lin PC, Wu B, Watada J (2010) Kolmogorov-Smirnov Two Sample Test with Continuous Fuzzy Data. In: Huynh, VN., Nakamori, Y., Lawry, J., Inuiguchi, M. (eds) Integrated Uncertainty Management and Applications. Advances in Intelligent and Soft Computing, vol 68. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-11960-617

[102] Garcke J, Vanck T (2014) Importance Weighted Inductive Transfer Learning for Regression. 10.1007/978-3-662-44848-9-30.

[103] Weber M, Doblander C, Mandl P (2020) Detecting Building Occupancy with Synthetic Environmental Data. In Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '20). Association for Computing Machinery, New York, NY, USA, 324–325. https://doi.org/10.1145/3408308.3431124.

[104] Sreenu, Karnam, and M. Sreelatha. "W-Scheduler: whale optimization for task scheduling in cloud computing." Cluster Computing 22.1 (2019): 1087-1098.

[105] Longxin Zhang, Kenli Li, Yuming Xu, Jing Mei, Fan Zhang, Keqin Li,Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster,Information Sciences,Volume 319,2015,Pages 113-131,ISSN 0020-0255, https://doi.org/10.1016/j.ins.2015.02.023.

[106] Zhang, Lei & Chen, Yuehui & Sun, Runyuan & Jing, Shan & Yang, Bo.' A Task Scheduling Algorithm Based on PSO for Grid Computing'. International Journal of Computational Intelligence Research. 4. 37-43. 10.5019/j.ijcir.2008.123,2008.

[107] Q. Cao, Z. -B. Wei and W. -M. Gong, "An Optimized Algorithm for Task Scheduling Based on Activity Based Costing in Cloud Computing," 2009 3rd International Conference on Bioinformatics and Biomedical Engineering, 2009, pp. 1-3, doi: 10.1109/ICBBE.2009.5162336.

[108] Boveiri, Hamid Reza, Reza Javidan, and Raouf Khayami. "An intelligent hybrid approach for task scheduling in cluster computing environments as an infrastructure for biomedical applications." Expert Systems 38.1 (2021): e12536.

[109] Boulougaris, G., Kolomvatsos, K., 'A QoS-aware, Proactive Tasks Offloading Model for Pervasive Applications', in 9th International Conference on Future Internet of Things and Cloud (FiCloud), 22-24 Aug, Rome, Italy, 2022.

[110] Deslauriers, Francis, et al. "Quartet: Harmonizing task scheduling and caching for cluster computing." 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16). 2016.

[111] H. Topcuoglu, S. Hariri and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," in IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274, March 2002, doi: 10.1109/71.993206.

[112] Liu, Yu, et al. "DeMS: A hybrid scheme of task scheduling and load balancing in computing clusters." Journal of Network and Computer Applications 83 (2017): 213-220.

[113] Savvas, Ilias K., and M-T. Kechadi. "Dynamic task scheduling in computing cluster environments." Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks. IEEE, 2004.

[114] Singh, Harvinder, and Gurdev Singh. "Task scheduling in cluster computing environment." 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE). IEEE, 2015.

[115] Wang, Lizhe, et al. "Energy-aware parallel task scheduling in a cluster." Future Generation Computer Systems 29.7 (2013): 1661-1670.

[116] Zheng Shi,Zhiguo Shi, 'Multi-node Task Scheduling Algorithm for Edge Computing Based on Multi-Objective Optimization', in Journal of Physics Conference Series 1607:012017, August 2020.

[117] Shanthan, B. J., et al. "Scheduling for internet of things applications on cloud: A review." Imperial Journal of Interdisciplinary Research 3.1 (2017): 1649-1653.

[118] Mei, Jing, Kenli Li, and Keqin Li. "Energy-aware task scheduling in heterogeneous computing environments." Cluster Computing 17.2 (2014): 537-550.

[119] Parsa, Saeed, and Reza Entezari-Maleki. "RASA: a new grid task scheduling algorithm." International Journal of Digital Content Technology and its Applications 3.4 (2009): 91-99.

[120] Yang, Tao, and Apostolos Gerasoulis. "PYRROS: Static task scheduling and code generation for message passing multiprocessors." ACM International Conference on Supercomputing 25th Anniversary Volume. 1992.

[121] Abdelkader, Doaa M., and Fatma Omara. "Dynamic task scheduling algorithm with load balancing for heterogeneous computing system." Egyptian Informatics Journal 13.2 (2012): 135-145.

[122] Abohamama, A. S., Amir El-Ghamry, and Eslam Hamouda. "Real-time task scheduling algorithm for IoT-based applications in the cloud–fog environment." Journal of Network and Systems Management 30.4 (2022): 1-35.

[123] Sedighi, Art, Yuefan Deng, and Peng Zhang. "Fariness of task scheduling in high performance computing environments." Scalable Computing: Practice and Experience 15.3 (2014): 271-288.

[124] Alizadeh, Mohammad Reza, et al. "Task scheduling approaches in fog computing: A systematic review." International Journal of Communication Systems 33.16 (2020): e4583.

[125] Yu, Yang, and Viktor K. Prasanna. "Energy-balanced task allocation for collaborative processing in wireless sensor networks." Mobile Networks and Applications 10.1 (2005): 115-131.

[126] Hu, Xiaoqing, and Bugong Xu. "Task allocation mechanism based on genetic algorithm in wireless sensor networks." International Conference on Applied Informatics and Communication. Springer, Berlin, Heidelberg, 2011.

[127] Aazam, Mohammad, and Eui-Nam Huh. "Fog computing and smart gateway based communication for cloud of things." 2014 International conference on future internet of things and cloud. IEEE, 2014.

[128] Krishnapriya, S., and P. P. Joby. "QoS aware resource scheduling in Internet of Things-cloud environment." International Journal of Scientific & Engineering Research 6.4 (2015).

[129] Kolomvatsos, K. (2021). Proactive tasks management for pervasive computing applications. Journal of Network and Computer Applications, 176, 102948.

[130] Kolomvatsos, K., Anagnotopoulos, C. (2020). Proactive Tasks Management based on a Deep Learning Model. arXiv preprint arXiv:2007.12857.

[131] B.Baranidharan, K.Saravanan, 'ETSI: Efficient Task Scheduling in Internet of Things ',in International Journal of Pure and Applied Mathematics Volume 117 No. 22 2017.

[132] Mallick, A., Hsieh, K., Arzani, B., Joshi, G. (2022). Matchmaker: Data drift mitigation in machine learning for large-scale systems. Proceedings of Machine Learning and Systems, 4, 77-94.

[133] Gemaque, R. N., Costa, A. F. J., Giusti, R., Dos Santos, E. M. (2020). An overview of unsupervised drift detection methods. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 10(6), e1381.

[134] Bayram, F., Ahmed, B. S., Kassler, A. (2022). From concept drift to model degradation: An overview on performance-aware drift detectors. Knowledge-Based Systems, 245, 108632.

[135] Sukparungsee, S., Areepong, Y., Taboran, R. (2020). Exponentially weighted moving average—Moving average charts for monitoring the process mean. Plos one, 15(2), e0228208.

[136] Abdulmalek, S., Nasir, A., Jabbar, W. A., Almuhaya, M. A., Bairagi, A. K., Khan, M. A. M., Kee, S. H. (2022, October). IoT-based healthcare-monitoring system towards improving quality of life: A review. In Healthcare (Vol. 10, No. 10, p. 1993). MDPI.

[137] H. Jiang and T. Ni, "PB-FCFS-a task scheduling algorithm based on FCFS and backfilling strategy for grid computing," 2009 Joint Conferences on Pervasive Computing (JCPC), Tamsui, Taipei, 2009, pp. 507-510, doi: 10.1109/JCPC.2009.5420131.

[138] M. Kumar and S. C. Sharma, "Priority Aware Longest Job First (PA-LJF) algorithm for utilization of the resource in cloud environment," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2016, pp. 415-420.

[139] Ayad, O. (2014). Learning under Concept Drift with Support Vector Machines. In: Wermter, S., et al. Artificial Neural Networks and Machine Learning – ICANN 2014. ICANN 2014. Lecture Notes in Computer Science, vol 8681. Springer, Cham. https://doi.org/10.1007/978-3-319-11179-7_74

[140] Klinkenberg, R., Joachims, T. (2000). Detecting Concept Drift with Support Vector Machines. International Conference on Machine Learning.

[141] Gâlmeanu H, Andonie R. Concept Drift Adaptation with Incremental–Decremental SVM. Applied Sciences. 2021; 11(20):9644. https://doi.org/10.3390/app11209644

[142] Meenal Jain, Gagandeep Kaur, Vikas Saxena, A K-Means clustering and SVM based hybrid concept drift detection technique for network anomaly detection, Expert Systems with Applications, Volume 193, 2022, 116510, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2022.116510.

[143] Firdose Saeik, Marios Avgeris, Dimitrios Spatharakis, Nina Santi, Dimitrios Dechouniotis, et al., 'Task Offloading in Edge and Cloud Computing: A Survey on Mathematical, Artificial Intelligence and Control Theory Solutions',in Computer Networks, Elsevier, 2021, 195, ff10.1016/j.comnet.2021.108177ff.ffhal-03243071, 31 May 2021.

[144] Khan, Atta ur Rehman and Othman, Mazliza and Madani, Sajjad and Khan, Samee, 'A Survey of Mobile Cloud Computing Application Models', in IEEE Communications Surveys and Tutorials. 16. 393-413. 10.1109/SURV.2013.062613.00160, 2014.

[145] Karanika, A., Oikonomou, P., Kolomvatsos, K., Loukopoulos, T., 'A Demand-driven, Proactive Tasks Management Model at the Edge', IEEE FUZZ-IEEE, 2020.

[146] Kolomvatsos, K., Anagnostopoulos, C., 'A Proactive Statistical Model Supporting Services and Tasks Management in Pervasive Applications', IEEE Transactions on Network and Service Management, vol. 19, no. 3, pp. 3020-3031, Sept. 2022.

[147] Kolomvatsos, K., Anagnostopoulos, A., 'Multi-criteria Optimal Task Allocation at the Edge', Future Generation Computer Systems, 93:358–372, 2019.

[148] Yuxuan Sun,Xueying Guo,Jinhui Song,Sheng Zhou, Zhiyuan Jiang, Xin Liu and Zhisheng Niu,'Adaptive Learning-Based Task Offloading for Vehicular Edge Computing Systems', in arXiv:1901.05205 [cs.IT], 16 January 2019.

[149] Tuyen X. Tran, Dario Pompili, 'Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks', in arXiv:1705.00704 , 1 May 2017.

[150] Siqi Zhang, Na Yi, Yi Ma, 'Correlation-Based Device Energy-Efficient Dynamic Multi-Task Offloading for Mobile Edge Computing', in 2021 IEEE 93rd Vehicular Technology Conference, 2021.

[151] Te-Yi Kan, Yao Chiang, Hung-Yu Wei, 'Task Offloading and Resource Allocation in Mobile-Edge Computing System ', in The 27th Wireless and Optical Communications Conference (2018).

[152] Mutlag, Ammar Awad, et al. "MAFC: Multi-agent fog computing model for healthcare critical tasks management." Sensors 20.7 (2020): 1853.

[153] Fang, Juan, and Wenzheng Zeng. "Offloading strategy for edge computing tasks based on cache mechanism." Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence. 2020.

[154] Guo, Shaoyong, et al. "Blockchain meets edge computing: Stackelberg game and double auction based task offloading for mobile blockchain." IEEE Transactions on Vehicular Technology 69.5 (2020): 5549-5561.

[155] Bhoi, Upendra R. and Purvi N. Ramanuj. "Enhanced Max-min Task Scheduling Algorithm in Cloud Computing." (2013).

[156] Alworafi, Mokhtar A., et al. "An improved SJF scheduling algorithm in cloud computing environment." 2016 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT). IEEE, 2016.