EdgeAlarm: Edge Assisted Real-time and Intelligent Alarm Management System for Oil Refinery

by

Waris Gill

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

September 17, 2020

EdgeAlarm: Edge Assisted Real-time and Intelligent Alarm Management System for Oil Refinery

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Waris Gill

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Öznur Özkasap (Advisor)

_____

Prof. Attila Gürsoy (Advisor)

_____

Prof. Suat Özdemir

_____

Assoc.Prof. Müge Sayıt

_____

Assist.Prof. Barış Akgün

Date: _____

To my beloved mother.

ABSTRACT

EdgeAlarm: Edge Assisted Real-time and Intelligent Alarm Management System for Oil Refinery

Waris Gill

Master of Science in Computer Science and Engineering

September 17, 2020

Edge computing is changing the course of data processing in many industries as it enables the processing of data closer to its origin. Although various edge computing architectures exist for different industries, to the best of our knowledge, there is no extensive and practical utilization of edge computing for oil refineries and specifically for alarm management systems. Alarm management system is an integral part of every industry to carry out various operations safely. An ineffective alarm system can increase the operator workload, which can lead to catastrophic events. The factors such as alarm flooding, momentary alarms, and nuisance alarms impede the operator productivity and hence increase his/her workload. In this thesis, we propose an efficient real-time alarm management system based on the edge computing paradigm, namely EdgeAlarm, without altering the existing infrastructure of an oil refinery. The proposed EdgeAlarm system consists of three modules: Storage Reduction (SR) module, Operator Workload Reduction (OWR) module, and an Artificial Intelligence assistant (AI) module. The SR module is responsible for filtering the momentary alarms at the edge and only sends useful alarms to the cloud (Historian server). In the OWR module, we propose two novel techniques to reduce the operator workload by classifying the true and nuisance alarms. The AI module consists of a Recurrent Neural Network (RNN) to predict the future alarms to assist the operator in real-time.

We evaluated the proposed EdgeAlarm system on the 13 months alarm dataset of an alarm management system of Tüpraş oil refinery plant. Our findings indicate that the SR module achieves a reduction in the storage utilization of the Historian server by approximately 10%. The OWR module reduces the operator workload approximately by 50% by finding the true and nuisance alarms. Finally, the AI

module assists the operator to avoid or prepare for any abnormal situation in advance by predicting the majority of the upcoming alarms with the help of RNN with an accuracy of 55% per alarm source at the edge in real-time and some alarm sources are predicted with more than 90% accuracy.

ÖZETÇE

EdgeAlarm: Petrol Rafinerisi için Sınır Bilişim Destekli Gerçek Zamanlı ve Akıllı Alarm Yönetim Sistemi

Waris Gill

Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans

17 Eylül, 2020

Sınır bilişim, verilerin kaynağına daha yakın bir şekilde işlenmesini sağladığı için birçok sektörde veri işleme sürecini değiştirmektedir. Farklı endüstriler için çeşitli uç hesaplama mimarileri mevcut olsa da, bildiğimiz kadarıyla, petrol rafinerileri ve özellikle alarm yönetim sistemleri için uç hesaplamanın kapsamlı ve pratik kullanımı mevcut değildir. Alarm yönetim sistemi, çeşitli işlemleri güvenli bir şekilde yürütmek için her endüstrinin ayrılmaz bir parçasıdır. Etkisiz bir alarm sistemi, operatörün iş yükünü artırabilir ve bu da felaket olaylarına yol açabilir. Alarm seli, anlık alarmlar ve rahatsız edici alarmlar gibi faktörler operatörün üretkenliğini engeller ve dolayısıyla iş yükünü artırır.

Bu tez çalışmasında, bir petrol rafinerisinin mevcut altyapısını değiştirmeden, EdgeAlarm adlı uç hesaplama modeline dayanan verimli bir gerçek zamanlı alarm yönetim sistemi öneriyoruz. Önerilen EdgeAlarm sistemi üç modülden oluşur: Depolama Azaltma (SR) modülü, Operatör İş Yükü Azaltma (OWR) modülü ve Yapay Zeka asistanı (AI) modülü. SR modülü, uçtaki anlık alarmları filtrelemekten sorumludur ve yalnızca yararlı alarmları bulut birimine (Historian sunucusu) gönderir. OWR modülünde, operatör iş yükünü azaltmak için gerçek ve rahatsız edici alarmları sınıflandıran iki yeni teknik öneriyoruz. AI modulü, operatöre gerçek zamanlı olarak yardımcı olmak için gelecekteki alarmları tahmin etmek için bir Tekrarlayan Sinir Ağından (RNN) oluşur.

Önerilen EdgeAlarm sistemini Tüpraş petrol rafinerisi tesisi alarm yönetim sisteminin 13 aylık alarm veri setini kullanarak değerlendirdik. Bulgularımız, SR modülünün Historian sunucusunun depolama kullanımında yaklaşık %10 oranında azalma sağladığını göstermektedir. OWR modülü, doğru ve rahatsız edici alarmları bularak operatörün iş yükünü yaklaşık %50 oranında azaltmaktadır. AI modülü ise, operatörün yaklaşan alarmların çoğunu RNN yardımıyla uçta gerçek zamanlı

olarak alarm kaynağı başına %55 doğrulukla tahmin ederek herhangi bir anormal durumdan önceden kaçınmasına veya buna hazırlanmasına yardımcı olmaktadır.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisors Prof. Öznur Özkasap and Prof. Attila Gürsoy for their support, guidance, and motivation throughout my studies at Koç University. I am grateful to my committee members Dr. Barış Akgün, Dr. Müge Sayıt, and Prof. Suat Özdemir for their support and participation. I also want to thank KUTEM (Koç University Tüpraş Energy Center) and Huawei for supporting me throughout my graduate studies.

In the end, I want to pay my gratitude to my friends and family for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AI	Artificial Intelligence Assistant
ATEX	Atmosphere Explosive
CNNs	Convolutional Neural Networks
DCS	Distributed Control System
FC	Fully Connected
GRU	Gated Recurrent Unit
NLP	Natural Language Processing
NN	Neural Network
OWR	Operator Workload Reduction
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SR	Storage Reduction
SCADA	Supervisory Control and Data Acquisition

Chapter 1

INTRODUCTION

1.1 Context

Sensor networks are increasingly deployed in the industry (e.g., an oil refinery) to monitor various operations such as pipeline leakages, pressure and temperature deviations, optimizing pumping operations, ensuring workers' safety and equipment health, over the past few decades [Lin et al., 2016][Aalsalem et al., 2018]. The sensors transmits the data to the distributed control system (DCS) or Supervisory Control and Data Acquisition (SCADA) [Boyer, 2009], which warns the operator by triggering warnings when a process shifts away from its normal behavior (e.g., a safe threshold was crossed by a process temperature). The sensors' data (i.e., alarms or warnings) are transmitted to DCS (which is considered as the brain of automated control systems) and then displayed on the operator screen. After the appearance of an alarm, the operator either silences the alarm, acknowledges it, or triggers the corresponding response, if required [Goel et al., 2017].

Due to the increasing deployment of sensor networks, an alarm system has to process a massive amount of data produced by these sensors, which has significantly increased the operator's workload and plant complexity. During an anomalous case, a plant process deviates from its standard operating state and many alarms (e.g., alarm flooding) are caused which would result in a catastrophic event if not properly entertained. Alarm flood is a scenario in which dozens of alarms are triggered during an abnormal situation and these alarms have a catastrophic cascading impact on the entire factory. A properly functioning alarm system and timely intervention by an operator can efficiently handle and avert such abnormal situations. Moreover, an operator can also configure alarms at the software level by a few clicks to monitor

a device status to spot irregular conditions [Nimmo, 2005] [Hollifield and Habibi, 2011]. To this end, many nuisance alarms (i.e., alarms that do not require the operator's action) are generated, which can overwhelm the operator even in normal circumstances. Often such alarms are activated to such a degree that it is extremely challenging for an operator to interpret and respond to such alarms. It is also very likely that vital warnings are missed during irregular conditions due to these redundant warnings.

If the alarms are not dealt timely, it would cause serious accidents. Several catastrophic accidents due to mismanagement of alarms are listed in [Goel et al., 2017]. For example, according to the report of the explosion at the Milford Haven refinery, the key causes of the explosion were the triggering of too many alarms, poor prioritization, poor control room alarm display design and an alarm flood (275 alarms in 11 minutes) before the explosions. Similarly, the failure to manage equipment and alarms leads to fires in BP Texas refinery in 2005, due to which 15 people died, 180 were wounded, and $1.5 billion was lost. Also, an oil refinery can benefit from an efficient alarm management system in various scenarios such as reduction of irregular conditions ($2.88 million), avoidance of maintenance ($1.11 million), equipment repairing ($0.22 million), and improvement in plant production ($1.68 million) [Goel et al., 2017]. Clearly, these incidents and financial gains show that the alarm management systems require a significant need to be upgraded and required novel state-of-the-art solutions.

Currently, oil refineries are widely using DCS and SCADA systems to process the alarms. In these architectures, after displaying alarms to the operator, without any preprocessing (e.g., removing momentary alarms), the alarms are directly forwarded to the Historian server for further analysis in the future. Additionally, these architectures are not flexible to protocol or software changes to integrate newer methodologies to process the alarm data. Moreover, these current technologies are costly (equipment and maintenance) and take a while to process the alarms. Besides that, the existing architectures do not offer much assistance to field operators during situations like alarm flooding. Above all, in these systems, alarm data is processed often weekly or biweekly by the experts to identify important alarms and other

problems, such as alarm chattering and staling alarms (alarms that remained active for more than 12 hours). Hence, in large production plants such as oil refineries, due to the enormous amount of historical alarms, the traditional storage approaches and analysis techniques are incapacitating [Li et al., 2015] [Dorgo et al., 2018]. To illustrate, according to [Li et al., 2015], a chemical operation needs to store and analyze 10 million alarms every year, and these numbers are enormously higher these days. Thus, a real-time and intelligent alarm management system is required, which can not only have financial advantages but can also reduce the operator's workload, help to prevent tragic scenarios, and address the limitations in the existing systems.

Edge computing enables the processing of data closer to its origin. Compared to cloud computing, an additional edge layer consisting of cloudlets is added between the cloud and the data source (e.g., sensor networks) to process the data closer to its source [Ray et al., 2019] [Alnoman et al., 2019]. Simply stated, a cloudlet can be defined as a "data center in a box". It is logically adjacent to the corresponding sensor network (e.g., single-hop Wi-Fi connection) to have low end-to-end latency and high bandwidth. It can perform basic cloud operations, such as data collection, training machine learning (ML) models on obtained data, real-time predictive data analysis, and can mask temporary cloud outages [Khan et al., 2019]. With the assistance of the edge computing architecture, the alarm management systems in an oil refinery can process and store its alarms data locally in real-time, and only a limited amount of valuable alarms' data would be forwarded to the cloud for future analysis.

1.2 Contributions

The major contributions of this thesis are as follows.

1. We propose EdgeAlarm, a novel real-time alarm management system based on the edge computing paradigm, which consists of three modules: Storage Reduction (SR) module, Operator Workload Reduction (OWR) module, and Artificial Intelligence Assistant (AI) module. It addresses the limitations of existing alarm management systems without altering the existing infrastructure

of the oil refinery.

2. The SR module filtered out the momentary alarms (i.e., alarms whose duration is very short (10 seconds)) at the edge in real time to reduce the utilization of Historian server storage space.

3. In the OWR module, we propose two novel techniques to reduce the operator workload for alarm processing. The former is used to classify the true and nuisance alarms from the alarm data. While the latter is used to further reduce the number of nuisance alarms.

4. In the AI module, we propose a Recurrent Neural Network (RNN) architecture to predict upcoming alarms in real-time at the edge to assist the operator.

5. We evaluated the proposed edge architecture and its techniques based on the reduction in the operator workload and storage utilization on the 13 months alarm dataset of a Tüpraş [tup,] oil refinery plant alarm management system. We have shown that the SR module can save storage space by approximately 10%, the operator workload can be reduced approximately by 50% or even more with the help of the OWR module, and the AI module can successfully forecast the majority of the upcoming alarms with an accuracy of more than 55% per sensor (i.e., source) and for some alarm sources even more than 90%.

6. We also make the code and dataset public for further analysis [edg,].

1.3 Organization

The rest of the dissertation is organized as follows: Chapter 2 gives an overview of the related work and the key characteristics and performance indicators of an effective alarm management system are discussed. Chapter 3 describes the edge computing architecture for an alarm management system. Chapter 4 explains the SR module to reduce to storage space of the Historian server by filtering the momentary alarms at the edge, the OWR module to reduce the operator workload, and the AI module

to predict upcoming alarms. Chapter 5 discusses the results and effectiveness of edge-based alarm management system. Finally, chapter 6 summarizes the work and gives the future directions.

Chapter 2

ALARM MANAGEMENT SYSTEM AND RELATED WORK

In this chapter, we explain the basics of an alarm management system and the different challenges faced by the existing alarm systems. Then, we provide a review of the different edge computing architectures proposed for various industries such as the health sector, the oil industry, and the autonomous vehicle industry [Gill et al., 2020b].

2.1 Alarm Management System

2.1.1 Introduction

ISA-18.2 [Hollifield, 2010] defines an alarm as follows: "an alarm is an audible and/or visible means of indicating to the operator an equipment malfunction, process deviation, or abnormal condition requiring a response". The alarm system is the collection of hardware and software that detects an alarm state, communicates an indication of that state to the operator, and records changes in the alarm state. Alarms are particularly important for the safe and accurate operation of a plant and therefore it is essential to process and analyze the alarm data efficiently and on time to avoid any abnormal situations. Since hundreds of alarms are generated in a plant, operators must be able to recognize high priority alarms that require immediate action. Engineering Equipment and Materials Users Association (EEMUA) suggests that an operator can handle 11 alarms per 10 minutes. However, according to [Rothenberg, 2009], the actual number of alarms occurred in different industries such as Power, Oil & Gas per 10 minutes is higher as compared to the suggested standard.

Table 2.1: Recommended Key Performance Indicators (KPIs) by EEMUA 191 & ANSI/ISA 18.2 recommended for the alarm management system of a single plant

KPI	EEMUA 191	ANSI/ISA 18.2
Per day average number of alarms	less than 144 (may be manageable up to 288)	less than 150 (300 maybe manageable)
Number of Average standing alarms per day	less than 10	less than 5
Peak number of alarms per 10 minutes	less than 10	less than 11
Average Alarms per 10 minutes interval	1	1-2
Distribution of % (low/med/high)	80/15/5	80/15/5

2.1.2 Alarm Management System Key Performance Indicators

EEMUA 191 guidelines and ANSI/ISA 18.2 have defined some Key Performance Indicators (KPIs) over a period that served as a goal for an effective alarm management system [Goel et al., 2017], which are listed in Table 2.1. The presented KPIs should be the target value for a plant to perform better. Initially, these values are a little bit demanding, but they are achievable over a period. According to EEUMA, the major characteristics of an alarm are as follows:

1. Uniqueness: For the same process parameter, no duplicate alarm should be configured.

2. Prioritization: The alarm should be prioritized in such a way that it gives significant critical info to the operator to respond to the alarm.

3. Timeliness: The alarms should not appear too early or too late. It should appear in its time. Showing it too early or too late may have an adverse effect on the operator's response.

4. Understandability: The alarm should have a significant description of its triggering event.

5. Relevance: It must be relevant to the process which is being monitored.

6. Response Requirement: Each alarm must require a response from the operator.

2.1.3 Issues with Current Alarm Systems

In the current systems such as SCADA and DCS, the alarms are shown on the operator screen based on their timestamp and after the appearance of an alarm, the operator activates the corresponding response. After the operator has acted, the controller transmits the operator's instructions to the actuators (e.g., control valves, dampers) to adjust the actuator to a new state such that the process variable returns to its normal state. Simply stated, an alarm is a notification to the plant operator that a process variable (e.g., temperature or pressure) deviates from the normal or safe state. Once the alarm is activated, it requires the operator's attention. After the appearance of an alarm, the operator executes one of the following actions: silence the alarm, or "acknowledgement" and activate an appropriate response [Goel et al., 2017]. An effective alarm system presents only the relevant and required alarms to the operator for action during critical situations. However, an ineffective alarm system has no master alarm database, no operator action required for most of the alarms, unavailability of the proper guidelines to add or remove alarms, missing of important alarms during critical incidents, minor upsets can cause many alarms which cannot be handle by the operator, appearance of an alarm even longer than 24 hours and too many alarms with high priority [Goel et al., 2017].

Alarms can be configured very easily in systems like SCADA and DCS by a few clicks. Due to this easiness, the number of alarms has dramatically increased from a few hundred to thousands [Nimmo, 2005] [Hollifield and Habibi, 2011]. Thus, the large number of configured alarms sometimes causes problems like alarm flooding [Bergquist et al., 2003] and if the response is not taken effectively, it will have a significant impact on the safety of the plant [Laberge et al., 2014]. Alarm flooding is one of the key reasons not to meet the KPIs mentioned in Table 2.1. It is a situation in which a plant triggers more alarms than an operator can effectively handle in a fixed time span [Hollifield, 2010] [Rothenberg, 2009] [Bransby, 2001] [Bullemer et al., 2011]. According to [Pariyani et al., 2010], a DCS roughly records 500 to 1000 abnormal events every day in a large-scale unit. Therefore, it is particularly important that the alarm management system should assist the operator in such

events by identifying true alarms that required the operator action, otherwise it will impede the overall performance of the plant. In chapter 4, we have proposed three modules to address the issues of existing alarm management systems with the help of edge computing.

2.2 Related Work

Table 2.2 lists the edge computing architectures and their corresponding use cases in different areas. From this table, we can see that the edge computing paradigm spans to solve problems related to health, security, and latency. A brief description of each listed architecture is as follows.

2.2.1 Edge Computing Architectures in Industry

Edge computing architectures are deployed in industries for various purposes such as identifying defected products and malfunctioning of a process. For instance, an edge computing architecture is proposed to classify faulty Dynagraph cards in real-time generated by rod pumps in [Saghir et al., 2018]. Rod pumps are used to produce hydrocarbons in the oil field and these pumps continually generate data from which a Dynagprah card is formed. It helps the operator to determine the health of the rod pump. Similarly, to find the defected products in the industrial manufacture process, a DeepIns system based on edge computing is proposed [Li et al., 2018]. The data from the installed cameras are sent to nearby edge nodes to find the degree of the defect at the edge. Several edge computing architectures are proposed to address different issues in the health sector. For example, to classify different abnormalities and to find heart problems in real-time, several edge architectures are proposed in [Azimi et al., 2018] [Tuli et al., 2020] [Hossain and Muhammad, 2016] [Dubey et al., 2015]. Similarly, an edge-based health monitoring system to detect falls in real-time is proposed in [Queralta et al., 2019]. In this architecture, only 10 bytes of data were sent to the cloud. Another edge computing architecture is presented in [Azimi et al., 2017] to detect arrhythmia for patients suffering from Cardio Vascular Diseases (CVDs). Similarly, an edge architecture is proposed in [La

et al., 2019] to detect abnormal events around the monitored patient.

The edge computing paradigm is also an important part of autonomous vehicles. Edge nodes are deployed alongside the road to provide a shared computing infrastructure. Moving vehicles can access the computing infrastructure with low latency. Since the edge nodes are static, it is hard to determine the best edge node (i.e., lower cost of using an edge node at a specified time and location). The best edge node varies based on the traffic. In [Memon and Maheswaran, 2019], an approach is presented to determine the best edge node based on the location and time.

Edge-based systems are also deployed in smart cities and homes for safety purposes. In [Janjua et al., 2019], the authors proposed a technique to detect rare events in audio such as a gunshot, glass break, scream, and siren based on edge computing paradigm.

2.2.2 Edge Computing Architectures to mitigate Security Attacks on Internet of Things (IoT) Devices

Edge computing architectures are deployed to provide security to the industrial internet of things devices such as sensor networks. An edge-based architecture is proposed in [Diro and Chilamkurti, 2018] [Abeshu and Chilamkurti, 2018] to detect security attacks on IoT devices. Likewise, to detect security attacks on vessels (e.g., spoofing the vessels positions data), an edge computing architecture was proposed in [Zissis, 2017]. In this architecture, each Automatic Identification System (AIS) was viewed as an edge node that was running a one-class SVM to autonomously detect the spoofing attacks on the vessels data at the edge. The proposed system can detect security attacks in real-time.

2.2.3 Edge Computing Architecture for Data Compression

Edge computing architectures are also used for data compression to reduce bandwidth usage. For instance, [Ghosh and Grolinger, 2019] utilizes the autoencoders to reduce data at the edge. The edge layer has the data encoder and the cloud has the decoder part of the autoencoder. The encoder compresses the data at the

Table 2.2: Edge computing architectures in different industries and their usage.

Solution Name	Area	Usage
Upstream Automation [Saghir et al., 2018]	Oil Industry	To classify Dynagraph cards.
Inspection Fog [Li et al., 2018]	Smart Industry	To classify defects in the manufacturing process.
Fog Healthcare [Azimi et al., 2018], [Tuli et al., 2020]	Smart Health	To classify health problems.
HealthIIoT [Hossain and Muhammad, 2016]	Smart Health	To detect abnormal ECG.
Enhancing Telehealth [Dubey et al., 2015]	Smart Health	To find patterns in ECG.
Edge-AI in LoRa [Queralta et al., 2019]	Smart Health	To detect falls.
HiCH [Azimi et al., 2017]	Smart Health	To detect arrhythmia.
Intelligent Fog [La et al., 2019]	Smart Health	To detect patient activities.
Deep Fog Security [Diro and Chilamkurti, 2018], [Abeshu and Chilamkurti, 2018]	Generic	To detect security attacks.
Intelligent Edge Security [Zissis, 2017]	Generic	To detect security attacks.
Vehicular Fog Computing [Memon and Maheswaran, 2019]	Automobile	To optimize fog to fog transitions.
Edge Analytics [Ghosh and Grolinger, 2019]	Generic	To reduce features.
IRESE [Janjua et al., 2019]	Generic	To find rare events in audio.

edge while decoders decompress the data at the cloud layer, hence, reducing the bandwidth usage.

Chapter 3

EDGE COMPUTING ARCHITECTURE FOR THE ALARM MANAGEMENT SYSTEM

This chapter explains the proposed 3-tier architecture for the alarm management system is based on the fundamental 3-layered structure of edge computing paradigm [Gill et al., 2019][Gill et al., 2020c], as shown in Figure 3.1. It consists of a sensor network layer, an edge layer, and a cloud layer. The sensor network layer consists of sensors, actuators, and gateway nodes to monitor a process such as an oil field, which has hazardous chemicals at elevated pressure and temperature. The edge layer consists of cloudlets that process data in logical proximity and the Historian/cloud layer is used to monitor all cloudlets. The cloudlets are interfaced with the DCS or SCADA system without altering the existing network infrastructure of the oil refinery. Now, the DCS will work as gateways and forward the data to the cloudlets. The detailed functionality and interaction of the architecture layers are described as follows and depicted in Figure 3.2.

3.1 Sensor Network Layer

Each **sensor node** is integrated with one or more sensors of different types such as temperature, pressure, and acoustic sensors to track events such as leakage of pipelines, pressure & temperature fluctuations, and workers' safety. Additionally, it also has a microcontroller, battery, radio transceiver, and lightweight operating systems such as Contiki [Dunkels et al., 2004] or TinyOS [Levis et al., 2005]. While operating in a hazardous area such as an oil field, all the nodes should be enclosed in ATEX (ATmosphere EXplosive) approved enclosures. According to [Yoon et al., 2011][Jovašević-Stojanović et al., 2015], many inexpensive sensors can be deployed to provide spatial and temporal resolution in the sensor readings. The sensors

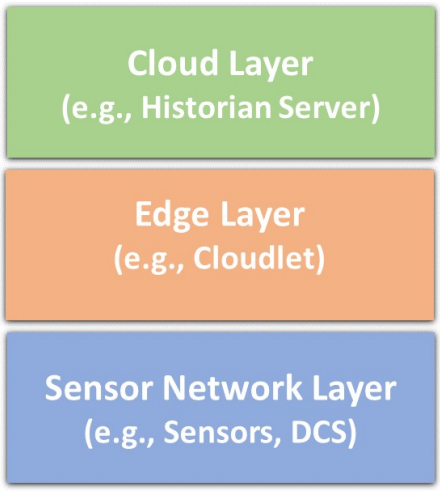

Figure 3.1: Edge computing architecture for an oil refinery. The latency between the sensor network layer and edge layer is lower as compared to the latency between the edge layer and cloud layer (Historian Server).

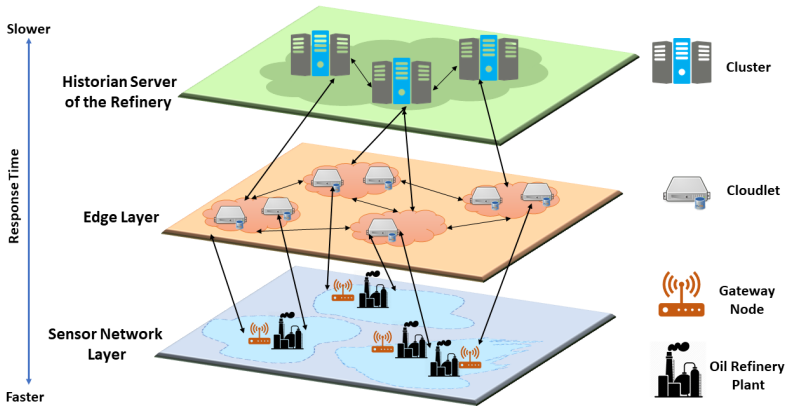

Figure 3.2: A 3-Tier architecture for the oil and gas industry. The sensor network layer monitors the oil refinery plant processes and sends the data to the cloudlets in the edge layer which are in the local area network of the refinery. After the processing, only the important data is forward to the cloud.

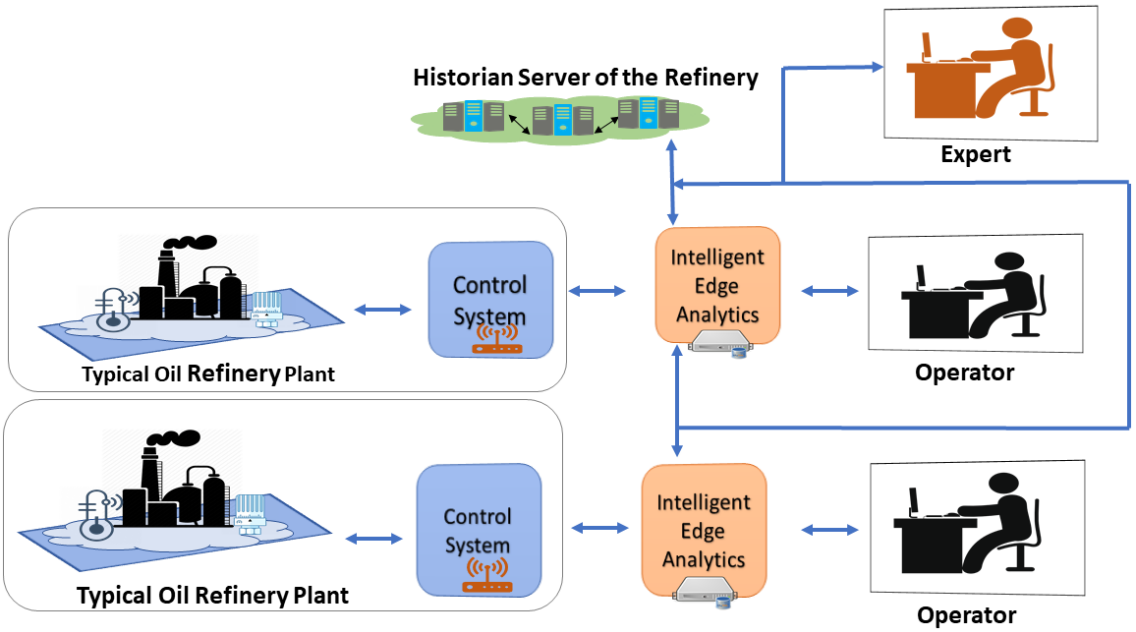

Figure 3.3: With the help of the edge analytics layer, the expert can track the performance of each plant and can assist the operators during critical situations in real-time.

transferred the data (i.e., alarms) towards the control system **gateways** (e.g., DCS) through short-range communication. The key task of a gateway (i.e., control system) is to transmit the sensors networks' data to the cloudlet and operator's commands to the actuators deployed in the plant as shown in Figure 3.3. A sensor network may have more than one gateway, and each gateway in the system has an association with more than one cloudlet to address failures and reliability issues.

3.2 Edge Layer

On each cloudlet, the SR, OWR, and AI modules are deployed. The SR module will filter out the momentary alarms, the OWR module will improve the productivity of the operator by classifying nuisance & true alarms in real-time, and the AI module will assist the operator by predicting upcoming alarms to avert or prepare for abnormal situations in advance.

Upon receiving data (i.e., alarms) from the gateways, the cloudlet processes it in real-time with these modules. After data processing, if there is an anomaly, it

sends a signal to the operator to take the corresponding action. It will also warn the staff employed in the area in the event of an emergency. The cloudlets will also consist of a dashboard that will help the operators to interpret & visualize the alarms data and take actions at once in real-time. One of the major challenges in oil refineries is the shortage of professional experts. To address this issue, the cloudlets deployed in the same local area network can also communicate with each other to share information with other operators and experts to get feedback in real-time. In an emergency, operators in a plant can seek real-time assistance from other operators and professionals to deal with such situations as shown in Figure 3.3, which will substantially increase the overall efficiency of an oil refinery. The additional potential advantages of the cloudlets are as follows:

Training of machine learning models: One of the key features of the cloudlet is that machine learning models can be trained on collected data with the help of distributed machine learning frameworks like Pytorch [Paszke et al., 2019], DeepSpeed [dee,], and Horovod [Sergeev and Balso, 2018].

Masking network failures: During network failure or delay in the communication with the cloud (e.g., the Historian server), the local facility will still be operational and will be able to perform the minimal function.

3.3 Cloud Layer

The **cloud** layer (i.e., Historian server) is responsible for storing the most relevant & important data from the cloudlets for analysis and business modelling. Moreover, this layer will also give a high-level overview of all the operations ongoing in the system. In the case of alarm management, professionals and experienced operators can monitor the operations of all the plants in real-time and can offer remote assistance to less experienced operators as shown in Figure 3.3.

Chapter 4

EDGEALARM MODULES OF THE REAL TIME ALARM MANAGEMENT SYSTEM

In this chapter, we explain the three proposed modules SR, OWR, and AI modules of EdgeAlarm [Gill et al., 2020a]. These three modules run in real-time at the edge to assist the operator as shown in Figure 4.1 .

4.1 Storage Reduction Module

The alarm flooding is one of the major challenges in the alarm management system, and one of the main reasons of alarm flooding is short-duration alarms such chattering alarms (alarms from the same source occurring more than 3 times in a minute) and momentary alarms (alarms whose duration of activation and deactivation is very short) [Goel et al., 2017]. The alarm flooding not only increases the operator workload but also results in missing of critical alarms by the operator [Azhar and Rissanen, 2011]. Thus, with the help of a storage reduction (SR) module, such fleeting alarms can be filtered out in real-time when the rate of alarms increases from a certain threshold (e.g., more than 10 alarms in 10 minutes). In such scenarios, the alarms whose duration is short (e.g., 20 seconds) will not be displayed to the operator. If there are no alarm floods, the alarm will appear immediately on the operator screen without any delay. This also satisfies one of the mentioned characteristics of an alarm in section 2.1.2, which is the alarm should not appear too early or too late.

Moreover, the first and major step in almost all the studies mentioned in the survey [Goel et al., 2017], which address issues like alarm flooding in alarm management systems, is to remove these short-duration alarms. Thus, such alarms are insignificant even for future analysis and filtering of such alarms at the edge will not reduce the storage space of the Historian server but also helps in the future to

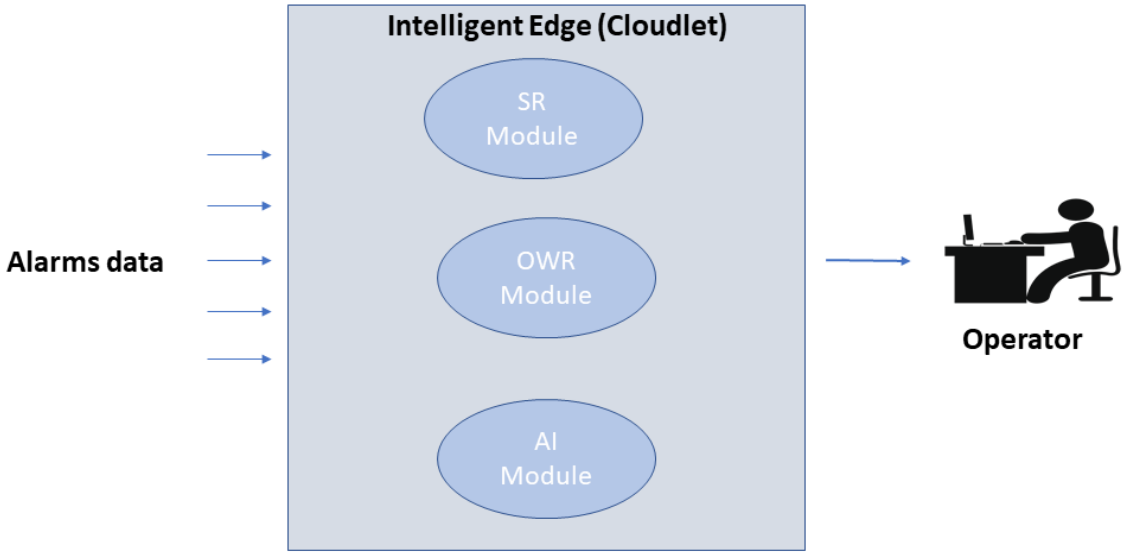

Figure 4.1: The cloudlet of edge-based alarm management system consists of SR, OWR, and AI modules to assist the operator in various ways.

analyze the data efficiently which is one of the major challenges faced in the existing system as mentioned in [Li et al., 2015] [Dorgo et al., 2018].

4.2 Operator Workload Reduction Module

Alarms can be categorized mainly into 2 groups: True alarms and Nuisance alarms. As per ISA and EEMUA instructions, the true alarm required the operator action in a fixed time interval, and a nuisance alarm does not require the operator action [Rothenberg, 2009]. If the operator does not take action on true alarms, then it will lead to inefficiency of plant performance and sometimes can cause safety hazards [Soares et al., 2016]. Thus, for an efficient and effective alarm system, it is essential to separate out the alarms which require operator actions (true alarms). If an alarm does not require any operator action, it must be ignored during abnormal situations. Thus, by identifying nuisance alarms, we cannot only reduce the significant number of alarms but also improve the productivity and focus of the operator significantly.

In most of the existing architectures based on DCS such as Foxboro [fox,], Honeywell [hon,], and Yokogawa [yok,], the operator actions are stored in a separate log with timestamps independent of the alarm data, i.e., two different logs are main-

tained: one for the operator action data and another for the alarm data. There is no one-to-one correspondence (i.e., which action is related to which alarm) between them and so true alarms cannot be determined by merely looking at both logs. The possible and main reasons of no one to one corresponds between operator's actions and alarms are that the alarms systems are considered at the end while constructing a plant, inflexibility of SCADA & DCS systems to update protocols and software, easiness of configuring alarms at the operator level and sometimes operator intentionally generate alarms to test the working condition of the system.

To address this problem, in the OWR module, we proposed a novel data-driven approach to find relationships between operator actions and alarms. The key idea is that if an alarm is triggered at a certain point in time and the operator took some action and if the same alarm is triggered again after some time, then it very likely that the same operator's action will appear again in the operator log. Thus, if we can find such repetitive patterns in the logs, we can be sure that such a relation exists, and so we can identify the true alarms. For instance, Figure 4.2 shows the timeline of the activation of alarm sources A, B, E, F, and G. The corresponding alarm actions of the operators' actions C and D are also shown. From this Figure, we can see that the alarms of the sources A, B, and F are deactivated by the operator action D and the E is deactivated by C, whenever these alarms are triggered. The source G has triggered only one alarm and the operator did not do direct action on this alarm. Thus, the alarms generated by A, B, E, and F are the true alarms while the alarms triggered by the source G are the nuisance alarms.

4.2.1 Procedure to Find True Alarms

The OWR module technique to classify true and nuisance alarms based on the operator actions and alarms data is as follows:

- Step 0: Remove the momentary alarms (i.e., whose duration is less or equal to 20 seconds) and staling alarms (i.e., alarms that remained active for 24 hours or longer). Sort the alarms log based on the start time ($startTime$) of each alarm. Similarly, sort the operator's action log by its action time

($actionTime$). In the next step, divide the alarm and action logs into m equal parts, where m can be chosen as the number of months in the data.

- Step 1: Construct m sub-graphs by Algorithm 1. A single sub-graph constructed by this algorithm from the timeline of Figure 4.2 is shown in Figure 4.3. The orange nodes are the operator's actions while green is the alarm nodes (i.e., alarm sources). The edge between an operator action node to an alarm source node means that an operator did some action after the appearance of an alarm, and the operator $actionTime$ is greater than the $startTime$ of the alarm and less than its $endTime$. The count on a node represents the number of times that sensor (i.e., source) has triggered an alarm (e.g., node A triggered an alarm 2 times). Similarly, the edge weight represents that how many times the operator action time ($actionTime$) was in between the starting time ($startTime$) and ending time ($endTime$) of the alarm (e.g., the weight of edge $D \rightarrow F$ in sub-graph 1 is 2). Figure 4.4, in its step 1, shows the 2 sub-graphs ($m = 2$) from the data.

Algorithm 1: Construct a directed graph G from alarms and operator actions.

Input: An alarms list $alarmsL$ and operator's action list $actionsL$.

Output: A directed Graph G

```
1  G                                                    // An empty directed graph
2  for each alarm al ∈ alarmsL do
3      stime = al.startTime
4      etime = al.endTime
5      for each action act ∈ actionsL do
6          act_time = act.actionTime
7          if stime < act_time ≤ etime then
8              if act ∈ G.nodes then
9                  G.increaseNodeCount(act, 1)
10             else
11                 G.addNode(act)
12             if al ∈ G.nodes then
13                 G.increaseNodeCount(al, 1)
14             else
15                 G.addNode(al)
16             e = edge(act, al)
17             if e ∈ G.edges then
18                 G.increaseEdgeWeight(e, 1)
19             else
20                 G.addEdge(e)
```

- Step 2: After constructing the m sub-graphs, take their intersections with the help of Algorithm 2. It takes a list of m sub-graphs and an intersection threshold parameter γ as arguments, where $1 \leq \gamma \leq m$. The optimal value for γ is greater or at least equal to $m/2 + 1$ i.e., the majority of the sub-

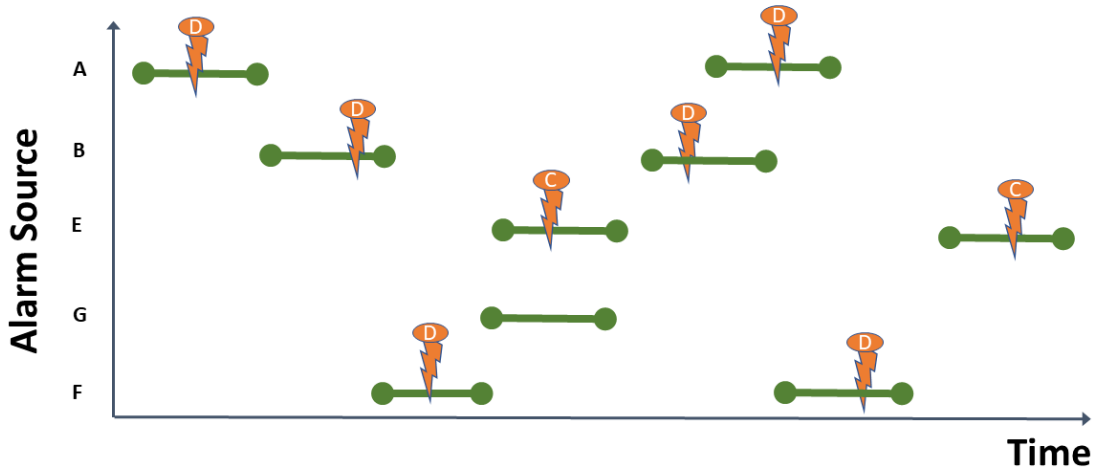

Figure 4.2: Operator action timeline.

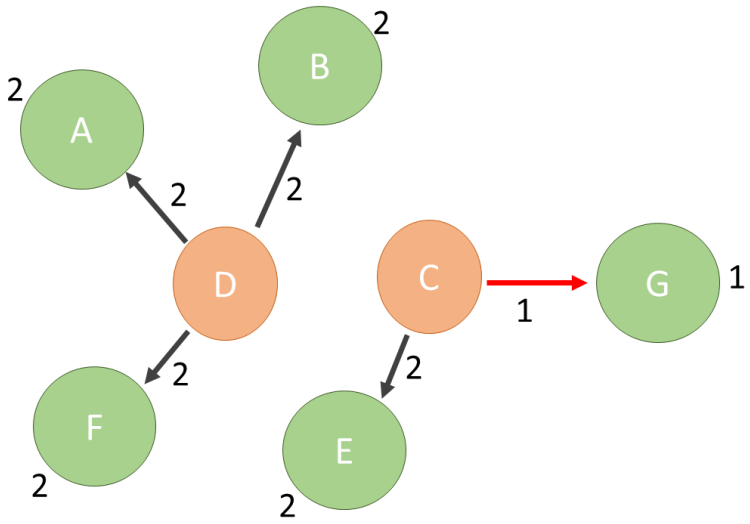

Figure 4.3: Graph from operator action timeline.

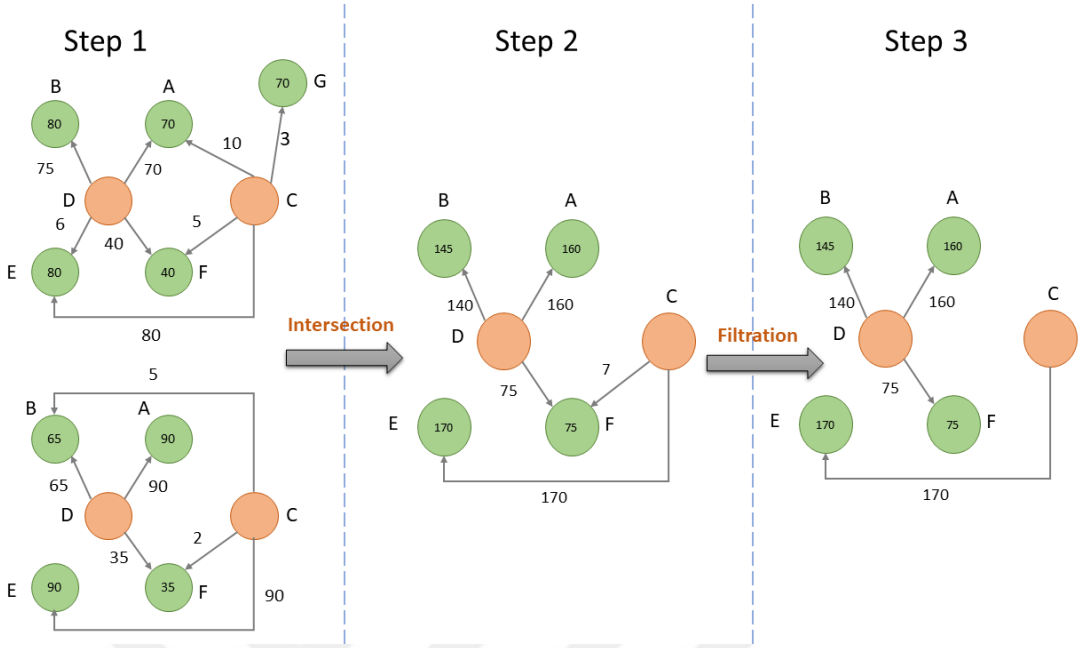

Figure 4.4: Relating operator actions with alarms.

graphs should have the same action-alarm relationships. The interaction nodes now contain the sum of corresponding nodes of sub-graphs and the same is the case with edges as shown in step 2 of Figure 4.4. If an edge (action-alarm relationship) is not present in at least γ times in the graph after the intersection, it will be evicted and such edge is called an unnecessary edge (action-alarm relationship). For instance, as shown in step 2 of Figure 4.4, the edges $C \rightarrow A$, $C \rightarrow G$, and $D \rightarrow E$ from the first sub-graph and the edge $C \rightarrow B$ in the second sub-graph are removed in step 2.

Algorithm 2: Intersection of graphs

Input: A list L of directed graphs and an intersection threshold γ

Output: A directed Graph G

```
1  G                                                    // An empty directed graph
2  edge_dict = {}                                       // A Key-Value store
3  for each graph g ∈ L do
4      for each edge e ∈ g.edges do
5          if e ∈ edge_dict.keys then
6              edge_dict[e] += 1
7          else
8              edge_dict[e] = 0
9  for each key e ∈ edge_dict.keys do
10     if edge_dict[e] ≥ γ then
11         s = e.source
12         t = e.target
13         if s ∉ G.nodes then
14             G.addNode(s)
15         if t ∉ G.nodes then
16             G.addNode(t)
17         G.addEdge(e)
18 for each node n ∈ G do
19     for each graph g ∈ L do
20         if node n ∈ g.nodes then
21             count = g.getNodeCount(n)
22             G.increaseNodeCount(n, count)
23 for each edge e ∈ G do
24     for each graph g ∈ L do
25         if edge e ∈ g.edges then
26             w = g.getEdgeWeight(e)
27             G.increaseEdgeWeight(e, w)
```

- Step 3 (Optional): Pruning of some of the edge in the final graph with the help of Algorithm 3. This procedure takes a graph G obtained from the previous step and a prune factor α as arguments, where $1 \leq \alpha \leq 2$. It will prune those edges for which $(weight \times \alpha)$ is less than the count of the incident node (i.e., target alarm).

  In other words, it will filter out those edges which are not significantly contributing to the deactivation of the alarm. As it is possible that some edges between operator actions and alarms in the graph are not real relations, because of the possibility that some alarms trigger quite often and the operators are also taking some actions on some other alarms at the same, it is quite possible that they will appear in γ number of sub-graphs as outliers, and will not be removed at the intersection step. Thus, we need to remove such edges to capture the real relations between operator actions and alarms. For instance, consider the edge $C \rightarrow F$ in both sub-graphs of Figure 4.4 in step 1. The weight of the edge is quite small as compared to the count of node F in both sub-graphs but since it was present in both graphs, it was also present after the intersection step. Thus, such edges should be evicted to remove unnecessary operator alarm relations. If the data is massive, then this step can be skipped as such edges will be filtered out at step 2.

Algorithm 3: Pruning of edges in a directed graph.

Input: A directed graph G and a prune factor α

Output: A directed Graph G

```
1  for each edge e ∈ G.edges do
2      source = e.source
3      target = e.target
4      weight = e.weight
5      count = target.count
6      if (weight × α) < count then
7          G.removeEdge(e)
8  for each node n ∈ G.nodes do
9      in = n.in_degree
10     out = n.out_degree
11     if (in + out) equals to zero then
12         G.removeNode(n)
```

The final graph obtained after this step contains only true alarms. The nodes A, B, F, and E of the final graph in Figure 4.4 represent the true alarms, while the nodes which are removed at the intersection such as G are the nuisance alarms.

The nuisance alarms can be further reduced by finding parent-child relationships among these alarms. The alarms which are activated within a reasonable time after the activation of another alarm (i.e., parent alarm) and turned off before it or right after it should be considered as child alarms. Figure 4.5 A shows the activation and deactivation of alarms $S1$, $S2$, $S7$, and $S9$. From the figure, we can see that whenever $S9$ is turned on, its activating $S1$, which in turn activating $S2$ and $S7$. Similarly, before or right after the deactivation of $S1$, the alarm sources $S2$ and $S7$ are also deactivated. Therefore, $S2$ and $S7$ are the child alarms of $S1$. The source $S9$ is the parent alarm of $S1$. Thus, such alarms can be grouped and represented by a single nuisance alarm as they represent the same process change to reduce the operator workload.

4.2.2 Procedure to Group Nuisance Alarms

To reduce the number of nuisance alarms by finding parent-child relationships, we proposed another technique in the OWR module which is as follows:

- Step 0: After finding the nuisance alarms with the help of the previous technique, sort these nuisance alarms based on the $startTime$ of each alarm. Divide the nuisance alarms into m equal parts, where m is equal to the number of months in the data.

- Step 1: Construct the m sub-graphs with the help of Algorithm 4. The algorithm takes the nuisance alarms list and 2 constants: $\delta 1$ and $\delta 2$. The child alarm must be active within $\delta 1$ duration and similarly, either child alarm should be turned off after turning off its parent alarm within $\delta 2$. For instance, in Figure 4.5 B, there is only a single edge from $S9$ to $S1$ because it is the only alarm that is activating within $\delta 1$ duration. The algorithm adds an edge between two nodes which has a parent-child relationship and update/increase the edge weight by one as shown in Figure 4.5 B.

Algorithm 4: To find parent-child relationships.

Input: A sorted list $alarmsL$ of alarms by start time and duration thresholds $\delta 1$ and $\delta 2$, where $\delta 1$ is the activation duration between the previous alarm to next alarm and $\delta 2$ is the duration of deactivation from previous alarm deactivation to next alarm deactivation.

Output: A directed Graph G

```
1  G                                                    // An empty directed graph
2  for i ∈ alarmsL.length do
3      prev = alarmsL[i]
4      for j ∈ (i, alarmsL.length) do
5          next = alarmsL[j]
6          Δ = next.startTime − prev.startTime
7          if Δ > δ1 then
8              break
9          p_etime = prev.endTime
10         n_etime = next.endTime
11         Δ = n_etime − p_etime
12         if n_etime ≤ p_etime or Δ < δ2 then
13             if prev ∈ G.nodes then
14                 G.increaseNodeCount(prev, 1)
15             else
16                 G.addNode(prev)
17             if next ∈ G.nodes then
18                 G.increaseNodeCount(next, 1)
19             else
20                 G.addNode(next)
21             e = edge(prev, next)
22             if e ∈ G.edges then
23                 G.increaseEdgeWeight(e, 1)
24             else
25                 G.addEdge(e)
```

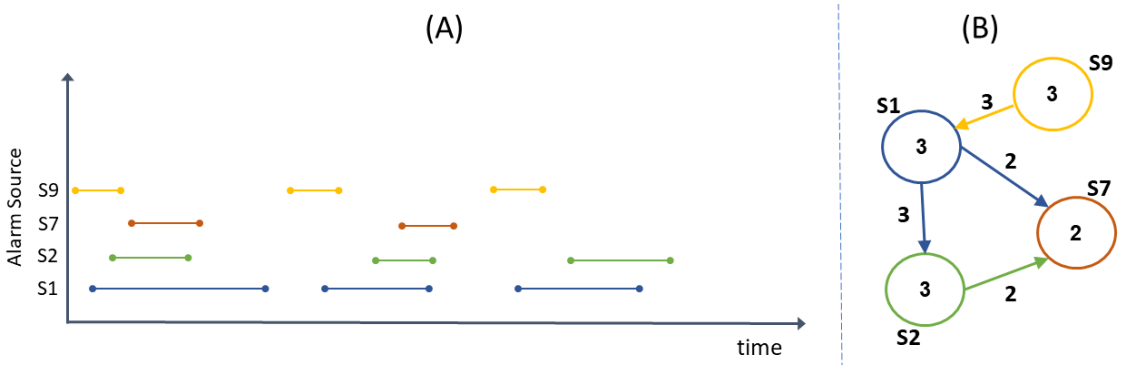

Figure 4.5: Grouping of Nuisance Alarms. *S*9 is the parent alarm of all the other alarms and hence such alarms should be considered as a single alarm.

- Step 2: Similar to step 2 of 4.2.1, after constructing m sub-graphs, now take the intersection of these sub-graphs with the help of Algorithm 2 to ignore those parent-child relationships which are not appearing in at least γ number of sub-graphs.

- Step 3 (Optional): Similar to step 3 of 4.2.1, prune the edges by using Algorithm 3. It will filter out those edges which are not significantly contributing to the activation of the target (i.e., child) alarm. As it is possible that some parent-child relationships in the graph are not true edges because of the possibility that some alarms trigger quite often and it is quite possible that they will appear in γ number of sub-graphs as outliers. Thus, we need to remove such edges to capture true parent-child relationships. Now, the final graph will have all the parent-child relationships and such alarms will be grouped to further reduce the number of nuisance alarms.

With the help of the OWR module, now the alarms will be categorized into two categories, true alarms and nuisance alarms in real-time on the operator screen at the edge layer to assist the operator. The nuisance alarms will be further reduced by finding parent-child relationships and grouping such alarms. During the critical events, the operator will only focus on the true alarms and will ignore the nuisance alarms, which will significantly improve the operator's performance and reduce his/her workload.

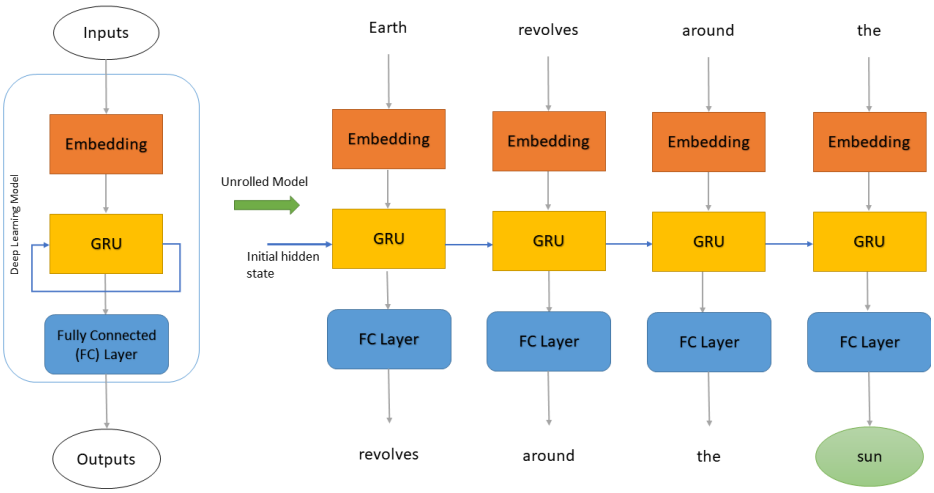

Figure 4.6: An example of a RNN for language modeling. In the unrolled version of the same model, an example of a prediction of the next word is shown.

4.3 Artificial Intelligence Assistant Module

Recurrent Neural Networks (RNNs) [Yu et al., 2019] have shown a lot of potential in many natural language processing (NLP) and sequence learning tasks. The main idea behind the sequential learning is that they make use of sequential information. In traditional feed-forward neural networks such as Convolutional Neural Networks (CNNs), it is assumed that all the inputs are independent of each other, e.g., in an image classification task, the pixels of an image are independent of each other. However, this approach is not valid for sequence learning tasks. For instance, if you want to predict the next word in a sequence (sentence), you need the prior information (i.e., previous words in the sentence) to do so. RNNs have the ability to remember previous states (information), i.e., RNNs have a "memory" in the form of hidden states which store information about what has been processed so far. At each input of the sequence, the model not only takes the current input but also remembers the preceding information. Like the human way of processing sequential information, this allows the model to learn long-term dependencies in the sequence, i.e., it considers the entire context when making a prediction. A typical RNN, based on the gated recurrent unit (GRU) [Yu et al., 2019], is shown in Figure 4.6, which consists of one embedding layer, one GRU layer, and one fully connected (FC) layer.

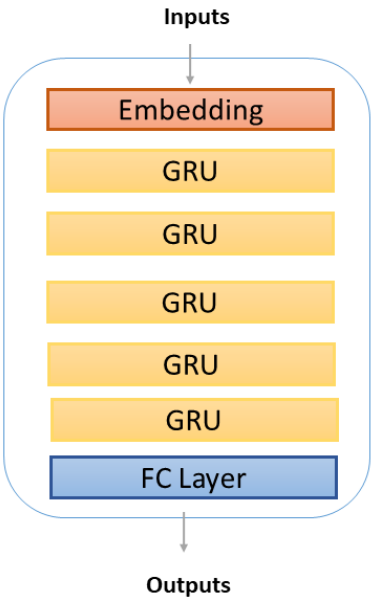

Figure 4.7: The proposed recurrent neural network (RNN) consists of an embedding layer, three GRU layers that are stacked on top of each other, and finally a fully connected linear layer to output the predictions.

The RNNs can be classified into different categories based on their input and output, such as one-to-one, one-to-many (image captioning), many-to-one (sentiment analysis), and many-to-many (machine translation). For instance, to predict the next word in a sequence (sentence), the RNN (many-to-one) takes input one word at a time and then predicts the next word at the last input as shown in the unrolled version of the model in Figure 4.6.

Prediction of the next alarm can also be modelled as a sequence learning task: given a sorted sequence of alarms based on their start time, the RNN can predict the future alarm. In this module, we proposed a RNN architecture consists of one embedding layer, three GRU layers, and one fully connected linear layer as shown in Figure 4.7 to predict the upcoming alarms.

We implemented the RNNs network with GRU by using the deep learning framework Pytorch. Pytorch library has built-in methods to define and train neural network architectures. As shown in Figure 4.8, the input to the network is a sorted sequence of alarms ($S7$,$S2$,$S40$,$S4$) based on the starting time and the target ($S20$)

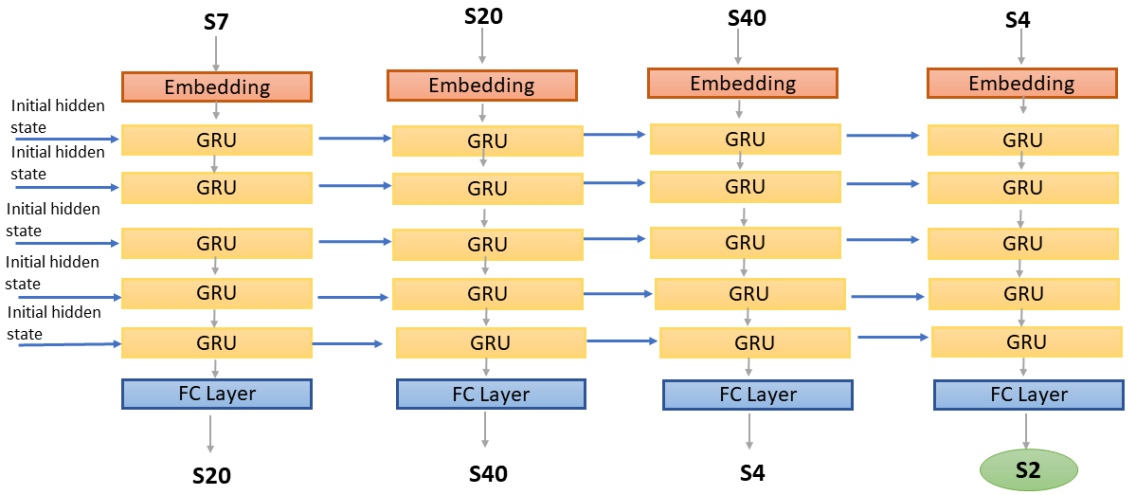

Figure 4.8: Given the sequence $S7$, $S20$, $S40$, and $S4$ the prediction of the next alarm $S2$ is shown.

is the next alarm based on the prior alarms. Initially, $S7$ is the first input of the sequence whose output is $S2$. Similarly, the next input is $S2$ and whose output is $S40$ and so on. Finally, $S4$ is the last input to the model and it will predict $S20$ as the future alarm. By correctly predicting future alarms in real-time at the edge with the help of the AI module, the operator may avert abnormal situations by taking corrective action or prepare for such situations in advance.

Chapter 5

PERFORMANCE RESULTS AND ANALYSIS

This chapter explains the results and performance of the proposed edge modules for the edge-based alarm management system.

5.1 Dataset and Performance Metrics

We analyze the 13 months (March 2019 - March 2020) alarms & operator actions data of the alarm management system of a Tüpraş [tup,] oil refinery plant. The data contains 12.54 million alarms triggered by 1452 number of sources and approximately 151 thousand of operator actions. Each alarm in the data has following fields: *SourceName*, *StartTime*, *EndTime*, and *Condition*. The different conditions present in the alarm data are shown in Figure 5.1. For instance, IOP and IOP- conditions represent the communications alarms and from the figure, we can see that there are millions of alarms related to these conditions. We measured the performance of the proposed edge-based alarm management system based on the following metrics: number of true alarms & nuisance alarms, storage space utilization, and accuracy of prediction of upcoming alarms.

5.2 Storage Space Reduction

According to ISA, the alarms whose duration is short (e.g., less than or equal to 20 seconds) must be ignored as such alarms do not require operator action and do not give any useful information to operators. From figure 5.2, we can see that the raw alarms are in hundreds of thousands and by ignoring the momentary alarms with the help of the SR module, the number of alarms is reduced to just a few thousand. For instance, in January 2020, 1.29 million were generated, and if the momentary alarms are filtered out, the number of alarms is significantly decreased

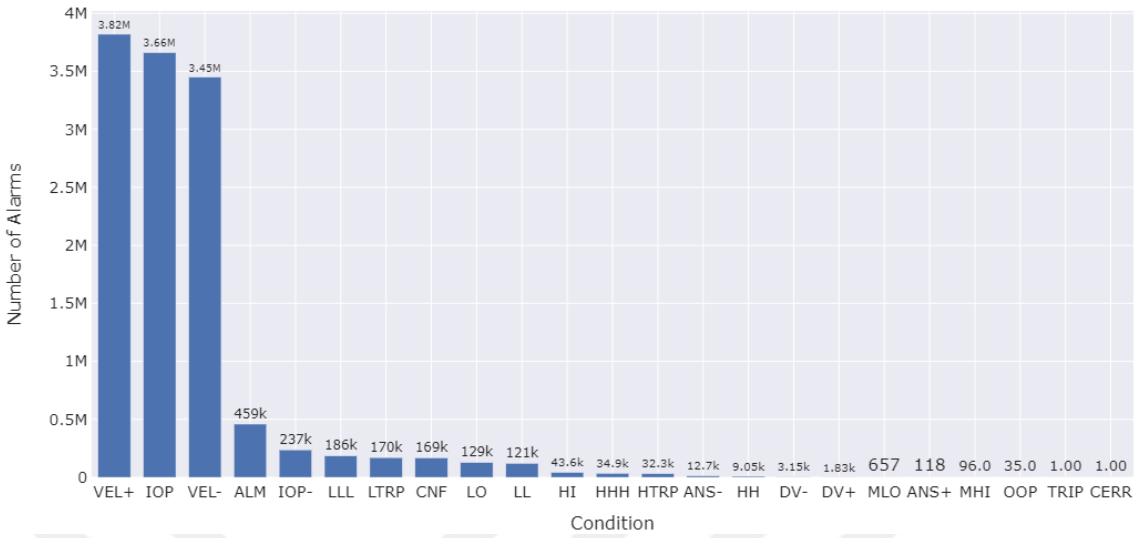

Figure 5.1: Different types/conditions in alarms.

to 92.5 thousand with 10 seconds filter and 43.8 thousand with 20 seconds filter. Ignoring such momentary alarms in real-time at the edge will not only just reduce the operator workload but also save the storage space of the Historian server as shown in Figure 5.3.

5.3 Operator's Workload Reduction

By finding the true and nuisance alarms with the help of the OWR module, a significant amount of operator workload can be reduced. To find true and nuisance alarms, we removed the momentary alarms because such alarms do not require the operator's action as the operator cannot do action within 20 seconds. Additionally, we removed the communication alarms (because they related to connectivity problems) and the alarms which are triggered less than 20 times in all 13 months data. After the preprocessing step, we have 399 unique source tags and a total of 222.384 thousand alarms. We classify the true alarms and nuisance alarms with the help of the technique mentioned in section 4.2 and found out that out of 399 sources only 177 need the operator action while 222 of the sources do not require the operator action at all. True alarm sources generated 84.752 thousand alarms while nuisance sources generated 137.632 thousand alarms. Figure 5.4 shows the number of alarms

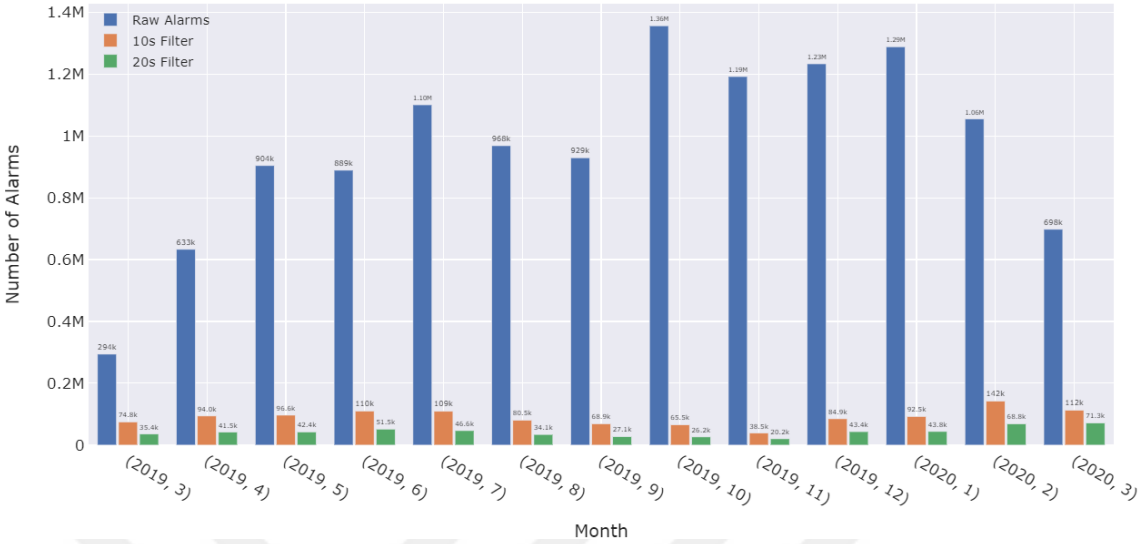

Figure 5.2: The alarms whose duration is very short can be filtered at the edge because such alarms do not help the operator or expert to determine the problem in the system. This plot shows that there are a significant number of momentary alarms, and by removing such alarms the number of alarms is significantly reduced to a few thousand. For instance, with 10 seconds filter, in October, the number of alarms is reduced to 65.5 thousand from 1.36 million.

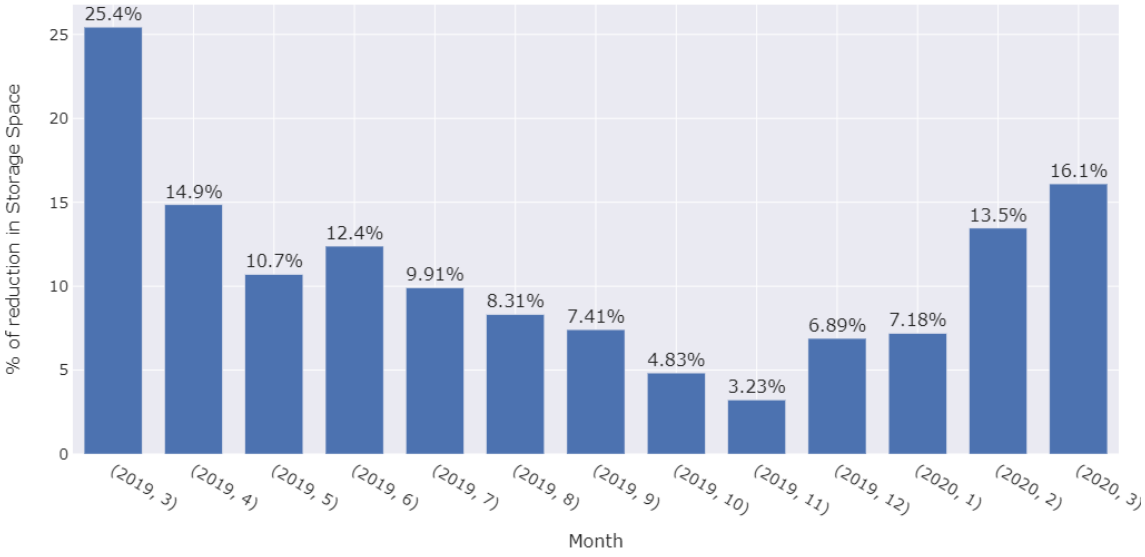

Figure 5.3: By filtering momentary alarms at the edge, the storage space of the Historian server can be saved. This plot shows the % of reduction in the storage space of the Historian server with 10 seconds filter. On average, 10.82% of overall storage space utilization will decrease.

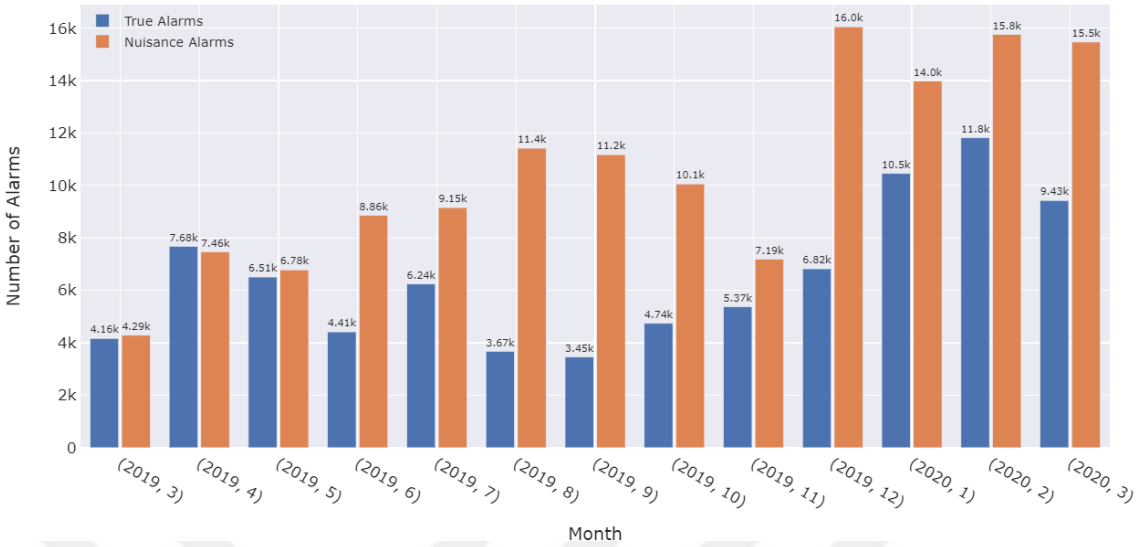

Figure 5.4: The alarms which do not require operator action are nuisance alarms, while the alarms which require operator action are true alarms. The graph shows that in each month there are roughly half or sometimes more than half the number of alarms that do not require operator action at all.

generated by true and nuisance sources in each month of the data. As we can see from the figure, the number of nuisance alarms is almost equal or sometimes significantly higher than the true alarms, which means that by real time classifying true and nuisance alarms, we can approximately reduce 50% or sometimes more of the operator's workload. Additionally, we further reduce the number of nuisance alarms by finding the parent-child relationships with the help of the proposed technique in the OWR module. With the grouping of parent-child alarms, we reduced the number of nuisance sources from 222 to 184 and the number of alarms from 137.632 thousand to 113.422 thousand. Figure 5.5 shows the percentage of reduction of nuisance alarms in each month. For instance, in September 2019, the number of nuisance alarms was 11.2 thousand and by grouping they were further reduced by 32.1%, which is approximately 7.5 thousand.

The nuisance alarms will only appear as alert to the operator. Thus, by real-time filtration of nuisance alarms at the edge, we can roughly reduce the operator workload by 50% or even more, and now the operator will only focus on the true alarms during the critical scenarios. Similarly, with the grouping of nuisance alarms,

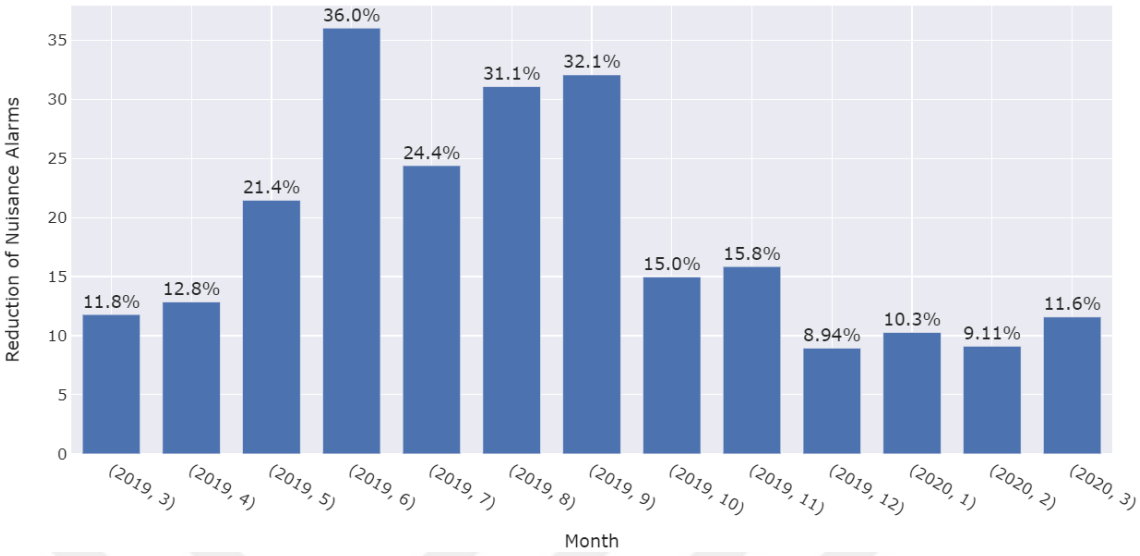

Figure 5.5: By finding the parent-child relationship, the number of nuisance alarms can be further reduced. This graph shows the percentage of reduction of nuisance alarms each month.

the number of nuisance alarms is further decreased approximately by 18%, which further decreases the number of alerts in the alarm management system.

5.4 Prediction of Upcoming Alarms

Like in subsection 5.3, to predict future alarms with the AI module, the first step is to remove the momentary alarms, staling alarms, the communication alarms, and the alarms sources which are not triggered more than 1500 times. Additionally, we only considered the alarms sources which were triggered in each month. After these preprocessing steps, the alarms were sorted based on their starting time and after that, they are divided into sequences of 15 minutes intervals.

The number of hidden units in each GRU and fully connected layer are 1024. The the size of each embedding vector is 512 and the batch size is 64. We used negative log likelihood loss (NLLLoss [pyt, a]) and the Adam optimizer [pyt, b] to optimize loss function and ReduceLROnPlateau [pyt, b] optimization algorithm to reduce learning rate when the validation loss is stopped improving. The model was trained for 400 epochs and the initial learning rate was set to 0.000001 and halved

Table 5.1: Hyperparameters and Architectural detail of the proposed RNN model.

Vocabulary Size	21
Embedding Vector Size	512
Number of GRU layers	5
Number of features in GRU hidden state	1024
Number of features in FC hidden state	1024
Batch Size	64
Number of Epochs	400
Loss Function	Negative Log Likelihood
Loss Optimizer	Adam
Initial learning Rate	0.000001

after 6 epochs if the validation loss was not improving. The hyperparameters detail is also given in table 5.1. We split the data into a training set (85%) and a validation set (15%). The validation loss of the model is shown in Figure 5.6. As we can see from the figure, during initial epochs the loss was high, which means that the model was less accurate and after 150 epochs the model became almost steady, i.e., the model was not learning more.

The validation set accuracy is shown in Figure 5.7, which shows the prediction of the alarms per sensor. For instance, the alarms triggered by the sensor $S12$ were predicted by 96% of the time successfully. Similarly, the alarms triggered by $S16$ were predicted accurately 75.09% of the time. From the figure, we can see that the majority of the sources have prediction accuracy more than 55% and a few alarms sources have accuracy lesser than 50% due to fewer alarm sequences (i.e., training instances) in the training data.

Figure 5.8 shows the confusion matrix for a detailed analysis of the predictions. It displays the RNN model's overall efficiency. The number of points at which the expected label is equal to the actual label is indicated by the diagonal of confusion

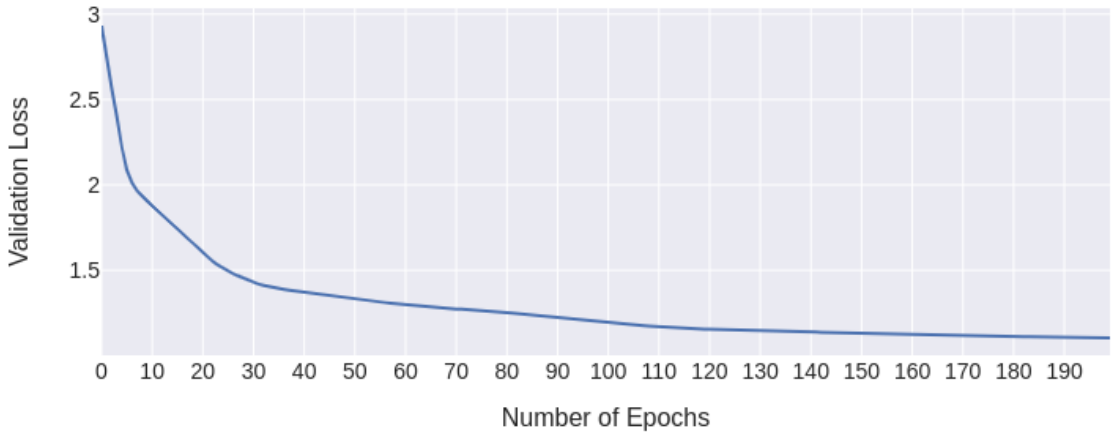

Figure 5.6: The lower the validation loss, the better the model will be. The validation loss is decreasing per epoch and after 150 epochs the model is not learning much.

matrix. Those that are mislabeled by the AI module are the off-diagonal points.

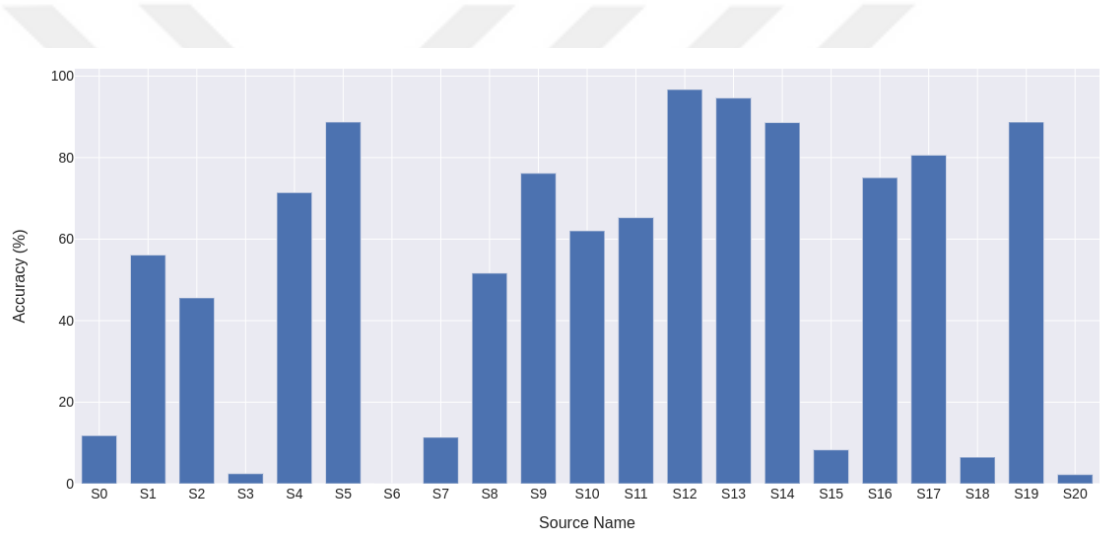

Figure 5.7: The proposed RNN architecture can successfully predict most of the future alarms with more than 55% accuracy per source. However, as we can see from the plot, the prediction is even higher than 90% for some sources.

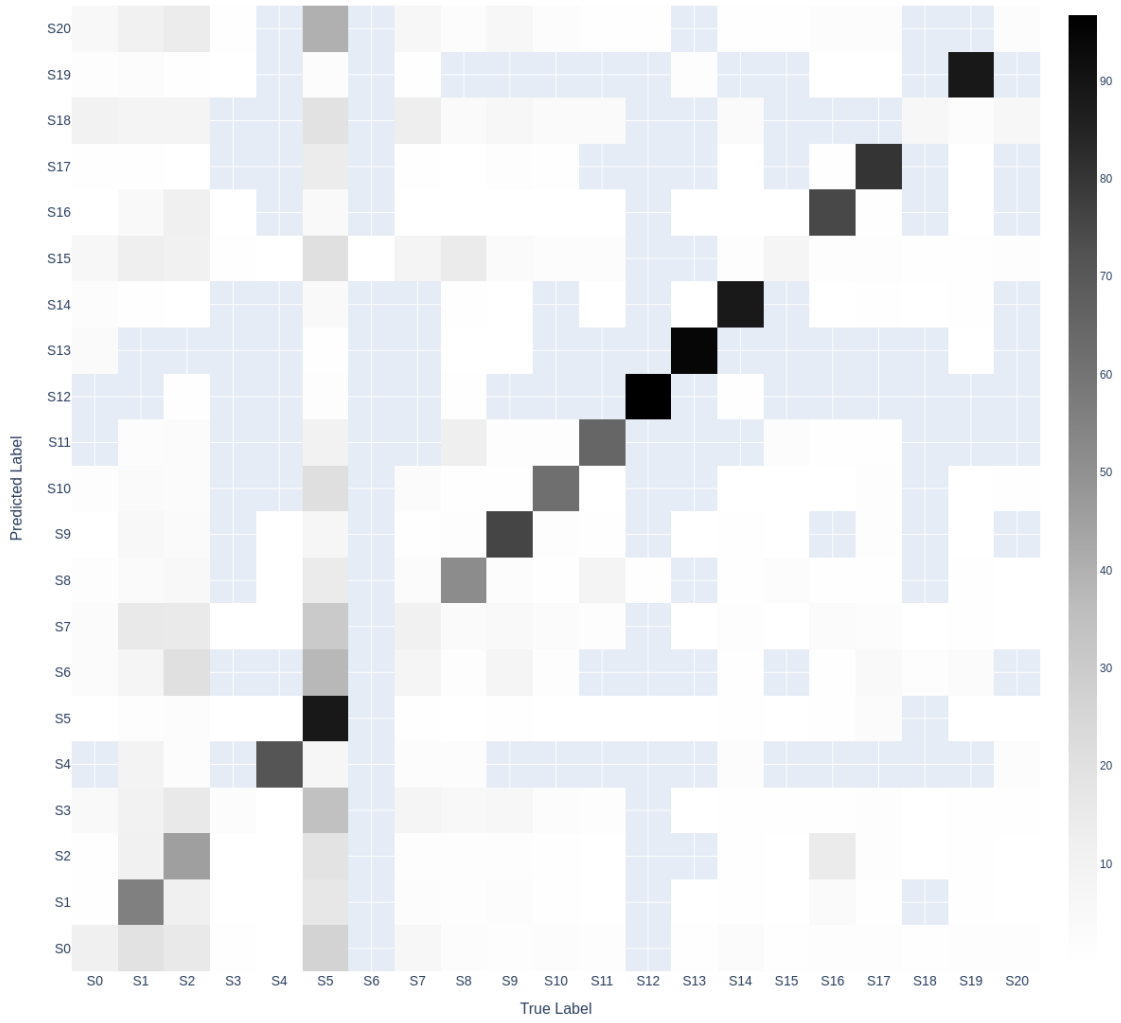

Figure 5.8: Confusion matrix shows the quality of the proposed RNN architecture. The diagonal elements represent the number of points for which the predicted label is equal to the actual/true label. The off-diagonal elements are the ones that are mislabelled by the AI module.

Chapter 6

CONCLUSION AND FUTURE WORK

In this thesis, we have proposed a unique alarm management system based on the edge computing paradigm. The SR module addresses the problem of storage utilization. In the OWR module, the novel techniques to reduce the operator workload are discussed. Lastly, in the AI module, we proposed a RNN architecture that consists of an embedding layer, three GRU layers stacked on top of each, and finally a fully connected layer to predict the upcoming alarms. We measured the performance of the proposed modules on 13 months of data gathered from the alarm management system of a Tüpraş oil refinery plant. We have shown that on average approximately 10% of storage space utilization can be reduced by filtering the momentary alarms with the SR module at the edge. Based on our analysis, we also have shown that by filtering nuisance alarms with the help of the OWR module, the operator's workload can be drastically reduced by approximately 50% or more, which is a significant improvement. Furthermore, with the AI module, we forecasted the majority of future alarms in real-time with more than 55% accuracy (per source) and more than 90% for some alarm sources. It can help the operator to avert or prepare for the abnormal situations in advance.

In the future, we plan to extend the proposed architecture as an active learning framework. For instance, if the AI module predicts an upcoming alarm incorrectly, then that alarm will be forward back to the AI module to retrain its model to improve its accuracy. Secondly, we will make use of transfer learning in the AI module to improve the accuracy and performance of the proposed architecture. Finally, we will deploy the proposed systems on multiple plants of the Tüpraş oil refinery.

BIBLIOGRAPHY

[fox,] Distributed control system - ecostruxure foxboro dcs — schneider electric australia. https://www.se.com/au/en/work/products/industrial-automation-control/foxboro-dcs. (Accessed on 09/08/2020).

[yok,] Distributed control system (dcs) — yokogawa america. https://www.yokogawa.com/us/solutions/products-platforms/control-system/distributed-control-systems-dcs/. (Accessed on 09/08/2020).

[hon,] Honeywell distributed control system (dcs) - experion lx - dgfg.nl. https://www.dgfg.eu/products/honeywell-distributed-control-system-dcs/. (Accessed on 09/08/2020).

[dee,] microsoft/deepspeed: Deepspeed is a deep learning optimization library that makes distributed training easy, efficient, and effective. https://github.com/microsoft/DeepSpeed. (Accessed on 09/08/2020).

[pyt, a] Nllloss — pytorch 1.6.0 documentation. https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html. (Accessed on 09/29/2020).

[tup,] Refineries - tüpraş. https://www.tupras.com.tr/en/rafineries. (Accessed on 09/08/2020).

[pyt, b] torch.optim — pytorch 1.6.0 documentation. https://pytorch.org/docs/stable/optim.html. (Accessed on 09/29/2020).

[edg,] warisgill/edge-alarm. https://github.com/warisgill/edge-alarm. (Accessed on 09/07/2020).

[Aalsalem et al., 2018] Aalsalem, M. Y., Khan, W. Z., Gharibi, W., Khan, M. K., and Arshad, Q. (2018). Wireless sensor networks in oil and gas industry: Recent

advances, taxonomy, requirements, and open challenges. *Journal of Network and Computer Applications*, 113:87–97.

[Abeshu and Chilamkurti, 2018] Abeshu, A. and Chilamkurti, N. (2018). Deep learning: the frontier for distributed attack detection in fog-to-things computing. *IEEE Communications Magazine*, 56(2):169–175.

[Alnoman et al., 2019] Alnoman, A., Sharma, S. K., Ejaz, W., and Anpalagan, A. (2019). Emerging edge computing technologies for distributed iot systems. *IEEE Network*, 33(6):140–147.

[Azhar and Rissanen, 2011] Azhar, S. B. and Rissanen, M. J. (2011). Evaluation of parallel coordinates for interactive alarm filtering. In *15th International Conference on Information Visualisation*, pages 102–109. IEEE.

[Azimi et al., 2017] Azimi, I., Anzanpour, A., Rahmani, A. M., Pahikkala, T., Levorato, M., Liljeberg, P., and Dutt, N. (2017). Hich: Hierarchical fog-assisted computing architecture for healthcare iot. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):174.

[Azimi et al., 2018] Azimi, I., Takalo-Mattila, J., Anzanpour, A., Rahmani, A. M., Soininen, J.-P., and Liljeberg, P. (2018). Empowering healthcare iot systems with hierarchical edge-based deep learning. In *IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 63–68. IEEE.

[Bergquist et al., 2003] Bergquist, T., Ahnlund, J., and Larsson, J. E. (2003). Alarm reduction in industrial process control. In *EFTA 2003. IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 03TH8696)*, volume 2, pages 58–65. IEEE.

[Boyer, 2009] Boyer, S. A. (2009). *SCADA: supervisory control and data acquisition*. International Society of Automation.

[Bransby, 2001] Bransby, M. (2001). Design of alarm systems. *IEE CONTROL ENGINEERING SERIES*, pages 207–221.

[Bullemer et al., 2011] Bullemer, P. T., Tolsma, M., Reising, D. V. C., and Laberge, J. C. (2011). Towards improving operator alarm flood responses: Alternative alarm presentation techniques. *Abnormal Situation Management Consortium*.

[Diro and Chilamkurti, 2018] Diro, A. A. and Chilamkurti, N. (2018). Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, 82:761–768.

[Dorgo et al., 2018] Dorgo, G., Varga, K., and Abonyi, J. (2018). Hierarchical frequent sequence mining algorithm for the analysis of alarm cascades in chemical processes. *IEEE Access*, 6:50197–50216.

[Dubey et al., 2015] Dubey, H., Yang, J., Constant, N., Amiri, A. M., Yang, Q., and Makodiya, K. (2015). Fog data: Enhancing telehealth big data through fog computing. In *Proceedings of the ASE bigdata & social informatics*, page 14. ACM.

[Dunkels et al., 2004] Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki-a lightweight and flexible operating system for tiny networked sensors. In *29th annual IEEE international conference on local computer networks*, pages 455–462. IEEE.

[Ghosh and Grolinger, 2019] Ghosh, A. M. and Grolinger, K. (2019). Deep learning: Edge-cloud data analytics for iot. In *IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, pages 1–7. IEEE.

[Gill et al., 2019] Gill, W., Özkasap, Ö., and Gürsoy, A. (2019). A reliable edge computing architecture for petrol industry. In *6th ACM Celebration of Women in Computing: womENcourage*. ACM.

[Gill et al., 2020a] Gill, W., Özkasap, Ö., and Gürsoy, A. (2020a). Edgealarm: Edge assisted real-time and intelligent alarm management system for oil refinery. In *Future Generation Computer Systems (To be Submitted)*.

[Gill et al., 2020b] Gill, W., Özkasap, Ö., and Gürsoy, A. (2020b). Intelligent edge computing: State-of-the-art techniques and applications. In *28th Signal Processing and Communications Applications Conference (SIU)*. IEEE (Accepted).

[Gill et al., 2020c] Gill, W., Özkasap, Ö., and Gürsoy, A. (2020c). A novel edge computing architecture for intelligent services in the oil refinery. In *Technical Report*.

[Goel et al., 2017] Goel, P., Datta, A., and Mannan, M. S. (2017). Industrial alarm systems: Challenges and opportunities. *Journal of Loss Prevention in the Process Industries*, 50:23–36.

[Hollifield, 2010] Hollifield, B. (2010). Understanding and applying the ansi/isa 18.2 alarm management standard. *PAS Inc., Houston*.

[Hollifield and Habibi, 2011] Hollifield, B. R. and Habibi, E. (2011). *Alarm management: A comprehensive guide: Practical and proven methods to optimize the performance of alarm management systems*. Isa.

[Hossain and Muhammad, 2016] Hossain, M. S. and Muhammad, G. (2016). Cloud-assisted industrial internet of things (iiot)–enabled framework for health monitoring. *Computer Networks*, 101:192–202.

[Janjua et al., 2019] Janjua, Z. H., Vecchio, M., Antonini, M., and Antonelli, F. (2019). Irese: An intelligent rare-event detection system using unsupervised learning on the iot edge. *Engineering Applications of Artificial Intelligence*, 84:41–50.

[Jovašević-Stojanović et al., 2015] Jovašević-Stojanović, M., Bartonova, A., Topalović, D., Lazović, I., Pokrić, B., and Ristovski, Z. (2015). On the use of

small and cheaper sensors and devices for indicative citizen-based monitoring of respirable particulate matter. *Environmental Pollution*, 206:696–704.

[Khan et al., 2019] Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., and Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235.

[La et al., 2019] La, Q. D., Ngo, M. V., Dinh, T. Q., Quek, T. Q., and Shin, H. (2019). Enabling intelligence in fog computing to achieve energy and latency reduction. *Digital Communications and Networks*, 5(1):3–9.

[Laberge et al., 2014] Laberge, J. C., Bullemer, P., Tolsma, M., and Dal Vernon, C. R. (2014). Addressing alarm flood situations in the process industries through alarm summary display design and alarm response strategy. *International Journal of Industrial Ergonomics*, 44(3):395–406.

[Levis et al., 2005] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., et al. (2005). Tinyos: An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer.

[Li et al., 2015] Li, D., Hu, J., Wang, H., and Huang, W. (2015). A distributed parallel alarm management strategy for alarm reduction in chemical plants. *Journal of Process Control*, 34:117–125.

[Li et al., 2018] Li, L., Ota, K., and Dong, M. (2018). Deep learning for smart industry: Efficient manufacture inspection system with fog computing. *IEEE Transactions on Industrial Informatics*, 14(10):4665–4673.

[Lin et al., 2016] Lin, C.-C., Deng, D.-J., Chen, Z.-Y., and Chen, K.-C. (2016). Key design of driving industry 4.0: Joint energy-efficient deployment and scheduling in group-based industrial wireless sensor networks. *IEEE Communications Magazine*, 54(10):46–52.

[Memon and Maheswaran, 2019] Memon, S. and Maheswaran, M. (2019). Using machine learning for handover optimization in vehicular fog computing. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 182–190. ACM.

[Nimmo, 2005] Nimmo, I. (2005). Alarm overload. *Chemical Processing*, 68(1):28–33.

[Pariyani et al., 2010] Pariyani, A., Seider, W. D., Oktem, U. G., and Soroush, M. (2010). Incidents investigation and dynamic analysis of large alarm databases in chemical plants: A fluidized-catalytic-cracking unit case study. *Industrial & Engineering Chemistry Research*, 49(17):8062–8079.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037.

[Queralta et al., 2019] Queralta, J. P., Gia, T., Tenhunen, H., and Westerlund, T. (2019). Edge-ai in lora-based health monitoring: fall detection system with fog computing and lstm recurrent neural networks. In *42nd International Conference on Telecommunications and Signal Processing (TSP)*, pages 601–604. IEEE.

[Ray et al., 2019] Ray, P. P., Dash, D., and De, D. (2019). Edge computing for internet of things: A survey, e-healthcare case study and future direction. *Journal of Network and Computer Applications*, 140:1–22.

[Rothenberg, 2009] Rothenberg, D. H. (2009). *Alarm management for process control: a best-practice guide for design, implementation, and use of industrial alarm systems*. Momentum Press.

[Saghir et al., 2018] Saghir, F., Gilabert, H., Boujonnier, M., et al. (2018). Edge analytics and future of upstream automation. In *SPE Asia Pacific Oil and Gas Conference and Exhibition*. Society of Petroleum Engineers.

[Sergeev and Balso, 2018] Sergeev, A. and Balso, M. D. (2018). Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*.

[Soares et al., 2016] Soares, V. B., Pinto, J. C., and de Souza Jr, M. B. (2016). Alarm management practices in natural gas processing plants. *Control Engineering Practice*, 55:185–196.

[Tuli et al., 2020] Tuli, S., Basumatary, N., Gill, S. S., Kahani, M., Arya, R. C., Wander, G. S., and Buyya, R. (2020). Healthfog: An ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated iot and fog computing environments. *Future Generation Computer Systems*, 104:187–200.

[Yoon et al., 2011] Yoon, S., Ye, W., Heidemann, J., Littlefield, B., and Shahabi, C. (2011). Swats: Wireless sensor networks for steamflood and waterflood pipeline monitoring. *IEEE Network*, 25(1):50–56.

[Yu et al., 2019] Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270.

[Zissis, 2017] Zissis, D. (2017). Intelligent security on the edge of the cloud. In *International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1066–1070. IEEE.