UNIVERSITY OF VAASA

THE SCHOOL OF TECHNOLOGY AND INNOVATIONS

AUTOMATION AND COMPUTER SCIENCE

Joel Reijonen

MASTER'S THESIS

Decentralized Machine Learning for Autonomous Ships in Distributed Cloud Environment

Master's thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, 1 November 2018.

Thesis supervisor	Prof. Mohammed Elmusrati
Thesis instructors	D.Sc. Miika Komu M.Sc. Miljenko Opsenica

PREFACE

“Decentralized Machine Learning for Autonomous Ships in Distributed Cloud Environment” has been an educational project where I had an opportunity to gain expertise especially in cloud computing, machine learning and autonomous ships from experts that are working in Nomadiclab, Ericsson Research Finland. During this project, my team and I filed a couple of invention disclosures regarding the topic of this thesis.

I would like to sincerely thank my supervisor Professor Mohammed Elmusrati, instructor D.Sc. (Tech.) Miika Komu and instructor M.Sc. (Tech.) Miljenko Opsenica for excellent guidance, support and feedback. In addition, I would also like to address my gratitude to Jan Melén, Jimmy Kjällman and Jani-Pekka Kainulainen for their supportive feedback and assistance.

Finally, I would like to thank my family and my girlfriend Nikolina for their altruistic and continuous support during on both this project and studies.

Jorvas, 1.11.2018

Joel Reijonen

TABLE OF CONTENTS

ABBREVATIONS

Agent	Machine learning agent
API	Application Programmable Interface
FIR	Finite Impulse Response
HTTP	Hypertext Transfer Protocol
IIR	Infinite Impulse Response
IPC	Inter-Process Communication
IT	Information Technology
ML	Machine Learning
NoSQL	Non Structured Query Language
OS	Operating System
RO	Recursive Optimization
RPI	Raspberry Pi
VPN	Virtual Private Network
XML	Extensive Markup Language
XPS	Extruded Polystyrene
YAML	YAML Ain’t Markup Language

UNIVERSITY OF VAASA
The School of Technology and Innovations
Author: Joel Reijonen
Topic of the Thesis: Decentralized Machine Learning for Autonomous Ships in Distributed Cloud Environment
Supervisor: Prof. Mohammed Elmusrati
Instructor: D.Sc Miika Komu
M.Sc Miljenko Opsenica
Degree: Master of Science in Technology
Major of Subject: Automation and computer science
Year of Entering the University: 2014
Year of Completing the Thesis: 2018 **Pages:** 95

ABSTRACT

Machine learning is a concept where a computing machine is capable to improve its own performance through experience or training. Machine learning has been adopted as an optimization solution in broad field of information technology (IT) industry. In addition, the availability of data has become more and more easier since the effective data storage and telecommunication technologies such as new generation cloud computing are developing. Cloud computing refers to a network-centric paradigm which provides additional computational resources and a scalable data storage. Even though the utilization of cloud computing enables improved performance of machine learning, cloud computing increases the overall complexity of the system as well.

In this thesis, we develop a machine learning agent which is an independent software application that is responsible for the implementation and integration of decentralized machine learning in a distributed cloud environment. Decentralization of machine learning enables parallel machine learning between multiple machine learning agents that are deployed in multiple clouds. In addition to the development of machine learning agent, we develop a data preparation module which ensures that the data is clean and complete.

We develop the machine learning agent and the data preparation module to support container implementation by taking advantage in Docker container platform. Containerization of the applications facilitates portability in multi-cloud deployments and enables efficient orchestration by utilizing Kubernetes. In this thesis, we do not utilize existing machine learning frameworks but rather we implement machine learning by applying known mathematical methods.

We have divided the development of the software applications in three phases: requirement specification, design and implementation. In requirement specification, we describe the essential features that are required to be included. Based on the requirements, we design the applications to fulfill expectations and respectively we utilize the design to guide the implementation. In the final chapter of this thesis, we evaluate functionality, ability to enhance performance and virtualized implementation of the applications.

KEYWORDS: Decentralized machine learning, distributed cloud computing, data preparation, containerization, orchestration

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Tekijä: Joel Reijonen
Diplomityön nimi: Hajautettu koneoppiminen autonomisille laivoille hajautetussa pilviympäristössä
Valvojan nimi: Prof. Mohammed Elmusrati
Ohjaajan nimi: TkT Miika Komu
DI Miljenko Opsenica
Tutkinto: Diplomi-insinööri
Oppiaine: Automaatio ja tietotekniikka
Opintojen aloitusvuosi: 2014
Diplomityön valmistumisvuosi: 2018 **Sivumäärä:** 95

TIIVISTELMÄ

Koneoppiminen tarkoittaa käsitettä, jossa tietokone kykenee parantamaan koneen suorituskykyä kokemusten tai opetuksen kautta. Koneoppimista hyödynnetään laajalti informaatioteknologian teollisuuden optimointiratkaisuissa. Tämän lisäksi datan saatavuudesta on tullut entistä helpompaa datan tallennus- ja tietoliikenneteknologioiden, kuten uuden sukupolven pilvilaskennan, kehittyessä. Pilvilaskenta viittaa tietoverkkoihin perustuvaan paradigmaan, joka tarjoaa sekä laskennallisia lisäresursseja, että skaalautuvaa datan tallennustilaa. Vaikka pilvilaskennan hyödyntäminen parantaa koneoppimisen suorituskykyä, se lisää myös järjestelmän yleistä kompleksisuutta.

Tässä diplomityössä kehitetään koneoppimista suorittava agentti, joka on itsenäinen ohjelmisto. Agentti vastaa hajautetun koneoppimisen toimeenpanemisesta ja integraatiosta hajautetussa pilviympäristössä. Hajautettu koneoppiminen mahdollistaa useiden agenttien rinnakkaisen koneoppimisen useissa pilviympäristöissä. Agentin lisäksi kehitämme datan valmistelumoduulin, joka takaa, että koneoppimisessa käytetty data on puhdasta ja eheää.

Agentti ja datan valmistelumoduuli kehitetään siten, että ne tukevat kontitettua käyttöönottoa hyödyntäen Docker-konttialustaa. Sovellusten käyttöönotto konteissa edistää niiden siirrettävyyttä yhdistetyissä pilviympäristöissä ja mahdollistaa tehokkaan orkestroinnin Kuberneteksen avulla. Tässä diplomityössä ei hyödynnetä valmiiksi luotuja koneoppimiseen käytettäviä viitekehyksiä, vaan toteutetaan koneoppimista soveltaen tunnettuja matemaattisia menetelmiä.

Diplomityössä sovellusten kehittäminen on jaettu kolmeen vaiheeseen: vaatimusmäärittely, suunnittelu ja toteutus. Vaatimusmäärittelyssä määritetään sovellusten välttämättömät ominaisuudet, jotka tulisi sisällyttää suunnittelussa ja toteutuksessa. Vaatimusmäärittelyjen pohjalta suunnitellaan sovellukset siten, että ne vastaavat vaatimuksia ja vastaavasti hyödynnetään suunnitelmaa toteutuksessa. Lopuksi arvioidaan sovellusten toiminnallinen, suorituskykyä parantava vaikutus ja virtualisoitu toteutus.

AVAINSANAT: Hajautettu koneoppiminen, hajautettu pilviympäristö, datan valmistelu, konttiteknologiat, orkestrointi

1 INTRODUCTION

The popularity of machine learning applications has increased over the past years in the field of information technology (IT) due to increasing amounts of available data (Smola & Bishwanathan 2008: 3). Machine learning is a concept where the computing machine is able to extract additional information from the data that is fed into the system. A machine utilizes the extracted information to learn and derive a reasonable result which can be used, e.g., in predictions, conclusions or decision-making operations. In most cases, the increased amount of data produces more precise results. For this reason, the data storage scalability and access together with cleaned data are fundamental requirements for machine learning.

Industrial trend is towards to having increasing number of devices that can be connected to the Internet[1]. Multiple connected devices increase the necessity of faster connectivity, larger data storage capability and higher amount of computational resources. To meet these expectations, the concept of cloud computing is one of the solutions that has gained reputation among the IT industry. Cloud computing enables network accessible and scalable deployment of data storage which grant additional computational resources (CPU, GPU, RAM, etc).

Cloud-based environments provide a potential response for resource demanding machine learning tasks. Clouds take advantage in virtualization of the mounted hardware which enables high scalability in the resources (Sosinsky 2011:3-4.). Scalable resources enable efficient management of the resources since a cloud does not necessarily has to reserve resources for operations that are not running continuously. Respectively, the cloud scales out the resources for the operations that have increase in demand. Furthermore, in order to locally scale resources, the cloud can be interconnected with other clouds and this way expand the overall amount of computational resources. An environment that consists of multiple connected clouds is known as distributed cloud environment.

[1] See for further information from Ericsson mobility visualizer: https://www.ericsson.com/en/mobility-report/mobility-visualizer?f=1&ft=1&r=2,3,4,5,6,7,8,9&t=8&s=1,2,3&u=1&y=2017,2023&c=1

In this thesis, we design and implement decentralized machine learning to optimize the performance of autonomous ships. Autonomous ships are self-acting ships that are composed of the usage of sensor fusion, control algorithms, communication and connectivity (Rolls-Royce 2016). We utilize sensor fusion to provide reliable data for machine learning which continuously strives to improve the efficiency of control algorithms. Moreover, we also take advantage in communications and connectivity when we decentralize machine learning between local and non-local clouds.

In this thesis, the distributed cloud environment consists of interconnected clouds in autonomous ships, harbors and data centers. Multiple connected clouds facilitate efficient computational load balancing for decentralized machine learning that, on the other hand, enables parallelism in learning.

1.1 Objective of the Thesis

In this thesis, we utilize parallel machine learning in order to harness computational resources from multiple clouds. Consequently, multi-cloud environment allows us to utilize even constrained resources for machine learning.

We implement and integrate data preparation module and decentralized machine learning agent in a distributed cloud environment. Data preparation module is a software application that guarantees the quality of the data that is used in machine learning. The quality of the data is a key factor when reliable learning results are desired. In this thesis, we utilize only the data that has been processed by the data preparation module.

Respectively, decentralized machine learning agent is responsible for operating machine learning related operations in various cloud-based environments. The agent is an independent software application which strives to improve overall performance of the system (Russel & Norvig 1995: 7). In the design and implementation of the agent, we consider "As a Service" -principles together with microservice architecture oriented development.

1.2 Structure of the Thesis

This thesis consists of seven chapters. Chapter 2 introduces relevant background theories and technologies that support the objective of this thesis. Chapter 3 defines the required features for data preparation module and machine learning agent, and presents an use case which sets certain requirements in design. Chapter 4 describes functional and deployment architecture of the components. In chapter 5, we describe the implementation of the components and use case specific testbed. Chapter 6 consists of an analysis and an evaluation that describes the performance of the implemented components. Finally, we discuss about the results and conclude this thesis in chapter 7.

2 FOUNDATIONS

In this chapter, we review the essential technology involving machine learning, distributed cloud computing and orchestration. First, we introduce the concepts of machine learning and how they can be utilized to foster the improved overall performance of the system. Secondly, we review the features of the cloud-based environment to enlighten the opportunities and challenges that they include. We deploy, manage and run applications as microservices in clouds where the environment specific advantages are utilized. Microservices, as an alternative software development concept, are reviewed together with orchestration as an approach that supports the development and management of cloud-based applications. Finally, we review the concepts of data preparation since clean and complete data support more precise machine learning.

2.1 Machine Learning

Machine learning is a concept which refers to the machine's ability to improve its own performance independently through experiences of the past or learning from examples (Brink *et al*. 2017: 3). Applications of machine learning are especially effective when the solution algorithm of the model is unknown or hard to determine, and when there are large amounts of data that needs to be processed. The goal of the machine learning is to optimize the parameters of the defined model by taking advantage of past experiences or examples. (Alpaydin 2010: 1–3)

Machine learning strives to extract hidden information from the data by utilizing mathematical methods such as theories in probability calculus and matrix algebra. However, proper extraction of the information does not always guarantee improved performance since the learned data may be incomplete, corrupted or it might include noise. Noise is an unwanted anomaly in the data which is mostly caused due to inaccuracies in measurements. (Alpaydin 2010: 30–31; Tan & Jiang 2013: 3)

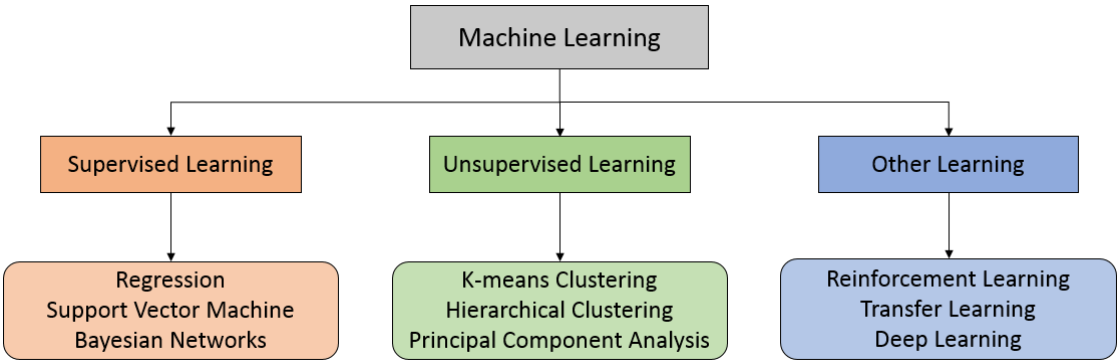


Figure 1. Machine learning techniques and some common methods.(Reconstructed from Hwang & Chen 2017: 33.)

Hwang & Chen (2017: 32) have defined the main machine learning techniques: supervised learning, unsupervised learning and other learning methods such as reinforcement learning, active learning and transfer learning (Figure 1). Machine learning techniques have different characteristics which should be considered in the design of the machine learning application. A certain technique may lead to the better results in optimization than using another learning technique.

2.1.1 Supervised Learning

Supervised learning is a technique where learning is based on training from examples which are provided by a supervisor. The supervisor is responsible for serving the system with a training set of labeled data which consists real observed input and output values. In supervised learning, learning utilizes the training set to learn generalized functionality of the system in such a way that the machine performs desired actions also in situations which have not been described in the training. (Sutton & Barto 2017: 2–3)

In supervised learning, the machine tries to fit a certain model which is based on the findings of the trained data. Alpaydin (2010: 9) has introduced classification and regression methods where the inputs are mapped to outputs by using supervised learning (Figure 2).

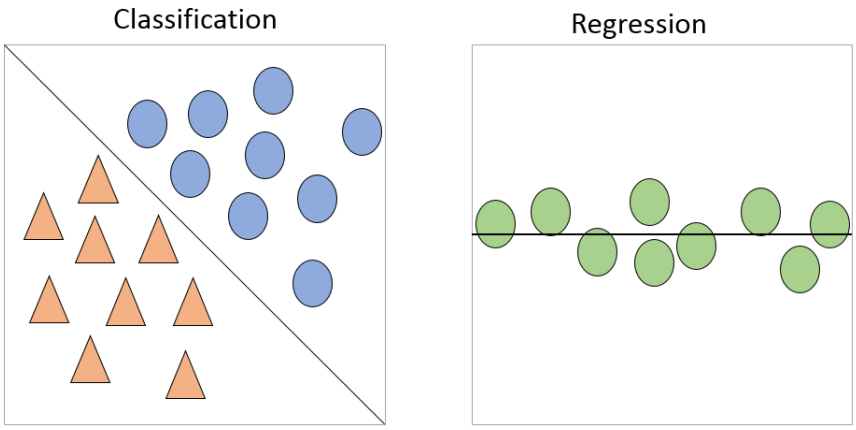


Figure 2. Classification categorizes inputs to certain classes whereas regression maps inputs to numerical output values.

Classification is a procedure that determines for which output or class do the sampling of inputs belong to. A function that maps the inputs to a certain class is called discriminant. In supervised learning the learned classification rule can be used to predict the classes of the inputs that have not been introduced in the training. Classification can be used in applications such as pattern recognition and natural language processing. (Alpaydin 2010: 5–8; Brink *et al*. 2017: 8)

Regression, on the other hand, is a procedure where the inputs of the system are mapped on outputs that are numeric values. In the supervised learning, the goal of regression is to train a model which maps inputs to outputs as precisely as possible. Regression model can be used to approximate the output values of certain inputs that are not represented in the training set. Regression is used in applications such as stock-market prediction, price estimation and risk management. (Alpaydin 2010: 10–11; Brink *et al*. 2017: 8)

The following formula defines how supervised learning can be used to solve classification and regression assignments (Alpaydin 2010: 9):

$$y = g(x|\theta) ,$$

where function g(·) represents the model, θ represents the parameters of the function and y represents a number in regression or class in classification. Supervised machine learning strives to optimize the parameters to fit the most satisfying model.

2.1.2 Unsupervised Learning

Unsupervised learning is a technique that does not utilize the observed output values and where the supervisor is not introduced (Alpaydin 2010: 11). Unsupervised techniques implement the learning operations by using the information of unlabeled data. A machine that utilizes unsupervised learning pursues to extract main features and structures of the input data and performs deductions from the findings (Brink *et al*. 2017: 26; Sutton & Barto 2017: 2).

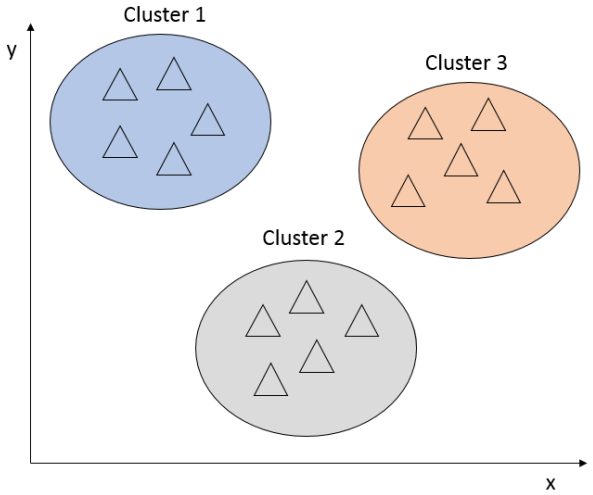


Figure 3. Clustering divides similar inputs into same clusters

Alpaydin has defined that the goal of unsupervised learning is to find repeating behavior of the input data where inputs can be divided into clusters or groups (Figure 3). Information of repeating behavior can be used to observe in which structures the similar occurrences of certain patterns occur more often and where not. The observation procedure is also known as density estimation. Concept of dividing inputs into clusters is also known as clustering which is one of the density estimation methods. *(*Alpaydin 2010: 11)

Clustering is a procedure where inputs with similar features and attributes are allocated in the same cluster. Clustering has no priori output values, so the construction of clusters is completely based on the information extracted from the input values. Clustering utilizes methods such as partitioning, density-based models and model-based methods. Applications that take advantage in clustering include, e.g., image analysis, data mining and bioinformatics. *(*Alpaydin 2010: 11–12; Bijuraj 2013: 169, 172)

2.1.3 Other Learning Methods

Other learning methods are techniques which may have similarities with supervised or unsupervised techniques but yet they have significant differences in their functionalities to be categorized differently. Other learning techniques include methods such as reinforcement learning, transfer learning and active learning. In this thesis, we introduce reinforcement learning and use it as an example how other learning techniques differ from supervised and unsupervised techniques. (Hwang & Chen 2017: 33–34; Sutton & Barto 2017: 2)

In reinforcement learning, the machine is rewarded if the actions or decisions that the machine has made have increased overall performance in a certain environment (Figure 4). The machine strives to learn which actions or decisions in a certain situation guarantee the maximized reward. Reinforcement learning relies on the machine to discover the set of desired actions by itself when the initial information about possibly rewarding actions is not provided. Although the machine does not have preliminary information, the machine must be served with responses related to the state of the environment, and the machine should have a determined objective relating to the state of the environment. The more actions affect positively on the objective of the machine, the better reward machine receives. (Sutton & Barto 2017: 1-2; Alpaydin 2010: 447–448)

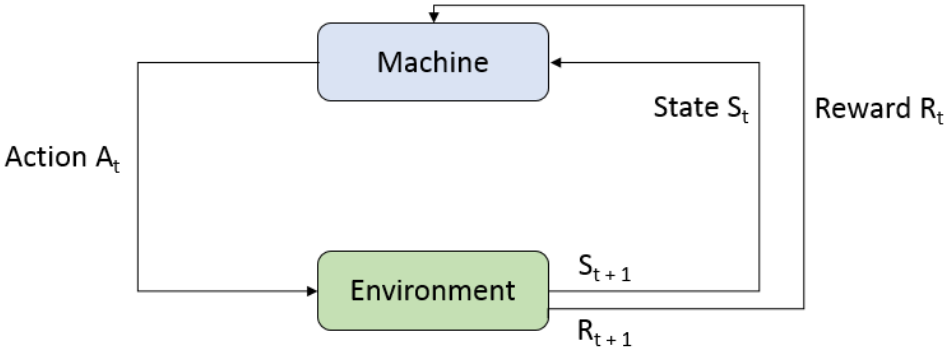


Figure 4. Machine receives rewards if taken actions improve performance of the machine in its environment. (Reconstructed from Sutton & Barto 2017: 38.)

Reinforcement learning differs from supervised learning since in reinforcement learning the training set of examples is not given. In supervised learning, a supervisor provides a training set of examples, but reinforcement learning does not employ any supervisor. Reinforcement learning is a learning procedure which strives to maximize the reward, and it does not provide information about the actions that should be taken but, instead, it provides information on how good the action was. (Sutton & Barto 2017: 1–2; Alpaydin 2010: 448)

Reinforcement learning does not belong to unsupervised learning methods since reinforcement learning pursues to maximize rewards instead of trying to find repeating behavior or the structure of the input data. Sutton & Barto (2017:2) have presented the idea of having more than two categories of learning techniques in the following way: “We therefore consider reinforcement learning to be a third machine learning paradigm, alongside supervised learning and unsupervised learning and perhaps other paradigms as well.”.

2.2 Distributed Cloud Computing

Nowadays the number of devices connected to the Internet has increased significantly which raises the requirements for connectivity, computing and data storage resources. One solution to tackle this problem is to utilize computation in a cloud.

Cloud computing refers to network accessible and scalable deployment of data storage which is capable of providing additional computational power and other resources. The benefits of cloud computing include lower software expenses due to reduced infrastructural maintenance, extensive access, shared environment and a standardized approach that supports integration of multiple platforms (Sosinsky 2011: 399). In this thesis, a platform that performs computation in multiple joint clouds is defined as distributed cloud environment.

2.2.1 Cloud Computing

The idea of centralized cloud computation and processing has raised its reputation during the past years due to its efficiency, scalability and accessibility. In this idea, the remote computation and processing of the information are handled in external data centers that supply network-centric computing and content management. Popularity of network-centric processing has led to the development of a paradigm called cloud computing where virtually shared resources are shared in a distributed network (Figure 5*)*. (Marinescu 2013: 1)

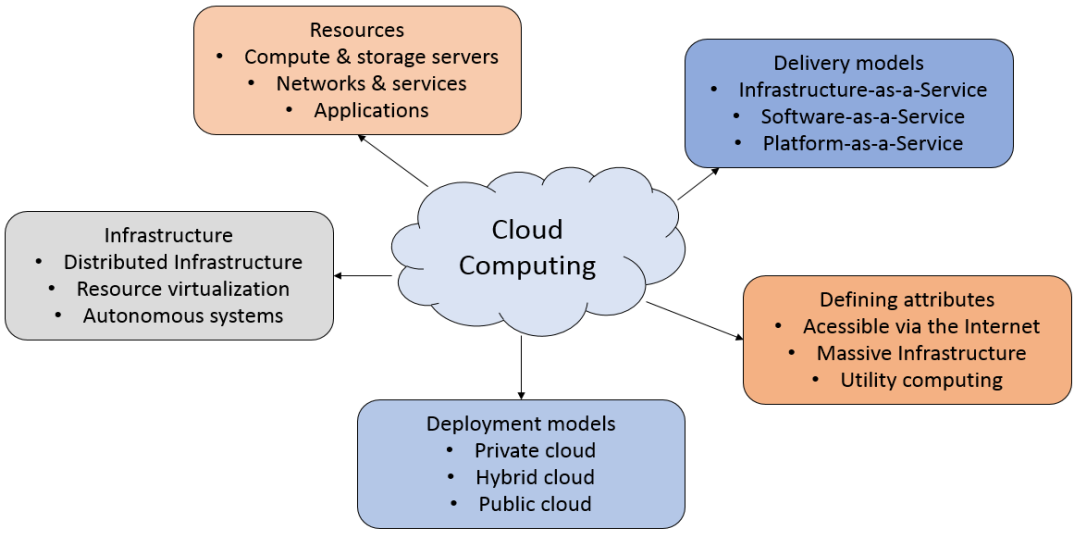


Figure 5. Concepts of cloud computing. (Reconstructed from Marinescu 2013: 2.)

"Cloud computing refers to applications and services that run on a distributed network using virtualized resources and accessed by common Internet protocols and networking standards." (Sosinsky 2011:3). Cloud computing provides remotely accessible and scalable resources and its popularity as a data storage solution has increased in the past years. Sosinsky (2011:4) has introduced how the word 'cloud' refers to two main concepts in cloud computing which are *abstraction* and v*irtualization.*

Abstraction in cloud computing means that the applications are running in unspecified physical systems, location of data storage is hidden from the user, and administration of systems is maintained by someone else. Virtualization on the other hand means that the cloud computing virtualizes mounted systems by pooling and shares resources in such a way that the resources are scalable. (Sosinsky 2011:3-4.)

2.2.2 Distributed Cloud Environment

Distributed cloud environment is a concept where multiple clouds are connected to each other. Clouds that form a distributed cloud environment can have differences in their features and computational resources. Different types of computational operations, such as sensor data collection and long term storing, take place in different clouds depending on

the architecture of the distributed cloud environment. Often the features change as the distance grows from the data source.

In this thesis the architecture of distributed cloud environment consists of three different clouds: edge, regional and central clouds (Figure 6).

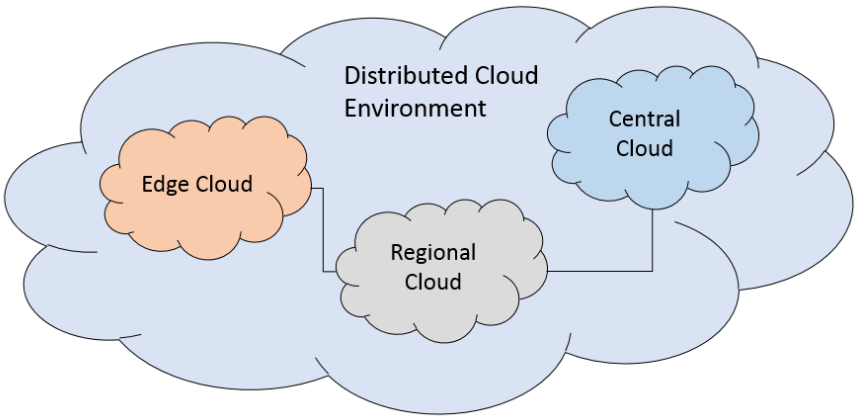


Figure 6. Distributed cloud environment consists of multiple connected clouds. (Reconstructed from Ericsson 2018).

Edge cloud is regarded as a cloud environment which is close to the end-user in this thesis. Edge cloud utilizes paradigm called edge computing which refers to the augmentation of computational capability at the edge of a network (Wang *et al*. 2017: 290). Edge computing reduces the network bandwidth usage and decreases the latency in the edge cloud. Edge cloud is a more constrained environment compared to the other cloud types especially when it comes to the overall computational resources and the reduced size of the data storage. In this thesis, the connectivity is also considered as a constrained resource in the edge cloud.

Regional cloud is a cloud environment that is bound to certain region. The concept of regional cloud is developed to guarantee that the cloud computational services are supported by the actors of certain area. Singh *et al*. (2014: 3) have defined the motivations behind the development of regional clouds with the following example: “An example is the proposal for a Europe-only cloud. Though there is often little detail surrounding the

rhetoric – indeed, the concept is fraught with questions and complexity – it generally represents an attempt at greater governance and control". In this thesis, regional clouds are part of distributed cloud environment where the management of the clouds is restricted to certain location.

In this thesis, the central cloud is located in a centralized data center which provides scalable computational resources on demand. Central cloud promotes remote accessibility which facilitates the computational load balancing in distributed cloud environment. Central cloud also acts as a long-term storage for the gathered data which supports analysis and monitoring of the devices and the cloud metrics.

2.3 Microservices

Cloud computing has gained a foothold in the IT industry, yet it has also declared novel challenges in software design. Cloud based systems are expected to improve overall reliability of usage and efficiency in performance and scalability but simultaneously they increase the complexity of the system. Increased overall complexity has led to a development of new design models such as microservice architectures and container technologies. (Hong & Bayley 2018: 152)

2.3.1 Introduction to Microservices

Traditionally software applications have been designed as monolithic applications which associates multiple software components into a single entity. Despite monolithic applications are quite common, they become more challenging to maintain and scale when the complexity of the system increases.

Components of monolithic applications rely strongly on each other which means that the components have to be managed, maintained and deployed as one aggregated entity. Due to their ponderous maintenance and deployment, the current industry trend is towards

microservices which are small and independent components that are responsible for handling their own operations (Figure 7). (Lukša 2018*:* 2–4; Rodger 2018: 46*)*

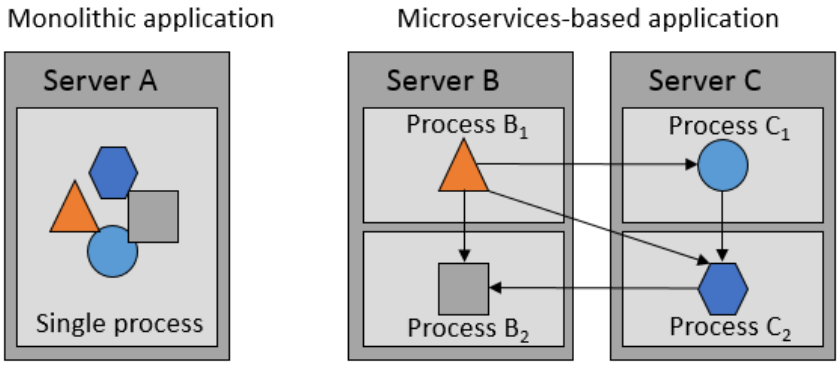


Figure 7. Similar operations running in monolithic application and in microservices-based application. (Reconstructed from Lukša 2018: 3.)

Microservices are intentionally developed as small and self-acting components which are relatively easy to maintain as standalone software. Rodger (2018: 35) has introduced the definition which states that the microservices should not have more than 100 lines of programmed code. Development of such small services are especially efficient when it comes to the debugging of the component or reconstruction of the code.

Hong & Bayley (2018: 154) have defined the benefits of preferring microservices over monolithic applications in cloud-based environment: continuous software evolution, seamless technology integration, optimal runtime performance, horizontal scalability and reliability through fault tolerance. These benefits foster the development, deployment and management of the system due to microservices' ability to receive individual updates and to scale resources individually.

Even though deployment of microservices has their pros there are also cons that need to be considered. Lukša (2018:5) has explained some challenges where the deployment-related decisions become more difficult as the amount of microservices increases. Lukša (2018:5) has also pointed out that the challenges are also harder to overcome if the amount

of deployment combinations increases which causes the increase in inter-dependencies of the components as well.

Parallel microservices share information between each other by utilizing technique called inter-process communication (IPC). Frequent communication of multiple microservices introduce another challenge where increasing overhead reduces overall performance of the system by increasing latency (Hammar 2014: 5, 35).

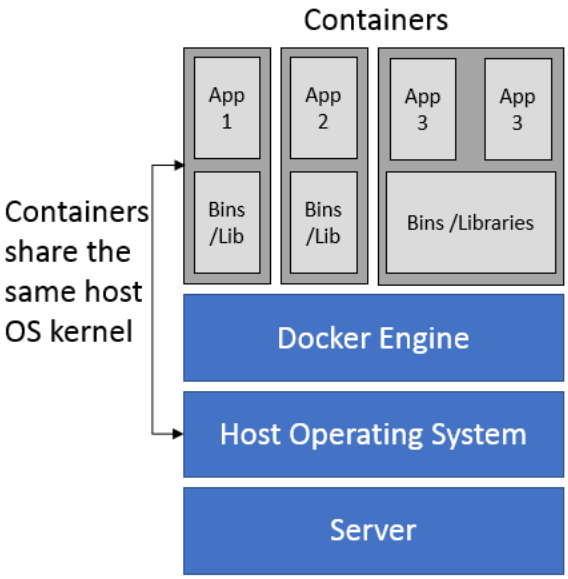
There are multiple options on how to overcome these challenges[2] but, in this thesis, the deployment of microservices is handled with Kubernetes and Linux containers. Kubernetes is an orchestration system that supports deployment and maintenance of containers. Kubernetes will be reviewed more in-depth in section 2.4.1.

2.3.2 Container Technologies

Container technologies can support microservices in such a way that the software components and required resources of a microservice are packed into a container image. Containers typically run a single application on top of the host operating system. The container runtime isolates the resources (memory, file system, network, etc) of the container from the rest of the system.

Figure 8 depicts how a single host can run one or multiple containers simultaneously where the containers share the same host operating system (OS) kernel (Hong & Bayley 2018: 154). Shared kernel increases the processing speed of the container instructions since they are in the same address space. In the other hand, kernel sharing decreases the level of security by amplifying the crucial kernel vulnerabilities and increases the latency.

[2] Challenges could be overcome also by utilizing unikernels and serverless architectures which are not reviewed in this thesis. See for further information about unikernels:
https://ieeexplore.ieee.org/document/7396164/
and information about serverless architectures:
https://ieeexplore.ieee.org/document/8360324/



s

Figure 8. Docker container virtualization. (Reconstructed from Juniper Networks 2018)

In this thesis, the development and deployment of the containerized applications are implemented using Docker container platform. Docker is a platform that supports development, deployment, packaging and execution of software applications in containers where application components are packed together with their execution runtime environments. Docker allows easy container portability for different hosts and Docker containers can be run on any device that is capable of supporting Docker. (Lukša 2018: 11–12)

Figure 9 depicts three essential concepts that illustrate how Docker platform supports development, deployment and running of containerized software applications. Essential concepts are: images, registries and containers. Lukša (2018: 13)

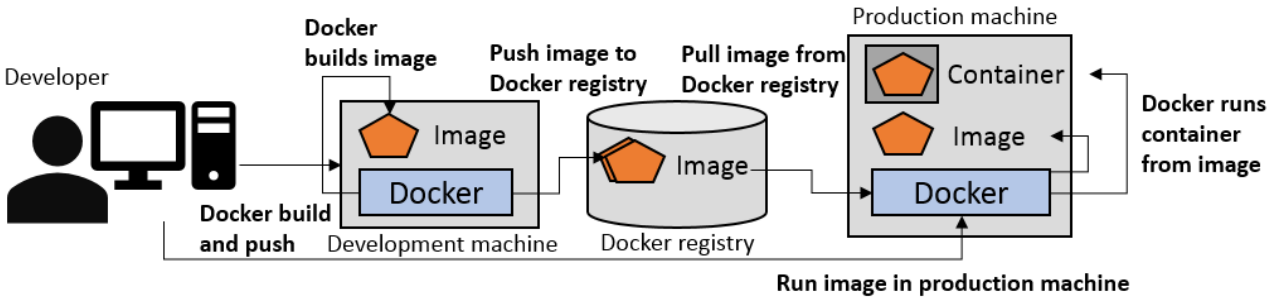


Figure 9. Development lifecycle of a Docker container. (Reconstructed from Lukša 2018: 13.)

Docker provides containerization tools which enable building of Docker images. Image is a layered entity that consists of components and environments for the software applications. Docker builds an image automatically by following the instructions that are described in the Dockerfile. Dockerfile composes command line commands that are needed for building an image in a text document (Docker Guides documentation 2018).

Developer can upload (push) and store successfully built Docker images into Docker registries which are responsible for storing images. Docker registry allows easy and shared access for machines where multiple hosts can download (pull) a desired image. The developer can also set the registries to be public or allow permissions for private machines depending on the confidentiality of the images. Lukša (2018: 13)

Docker container platform is the most popular of the container technologies, so it also enables the container creation from a Docker image. Docker containers are isolated processes that are running isolated from the host and other processes. A developer can restrict the resources of the Docker container in such a way that the container resource usage cannot exceed a certain level (Lukša 2018: 13).

2.4 Orchestration

Cloud-based systems consist of application components which are running on both virtualized and physical hardware that can be distributed in multiple locations (Sosinsky 2011: 46). The development, deployment and management of the application components in

cloud environment have been problematic and it rose the need for management standards and cloud orchestration (Kena *et al*. 2017: 18862). Orchestration of cloud-based applications and components strives to automate their deployment and management.

2.4.1 Kubernetes

Efficient deployment, configuration and management of increasing amount of deployed applications in cloud environment requires usage of orchestration. Kubernetes is an open-source system for automated deployment, scaling and management of containerized applications that are running in a cloud environment. Kubernetes is developed and introduced by Google in 2014. (Lukša 2018: 2, 16, 19)

In this thesis, we employ Kubernetes solely in the context of Docker container orchestration. Lukša (2018: 16) has pointed out that Kubernetes covers much more than Docker container orchestration but, on the other hand, containers are a convenient way of running applications in distributed cluster nodes. Container cluster in Kubernetes is a term which refers to composition of cluster master(s) and worker nodes. The structure of a container cluster is illustrated in the Figure 10.

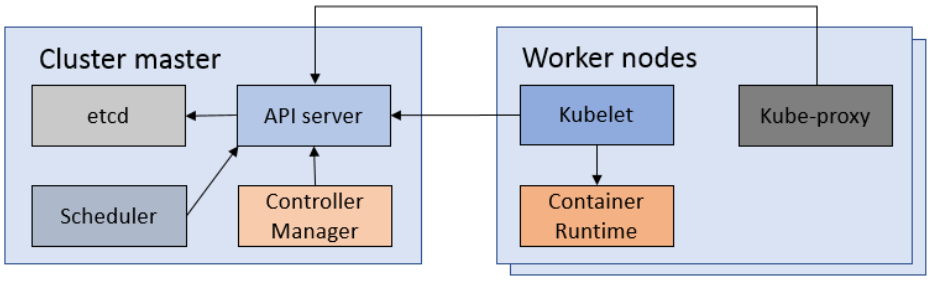


Figure 10. Kubernetes cluster consists of cluster master and worker nodes. (Reconstructed from Lukša 2018: 18.)

Cluster master (Control Plane) takes care of the functionality and control of the cluster. Cluster master consists of four types of components: Kubernetes application programming interface (API) Server, Controller Manager, Scheduler and etcd which are responsible for maintaining and controlling the state of the cluster (Figure 10). Applications however are not run by Cluster master components and that is where the role of worker nodes takes place. (Lukša 2018: 18–19)

The worker nodes take care of executing and running the applications in containers. A single node consists of three types of components: Container runtime, Kubelet and Kubernetes Service Proxy which are responsible for running, monitoring and serving the executed application (Figure 10). (Lukša 2018: 19)

2.4.2 Container Orchestration

Since, in this thesis, Kubernetes is used to orchestrate Docker-based containers, it is mandatory to wrap runnable applications into Docker images. Figure 11 shows that, in addition to initialization of container images, the images need to be pushed (uploaded) into an image registry where worker nodes can access and pull (download) the images that they require. Cluster master manages worker nodes by following the configurations of application description that consists of deployment-related instructions. (Lukša 2018: 19).

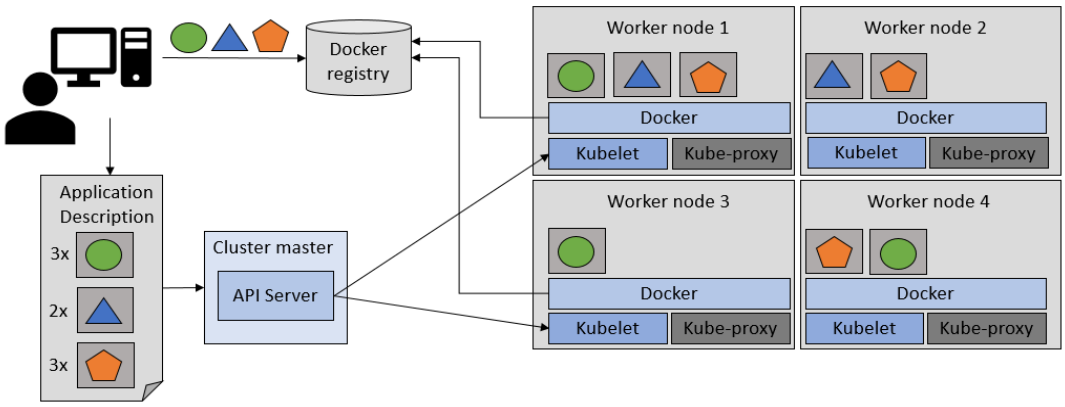


Figure 11. Kubernetes tells worker nodes to pull container images according to application description. (Reconstructed from Lukša 2018: 20.)

Application description provides instructions to the Kubernetes API server which is responsible for communication between nodes, user and other components of the Control Plane. The description provides information about the required container image or images that constitute the components of application and possible relationships to other nodes. In the description, it is possible to determine more specific instructions such as how many replicas of an instance should be running and whether the provided services are meant to be used by internal or external clients. (Lukša 2018: 18–20)

Kubernetes is an orchestration system that ensures the running of the applications as declared in the description. Kubernetes automatically takes care of the deployment and maintenance of the application and it can be seen e.g. if the application experiences an unexpected error, Kubernetes restarts it on the same or another worker node. (Lukša 2018: 20–21)

2.5 Data Preparation

Data preparation is essential requirement for machine learning because noisy, corrupted or incomplete data can lead to unwanted learning results. Data preparation in this thesis, mainly consists of noise removal, redundant information removal and imputation operations. In prepared data, the undesired anomalies are filtered from the data, the redundancy is reduced, and missing data points are derived.

2.5.1 Noise Removal

Appropriate noise removal should omit the noisy values from the data set in such a way that the anomalies are filtered out while preserving the original data pattern. Otherwise, data related operations would suffer from falsely derived bias if noisy values are present. (Tan & Jiang 2013: 3–4).

Multiple noise removal techniques exist such as finite impulse response (FIR), infinite impulse response (IIR) and rolling median filters which have certain advantages in different scenarios (Tan & Jiang 2013: 4). The design of the noise removal procedure should consider the characteristics of the collected data e.g. optimal filter for continuous data would not necessarily be the best choice for quantized data.

2.5.2 Redundancy Removal

Data redundancy refers to the data values that do not provide any additional information (Lucky 1968: 551). Redundant data increases the amount of computation that the machine learning operations have to perform without having any benefit in the learning.

Different methods exist for eliminating redundancy such as normalization, duplication removal and recursive optimization (RO) algorithms (Zhang *et al*. 2013: 106). These methods aim to reduce the size of the data without having negative impact on the results of machine learning.

2.5.3 Imputation and Excluding Methods

Collected data set may include missing data values due to failures in measurements. Missing data causes gaps between known data points and missing information might have harmful influence in machine learning. In this thesis, imputation and excluding methods handle the missing data values. Imputation replenishes the missing data whereas excluding methods disregard the missing data. (Alpaydin 2010: 89; Allison 2001: 5)

Different imputation techniques exist such as interpolation and regression which provide approximations of the missing data values (Alpaydin 2010: 90). Respectively, different excluding methods exist such as pairwise deletion and listwise deletion where the missing data is removed in a certain way (Allison 2001:5).

2.6 Summary

Machine learning refers to a concept where machines can learn to improve their performance based on the experience. Machine learning extracts hidden information from the data which can be used in optimization. Quality and quantity of the utilized data influences to the results of machine learning. High amount of prepared data leads to more satisfying learning results than less amount of unprepared data.

Cloud computing supports network accessible usage of scalable resources. Applications, in cloud environment, are running in unspecified physical systems and clouds pool resources by virtualizing. An environment that composes multiple connected clouds is called as distributed cloud environment.

Microservices are small and self-acting software components which can be deployed efficiently in a cloud-based environment. Microservices handle their own processes, and they share information between each other by utilizing inter-process communication. Container technologies can support containerization of microservices where the software components and their execution runtime environments are bundled into containers. In container development, Kubernetes facilitates deployment and management for increased number of containers. Kubernetes, as an orchestration system, automates deployment, scaling and management of containers.

3 PLATFORM REQUIREMENTS

In this chapter, we specify requirements for the machine learning based optimization which supports decentralized functionality. The requirements are based on a real use case in a distributed cloud environment that discloses opportunities and challenges for machine learning. Coupled with machine learning requirements, we introduce the requirements for data preparation module since the usage of raw data would be unfavorable in the learning operations. We describe the requirements on high level, and they are used to guide the technical design of the machine learning software.

3.1 Use case: Autonomous Ships

Our use case is an example of a real-world application where we employ and integrate decentralized machine learning in a distributed cloud environment. In this thesis, the use case involves deployment of autonomous ships that utilize edge computing in independent control and in decision-making procedure. Autonomous ships are miniature prototypes that demonstrate the full functionality of technical implementations which could be deployed on devices in production.

Autonomous ships act independently in such a way that human interactions are not needed in sailing. Ships cruise from one harbor to other by calculating an optimal route and avoiding possible obstacles in the water such as other ships or underwater rocks. Ships control the movement and the power usage autonomously by following instructions of control algorithms that utilize machine learning for optimization.

Ships perform autonomous operations in a cloud-based environment. The ships take advantage of on board edge computing where device related and computationally light weight operations are executed with low latency. We conduct more demanding operations in other clouds that have higher computational capacity. Deployment of autonomous ships introduces use case specific requirements for the design and implementation of the machine learning application.

3.1.1 Connectivity and Communication

The edge cloud of an autonomous ship has a constrained connectivity if the edge cloud is not connected to other clouds. The edge cloud, located within a feasible range of a harbor area, establishes a virtual private network connection (VPN) into the regional cloud of the harbor. Consistently, the edge cloud disconnects from a regional cloud when the ship sails outside of the harbor area.

The term VPN refers to a software that remotely connects a computer to private network across a public network. Thus, a VPN gives the illusion to the user of the computer as if it were directly connected to the private network. VPN consists of virtual connections which are provisional connections that have no physical instances. Connectivity in VPN is based on the packets that are routed over multiple machines on the public network. (Scott *et al*.1999: 2)

Autonomous ships should minimize data transmission between the local edge cloud and remote cloud(s) when the ship is sailing. In sailing, the ship relies solely on narrow band satellite communications. The ships should, instead, transfer the collected data in the harbors where the edge cloud is able to connect to the distributed cloud environment. Computationally heavy operations, such as machine learning, should be performed in a central cloud because the edge cloud needs to guarantee availability of the resources for the use of more essential procedures. However, the edge cloud may perform machine learning locally if the learning task is computationally light or if there are sufficient amount of available resources.

3.1.2 Sensor Fusion

Autonomous ships monitor their performance constantly with sensors and the ships collect monitored data for further processing. Reliability of machine learning and analysis of the performance of the ship is highly dependent on the amount of collected data. Performance of machine learning and analytic procedures improves generally when the

volume of collected data increases because a low amount of data may not introduce enough instances of possible events.

A control unit in the ship manages the autonomous sailing of the ship and additionally the control unit is responsible for data collection. Control unit collects data from the sensors of the ship. Sensors measure physical quantities of the ship such as velocity, acceleration and power consumption. In our use case, a ship includes multiple quantities that are measured and the measurements are performed three times per second.

The control unit stores measured data in a database which is located in the edge cloud. The control unit replicates the content of the database to a central cloud when the ship reaches a harbor area. Database replication, to the central cloud, clears space in the edge cloud.

3.1.3 Optimization of Engine Performance

We utilize the collected data to optimize the power usage of the ship's engine. The optimization aims to improve the performance of the engine by finding a model or an algorithm that describes the behavior of the data as precisely as possible. Efficient usage of the engine power minimizes ship's energy consumption and extends the potential sailing time.

The control unit of a ship has a state machine with different states which introduce state-specific objectives for optimization. In our case, we have two basic states, traveling and docking states, for which the ship alternates the objective of its performance. A very generic optimization is challenging to be define with a single algorithm even in our simple use case of two states, let alone in a more complex scenario involving a real ship. For these reasons, we take advantage of machine learning to optimize the overall performance of an autonomous ship.

3.2 Machine Learning Agent

In this thesis, a machine learning agent is a software application that is responsible for performing machine learning related tasks and interacting with other agents that are deployed in different environments. The agent composes of machine learning, evaluation, deduction and decentralization operations which should be designed in a generic way that supports possible deployment in multiple use cases. Thus, the agent should follow *as a service* principle where the application is available to be used and configurable by an end-user, but software updates and maintenance are managed by a developer. The primary objective of the agent is to conduct accurate inferences that rely on the findings from the collected data.

A machine learning agent can divide and distribute its workload to other agents in order to maximize performance. An agent needs to support interoperability in different environments since it should be able to operate in a decentralized way in multiple clouds.

3.2.1 Interoperability Requirements

In this thesis, the computational environment consists of multiple connected clouds, i.e., distributed cloud environment, where the computational resources can vary between the cloud types. Figure 12 illustrates how machine learning agents should adapt and adjust their performance in different cloud-based environments.

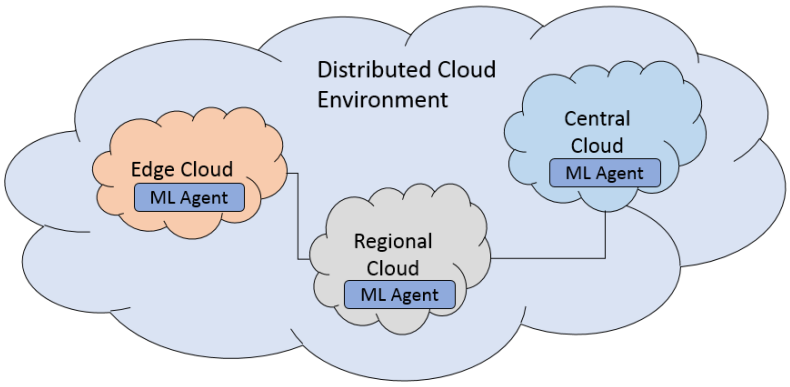


Figure 12. Machine learning agents operate in different cloud-based environments

Functionality and behaviour of the agent relies on the deployment environment since the agent should not reserve computational resources from other, more essential operations. For instance, when free available resources are very limited in a certain environment, the agent should not run computationally heavy machine learning operations simultaneously since it would critically reduce the performance of the system.

3.2.2 Orchestration Requirements

We require machine learning agents to be interoperable and act independently in the environment where they are deployed. Deployment and management of multiple independent agents become more laborious as the number of deployed agents increases which is a similar challenge when deploying multiple microservices. Proper management and deployment of agents requires orchestration.

Orchestration should be centralized to promote convenient deployment and life-cycle management of multiple agents. In other words, an orchestration system should manage the software configuration, operational optimization, provisioning, start up and termination of agents. Figure 13 depicts how the centralized orchestration system should manage multiple distributed cloud-based applications.

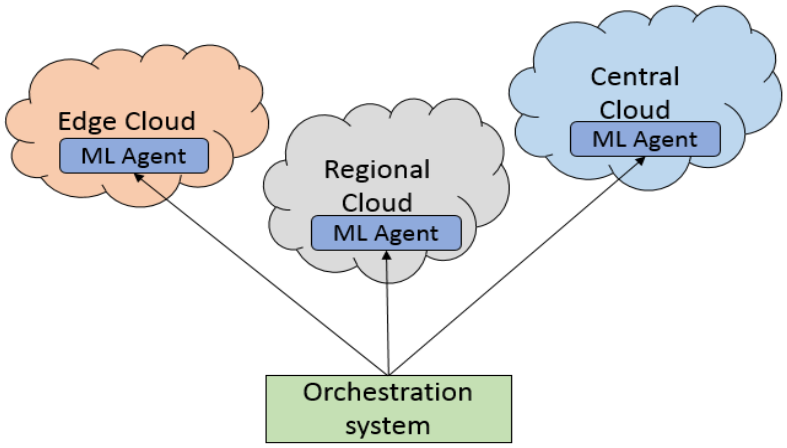


Figure 13. Orchestration facilitates deployment and maintenance of multiple machine learning agents

The orchestration system deploys and manages the operational state of the agents in an automated way. If an error occurs in a running agent, orchestration system recovers the situation, for instance, by re-launching the agent on another host.

3.2.3 Reusability Requirements

Machine learning agents need to support reusability in such a way that the changes or adjustments in operations are not required to be reconstructed in the source code. For example, if the context of machine learning changes over time, the agent automatically starts to perform additional learning and adjusts the learning-related parameters to fit the new circumstance. A developer should be able to inform the changes of desired functionality to the agent by using external configuration files.

External configuration files act as a customize template to the functionality of the software. A developer can serve the configuration files to the orchestration system which is responsible of ensuring that the states and instances of the software matches with the requirements that are defined in the configuration files.

3.2.4 Performance Requirements

A machine learning agent has to be able to extract hidden information from data in such a way that decisions are based on the extracted information and they improve the overall performance of the system. Hidden information consists of knowledge of the data which is not described in the initial data set. In addition, the hidden information can be knowledge that has not been described by predefined performance evaluation algorithms.

Occasionally developers may find performance evaluation algorithms difficult to define especially if the pattern of the data is complicated. The agent needs to conduct learning from the collected data even if the agent has no knowledge of predetermined algorithms or models. With this intention, the agent evaluates the fitness of the learning results in such a way that the evaluated results facilitate the decision-making processes and thus improve the overall performance.

As explained earlier, autonomous ships may have multiple states where the objective of the ship varies. The agent has to utilize learning for state-specific optimization which increases the coverage of learning. Further, state-specific learning enhances the effectiveness of state-specific optimization which correspondingly improves the overall performance of the autonomous ship.

The agent should support decentralized functionality where resource demanding machine learning operations can be distributed among multiple agents. Decentralization enables parallel processing which further accelerates learning and also enables learning in a constrained environment.

3.3 Data Preparation Module Requirements

In efficient machine learning, raw (unprocessed) data needs to be prepared before machine learning agents can utilize the data as an input because raw data may include noise, redundancy and missing values which may cause counterproductive effects. Proper data preparation is a preliminary requirement for machine learning related performance improvements.

Data preparation module needs to be interoperable in different cloud-based environments, so that the preparation can be executed in the any cloud that has available computational resources. For instance, if the data preparation is handled in the edge cloud, it would save data storage from other clouds and also save bandwidth by reducing the data volumes.

3.4 Summary

In this chapter, we described the requirements for the machine learning agent and the data preparation module that are necessary for performing decentralized machine learning in a distributed cloud environment. We presented an use-case scenario which introduced use case specific requirements for the machine learning agent.

The machine learning agent should be able to adapt to different cloud-based environments, to optimize the engine performance of an autonomous ship, to adjust its operations to manage multiple state-specific optimizations and to support decentralization. Data preparation module, on the other hand, improves the efficiency of machine learning agent by preparing the input data. The preparation consists of redundancy reduction, noise removal and handling of missing data.

4 DESIGN PROCEDURES

In this chapter, we describe the design of the architecture and the functionality of the machine learning agent and the data preparation module. We review the mathematical methods behind the designed functionality in detail since we later implement the components without utilizing existing machine learning frameworks.

4.1 Architectural Design of the Machine Learning Agent

Architectural design describes the high-level structure of the machine learning agent. The structure consists of software components and their relationships. Architectural design clarifies the functional and the deployment design which are explained in their separate sections in this thesis.

4.1.1 Functional Design

The functionality of the machine learning agent supports supervised machine learning. The utilized data is labelled where the inputs are mapped to the corresponding outputs. Labelled data is ideal for supervised learning since it enables the supervisor to construct a training set of examples.

Figure 14 depicts the functional architecture of the agent where the layers have are either visible or hidden. Visible layers consist of values that are exposed to the execution environment and hidden layer handles the computational processing of the agent.

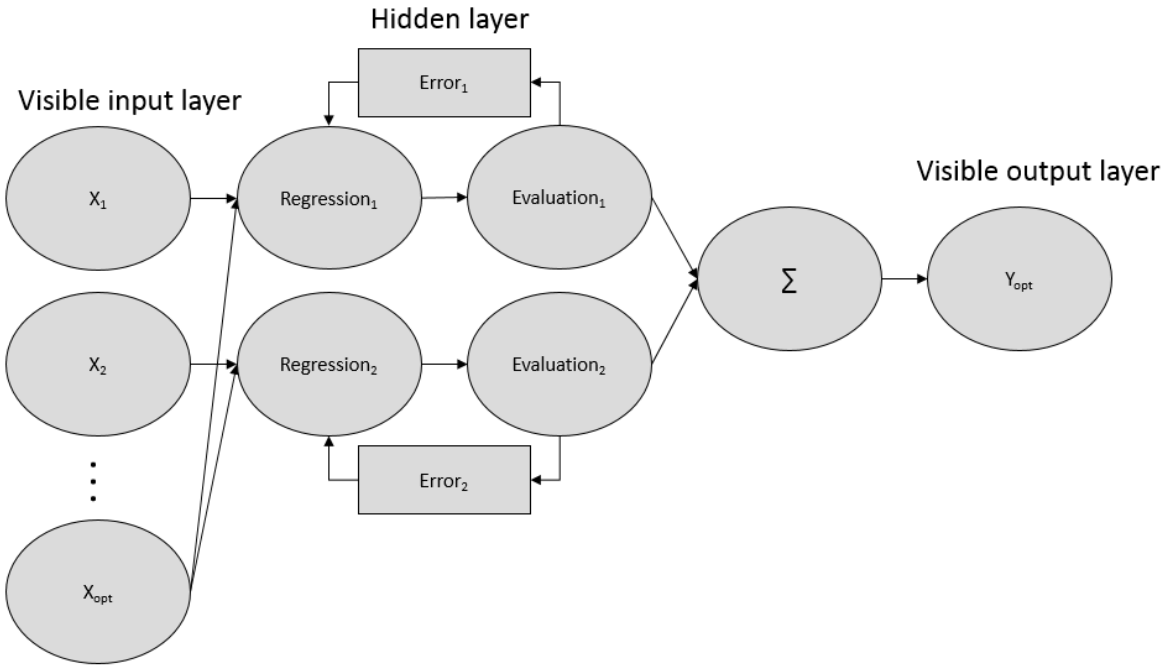


Figure 14. Overview of functional architecture of a machine learning agent

Input layer of the agent composes the real inputs of the system where one or multiple inputs are determined to be optimized. The agent receives the inputs from the data preparation module which provides non-redundant, noise-free and complete data.

Hidden layer of the agent utilizes regression based machine learning. Hidden layer includes the learning procedure of the regression model, evaluation of the learning results and logic for conducting deductions.

In model training, the agent relies on a supervisor to train the regression model of the system according to the given input values. After training, the agent evaluates the best fitting model and derives the parameters and the degree of the model. The best fitting model, in turn, improves the agent's ability to perform accurate deductions.

4.1.2 Deployment Design

We design the machine learning agents as relatively small and independent software components that utilize the Hypertext Transfer Protocol (HTTP) in communication. We consider microservice-oriented architecture in the design of the agent.

We deploy the agents in a cloud-based environment where the amount of available resources varies between the clouds and, thus, the agent needs to adapt to the situation. Microservice-based design of the agents supports easier integration, better runtime performance and more reliable fault tolerance than monolithic design.

Figure 15 depicts the deployment design of the agents with distributed cloud environment. In this thesis, the distributed cloud environment consists of three connected clouds: edge cloud, regional cloud and central cloud.

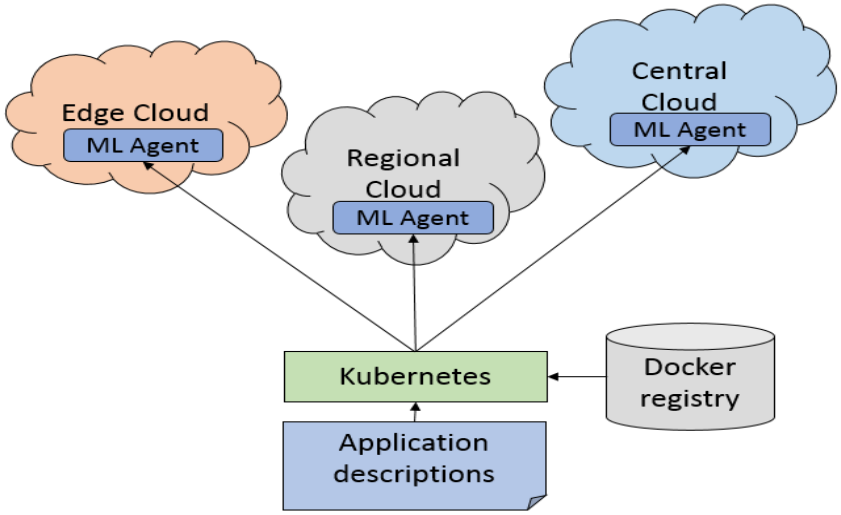


Figure 15. Overview of the deployment design of machine learning agents

In deployment design, we enhance the efficiency in portability of software by packing the software components and the required resources of the agent into Docker images. Docker platform supports the deployment of software applications as containers, and the containers can run on any machine that is running Docker software. Docker, in other words, enables convenient software portability for multiple working nodes. Moreover, we employ Kubernetes to orchestrate deployment, maintenance and running of containerized applications.

Kubernetes manages the containerized agents according to the deployment configurations. The configurations include information, e.g, about the image to be deployed,

number of desired replicas and the visibility of the services. Kubernetes guarantees that the containerized agents are running as they have been configured to run while recovering malfunctions by restarting terminated agent containers. Kubernetes also supports the network communication between the agents and other deployed applications.

4.2 Machine Learning Design

We design machine learning operations to fulfil our requirements. The distributed cloud environment and the use case introduce challenges and opportunities for machine learning. Requirements for machine learning emphasizes interoperability, reusability and improved overall performance.

4.2.1 Supervised Learning Design

A machine learning agent receives labeled data, as input, which is ideal to be used by supervised learning techniques. Supervised learning utilizes a supervisor which constructs a training set of examples. Training set is composed of collected data that consists of numeric values where the states of the machine are included. Since the collected data represents numeric values, we design the agent to takes advantage of polynomial regression (Figure 16).

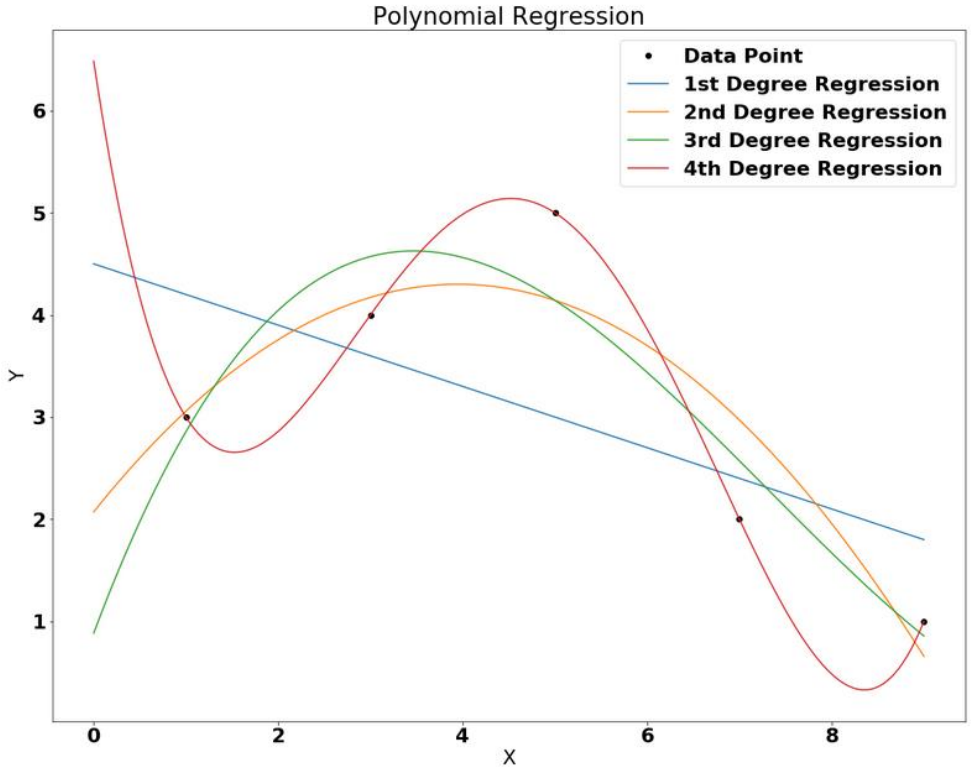


Figure 16. Example of regressions with four different degrees

The agent strives to learn the best fitting regression model that maps the inputs of the system to the outputs as accurately as possible. Regression model provides information about the pattern in relation between the values. The degree of the best fitting regression model depends on the relation pattern that varies between different inputs and outputs. The desirable result of learning is to find out the degree and the parameters of the best fitting regression model.

4.2.2 Conditional Learning Design

In this thesis, conditional learning refers to a concept where the machine learning operations depend on both their execution environments and states of the machine. With the environment, we mean here that the machine learning agent should adapt and reduce the complexity of the operations if the agent detects that the execution environment is restricted (Figure 17).

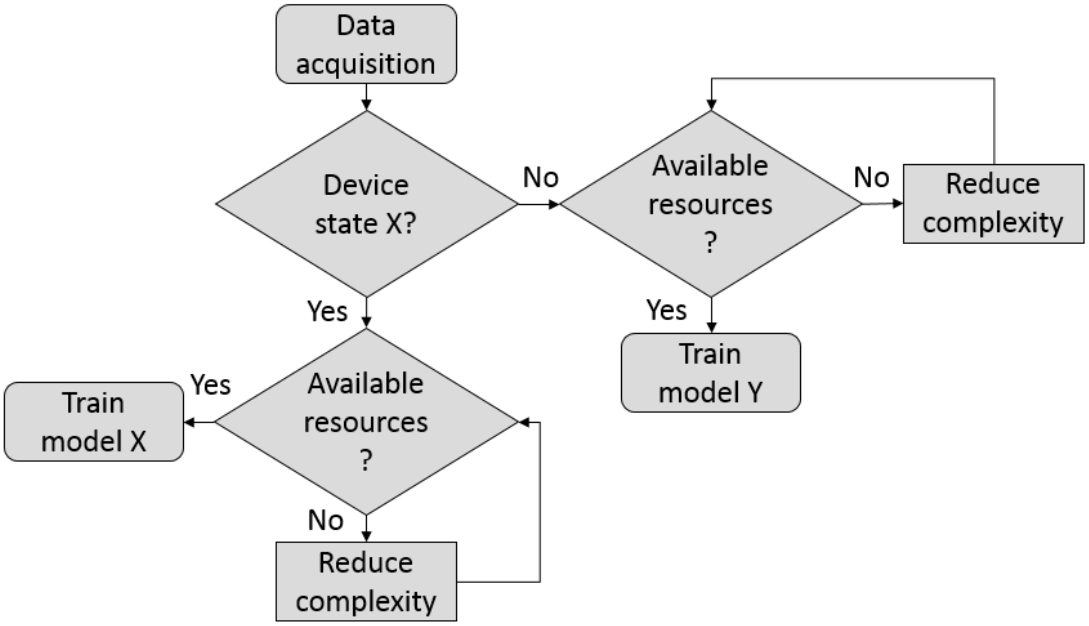


Figure 17. Flowchart of conditional learning where an agent can reduce the complexity of the machine learning according to the available resources

A device, that is optimized by machine learning, may have multiple states so we design conditional learning to support state-specific features. States may have different priorities where conditional learning aims to optimize the crucial features of a certain state. Conditional learning benefits the overall system performance as well because optimization of certain states may require reduced amount of computational resources, and therefore learning is possible to be conducted in a more constrained environment.

4.2.3 Decentralized Learning Design

Machine learning agent may detect that the available amount of resources are not sufficient enough to conduct machine learning even if the agent has reduced the complexity of learning. Consequently, we design the agents to be able to request assistance for learning from other agents that are on a feasible proximity. For instance, a machine learning agent in constrained edge cloud could request assistance on-demand from agents in regional and central clouds. Figure 18 depicts the design of the decentralized machine learning.

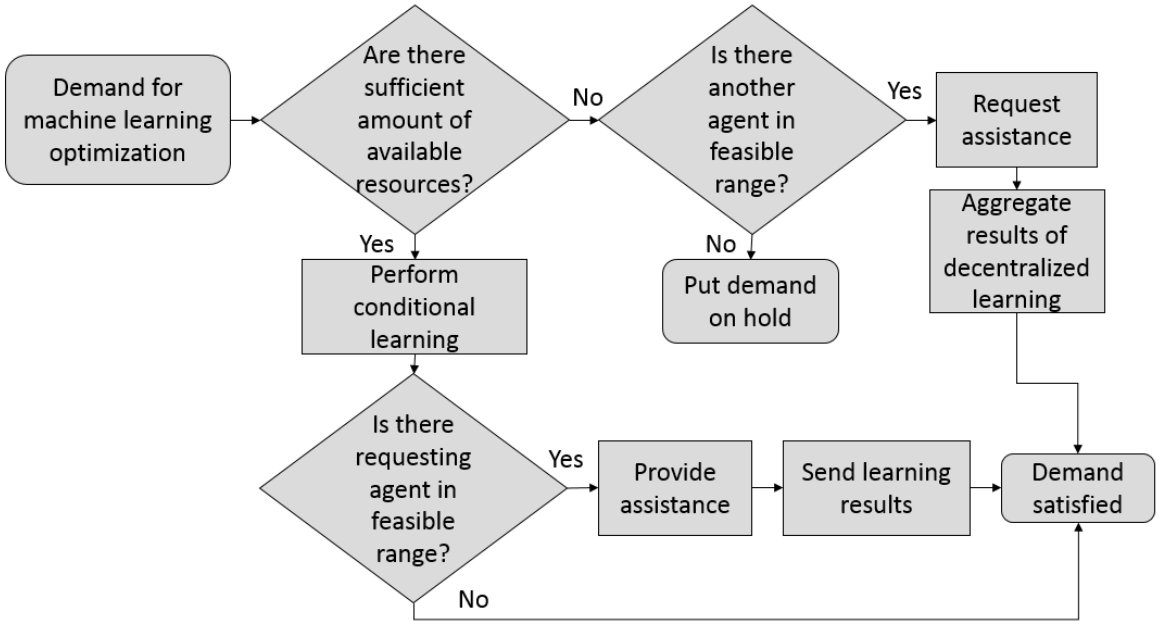


Figure 18. Flowchart of decentralized learning where an agent can request or provide assistance from other agents

The requesting machine learning agent sends a request for other agents where the requesting agent describes a machine learning objective that is desired to be processed decentralized. Respectively, the agents that can provide assistance inform the requesting agent that they can begin parallel and decentralized learning. After all the parts of learning are finished, the requesting machine learning agent aggregates the results of learning.

4.2.4 Fitness Evaluation Design

A machine learning operation may end up producing multiple feasible regression models so a machine learning agent needs to evaluate the feasible solutions and select the best fitting model. As our solution, we employ least squares estimation, where we compare the values of the regression model to the corresponding real values. The agent chooses the model that results the minimum sum of the deviations, i.e, the best fitting model and utilizes that model in deduction as explained in the next section.

4.2.5 Deduction Design

Deduction logic of the machine learning agent evaluates the best fitting model, which the agent learns from the training data, and pursues to find the most optimal solution that fulfills the initial learning objectives. Learning objectives refer to the prioritized objectives of the state-specific optimization.

As an use case related example, let's assume that the agent should find the optimal "load" of an autonomous ship in the sailing state. Here, optimal load means the optimal relation between the velocity and the energy consumption in such a way that the ship travels the furthest distance by consuming the minimum amount of energy as possible. The outcome of the deduction procedure would be parameters or information that improves the performance of the control algorithm of the ship.

In this thesis, we improve the deduction logic by conducting root analysis of a derived function where the function represents the best fitting regression model. Root analysis of the derived function provides information about maxima and minima values of the original function, and depending on whether the original function is monotonic or not.

4.3 Data Preparation Module Design

The data preparation module performs redundancy removal, missing data handling and noise removal of the collected data. The data preparation module guarantees that the utilization of the prepared data leads to significantly better results in machine learning.

We design data preparation module to support containerization where the designed components can be bundled into a container. Containerization facilitates running of the data preparation module in any cloud-based environment which improves the reusability of the module. In addition to reusability, containerization enables efficient deployment and maintenance of the data preparation module by utilizing Kubernetes.

4.3.1 Architecture

The data preparation module receives raw collected data as an input and produces cleaned (prepared) output data that enables more precise monitoring, analytics and machine learning. Figure 19 depicts the architecture of the data preparation module.

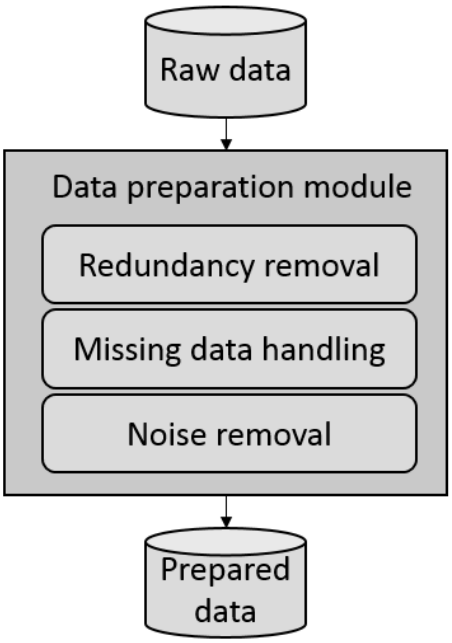


Figure 19. Overview of the architecture of the data preparation module

The functional parts are redundancy removal, missing data handling and noise removal units that are responsible for improving the quality of the output data. We regard the output of the data preparation module as prepared or clean data.

In order to reduce unnecessary computational processing in the data preparation module, the operations occur in the following sequence: 1) redundancy removal 2) missing data handling 3) noise removal.

4.3.2 Redundancy Removal Design

Collected data may include redundancy such as duplicates that should be discarded to avoid superfluous processing. Duplicates consist of multiple data samples that have identical content and the redundancy is handled by deleting all duplicates until only one exists. The elimination of redundant samples reduces the computation time in data-intensive operations such as machine learning, monitoring and analytics.

4.3.3 Missing Data Handling Design

Missing values in the data may produce false bias which affects to the accuracy and effectiveness of the data related operations. We design the data preparation module to handle missing values by utilizing listwise deletion.

Listwise deletion is an exclusion method which removes the data sample if it includes any missing values. There are two main advantages in listwise deletion: it can be used with any kind of data regardless of the data pattern and it does not introduce additional computational overhead. Respectively, listwise deletion has a remarkable disadvantage by removing significant fraction of the data if the occurrence of missing values is frequent. (Allison 2001: 6)

In the context of this thesis, the missing values in the collected data occur infrequently which means that the main disadvantage of listwise deletion is irrelevant. Avoiding extra processing is important in our scenario since it involves resource constrained environments. It is also worth noting that Allison (2001: 5) has compared different missing data handling methods and stated that many of the other methods are inferior to listwise deletion.

4.3.4 Noise Removal Design

We design the noise removal to remove anomalies from the collected data by utilizing moving average filter which calculates the average value of the data points within a predetermined interval. The interval for the filter is a configurable parameter and it represents

the order of the filter. A moving average filter smoothens the intervals of the data by filtering local unwanted anomalies. (Gonzalez & Catania 2018: 1542)

The collected data includes noise whose occurrence is stochastic. Our initial assumption is that the collected data is mostly unnoisy, so the moving average is a feasible solution since it is able to filter the local noise.

We define the order of the moving average filter in such a way that it removes the noise but also preserves the original data pattern. In the implementation phase, the proper order of the moving average needs to be tested and validated since the formal method for order predetermination is not defined in the theory in digital signal processing (Gonzalez & Catania 2018: 1542).

4.4 Selection of Mathematical Methods

In this section, we select and introduce mathematical methods to be utilized in machine learning. Selected mathematical methods realize the funcitonality, evaluation and deduction of the learning occurring in the agent. The machine learning agent does not utilize any existing machine learning frameworks, so mathematical methods need to be introduced and scrutinized properly.

4.4.1 Regression

The term regression refers to a mathematical procedure that estimates the relations between variables of interest by constructing a model. Construction of the model is based on the development of mathematical expressions that represent the pattern in the relation between dependent and independent variables. The constructed model includes parameters that are initially unknown, constant coefficients that determine the behavior of the model. In addition to the parameters, a model both has variety in its degree and its mathematical complexity which both are dependent on the modeled mathematical operation. (Rawlings *et al*. 1988:1-2)

Regression, in machine learning, has no predefined information about the degree and the parameters of the model in the context of this thesis. The main objective of machine learning agent is to learn the parameters and the degree of the best fitting model. We define the supported formulas of the regression models to be linear, quadratic and polynomial in their parameters for the use of the machine learning.

The simplest regression model is a linear model which represents change of a dependent variable at constant rate as the independent value increases or decreases. Rawlings *et al*. (1988: 2) have introduced following formula for the linear model:

$$Y_i = B_0 + B_1 X_i + \varepsilon_i ,$$

where Y represents the dependent variable, X represents the independent variable, coefficients B_0 and B_1 are the parameters of the model, ε represents the added random error and subscript i represents the observation unit where i = 1, 2, 3 ... n. In this formula, B_0 is the intercept and B_1 is the slope of the line

Linear regression model is accurate if the relation between dependent and independent variables has a constant rate of change. If the rate of change is not constant, the linear model would be inaccurate, and it is reasonable to fit a nonlinear model instead.

Quadratic and polynomial regression models are nonlinear which means that the degree of the model is second-order or higher. Rawlings *et al*. (1988: 236) have introduced the following formula for quadratic model which is the simplest extension of the straight-line model:

$$Y_i = B_0 + B_1 X_i + B_2 X_i^2 + \varepsilon_i ,$$

where second-order term, X^2 is involved in addition to X.

The formula for regression model of higher-order polynomials has been defined by Rawlings *et al*. (1988: 236) in following form:

$$Y_i = B_0 + B_1X_i + B_2X_i^2 + B_3X_i^3 + \ldots + B_pX_i^p + \varepsilon_i \,,$$

where the number of terms is directly proportional to the pth degree of the polynomial model.

Rawling *et al.*(1988: 2) have stated that a model usually falls into the class of models that are linear in the parameters in preliminary studies of the operation but the more realistic models are often nonlinear in the parameters. In this thesis, the optimal performance of the machine learning operations relies on regression models which are as realistic as possible. Linear models have reduced complexity compared to polynomial models, yet the machine should utilize the model with the most accurate fit.

4.4.2 Least Squares Estimation

Fitness of the regression model needs to be evaluated to find the best model. The least squares estimation is a method which we use to evaluate and select the model with the best fit. Potential models are evaluated by comparing sums of squared deviations between the real measurements and the estimations of the models. In least squares estimation, the best fit is the model that results the smallest sum of squared deviations. (Rawlings *et al*. 1988:3)

Machine learning operations validate and select the best fitting regression models by utilizing the principles of the least squares estimation. Rawlings *et al*. (1988:3) have presented the following formula for least squares estimation by calculating the sum of the squares of the residuals, $SS(Res)$:

$$SS(Res) = \sum_{i=1}^{n}(Y_{ri} - Y_{ei})^2 \,,$$

where the Y_{ri} is the real observation, the Y_{ei} is the estimation of the model and $Y_{ri} - Y_{ei}$ is the residual.

4.4.3 Root Analysis of the Derived Function

Root analysis of the derived function refers to the observation of a derived function at its roots. We utilize roots of the derived function to find the local minima or maxima of a function and to find out if the function is monotonic or not. Root or roots of the function are values that can be found when the function of x equals zero. (Press *et al*. 2007: 442) have defined root finding generically in the following way:

$$f(x) = 0 \, ,$$

where all terms are traditionally moved to left- or right-hand side, leaving zero to the other side.

Finding of the root is more challenging as the dimension of the equation grows. In one dimensional root finding the root is known to be found but in higher dimensions the existence of the root or roots are unknown until they are found. Possible number of roots that satisfies the equation simultaneously varies on the amount of independent variables which means that equation may have more than one solution. In addition to having multiple satisfying solutions, the equations may have no real roots at all. (Press *et al*. 2007: 443)

Root finding can be implemented by using different numerical methods such as Newton's method, bisection method or secant method. These methods can be divided into enclosure and fixed-point methods. Enclosure methods pursue to find the root by shrinking an interval that consists of at least one root, and fixed-point methods strive to approximate the root by taking advantage of the information about of the function. The efficiency of different root finding methods is evaluated by observing the required amount of computational iterations that produces a convergence with a satisfying accuracy.

Derivative describes the immediate rate of change of a function. The functional principle of the derivative is based on the comparison between the values of the function f with the values of x and $x + h$ where h is a small quantity. The comparison is used to define

the change, Δf by subtracting the value of $f\ (x)$ from the value $f\ (x + h)$. The rate of change, derivative at point x, is calculated by dividing the change by the interval Δx which is the change from x to $x + h$ (Anthony 2011: 41–42) :

$$\frac{\Delta f(x)}{\Delta x} = \frac{f(x+h) - f(x)}{(x+h) - x} = \frac{f(x+h) - f(x)}{h}$$

Strang (1991: 44) has introduced the formal definition of derivative as follows:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

where the f' is the derivative of function f, x is a variable and lim is the limit. Derivative of the function f can be defined only if the limit exists. Otherwise the derivative does not exist. (Strang 1991: 45; Anthony 2011: 42)

4.5 Summary

We have described the architectural and functional design of the machine learning agent and the data preparation module. We have designed the system to support interoperability, reusability and ability to enhance the system performance. Both the machine learning agent and data preparation module utilize cloud-based environments, where the amount of computational resources varies, so the software has been designed to adjust their operations according to their deployment environment.

Moreover, we designed the machine learning agent to be able to optimize the overall performance of the system by taking advantage of supervised, conditional and decentralized learning. Conditional learning enables the agent to reduce the complexity of learning task and to conduct state-specific learning, whereas decentralized learning enables parallel machine learning to be employed among multiple agents operating in different clouds.

Finally, we introduced the mathematical methods that we utilize in the implementation phase. The mathematical methods act as the “backbone” of machine learning in this thesis.

5 IMPLEMENTATION PROCESS

In this chapter, we implement the designed data preparation module and the machine learning agent in order to fulfill the functional and deployment requirements. We implement the data preparation module first since the machine learning agent is dependent on prepared data for reliable results. We develop it using Python programming language and two libraries called NumPy and pandas. The mentioned libraries support scientific computation, data analysis and the usage of different data structures (Numpy and Scipy Documentation 2018; pandas: powerful Python data analysis toolkit 2018). However, we do not use any existing machine learning frameworks in this implementation but rather develop machine learning and data preparation software based on the selected mathematical methods.

5.1 Implementation of the Data Preparation Module

The data preparation module is an independent software application that receives collected raw data as input and produces prepared data as an output. We implement the data preparation module to be configurable in such a way that the attributes to be prepared can be selected. Moreover, the data preparation module supports state-specific data in order to enable state specific machine learning processing. It is worth noting that reconfigurability can spare computation resources because processing of unnecessary attributes can be omitted.

Data preparation module takes advantage of so called data frames in processing. A data frame is a data structure where the data is stored as list of vectors of equal length. Then, prepared data is stored into a database where the data is accessible for other operations such as machine learning and analytics.

5.1.1 Duplicate Removal

We implement redundancy reduction by removing duplicates from the given data set. Duplicates do not provide any additional information for data reliant tasks such as in machine learning but instead they just increase the computational load. In this thesis, we regard duplicate removal to be an effective operation to be implemented since our data includes unique time stamps of the measurements. If we would not include unique key values such as time stamps, the possibility of having multiple duplicate measurements, which may or may not be redundant, exists.

The data preparation module removes duplicates from the data frames by detecting and deleting the redundant rows. However, duplicate removal preserves one instance to represent the identical set of measurements. After the duplicates are removed, the data frame is re-indexed so that the duplicate removal does not violate the structure of the data frame. Re-indexing refers to a procedure where the indexes of a constructed list are ordered.

5.1.2 Listwise Deletion

We implement missing data handling, in the data preparation module, by utilizing listwise deletion. Listwise deletion is an exclusion method where the observation is disregarded if it includes missing data. The volume of missing value occurrences is relatively low in the collected data so the negative impact of listwise deletion is negligible.

Listwise deletion utilizes data frames by detecting missing values and removes the rows that contain missing values. After listwise deletion, the data frame is re-indexed again so that the listwise deletion does not violate the structure of the data frame.

5.1.3 Moving Average Filter

We implement noise removal by developing a filter which calculates the moving average of the data. Moving average computes the average of data points within a predefined interval. In fact, we specify the interval for this filter to have a length of 12 data points.

An interval of this length has relatively accurate efficiency in data pattern preservation and in the removal of local noise within the scope of our use case.

Moving average filters the data values that have been defined in the configuration. Configuration consists of the quantities to be filtered and the states in the case state-specific filtering is required. The moving average computes the result from non-redundant and complete data frames by utilizing *rolling* function from the pandas library (Figure 20). After the data is filtered, it is stored into a database where it can be accessed by machine learning operations.

```
for state in states:
  # State-specific filtering
  state_df = complete_df.loc[complete_df['state'] == state]
  # Filter attributes of interest
    for interest in attributes:
      df.append(state_df.loc[:,interest].rolling(12).mean())
```

Figure 20. Noise removal by utilizing rolling mean

5.2 Implementation of Machine Learning

We compose machine learning of regression, conditional and decentralized learning. First, regression learning is an ideal technique to be utilized since the collected data is labeled and the data consists of numerical values. Second, conditionality enables a machine learning agent to perform state-specific learning and reduce the complexity of on-demand learning. Third, decentralization facilitates parallel machine learning and, thus, supports learning even in constrained environments.

We describe the implementation of machine learning and deduction logic in two sections. In this section, we implement only describe related functionality whereas in the next section we implement deduction logic that relies on the results of learning.

5.2.1 Supervised Learning: Regression

Collected data consists of location and measured sensor data from an autonomous ship. Sensors measure and gather values of physical quantities and enable the data to be labeled. Labeled data supports the deployment of supervised learning where a supervisor can establish a training set of examples. We implement machine learning by utilizing regression which is one of the supervised learning techniques.

We implement regression by utilizing polynomial regression where the result of the learning is the degree and the parameters of the model. The learning operation relies on a set of training data which is provided by a supervisor. The supervisor constructs the training data set from the real measurements from the sensors and labels the measurements as inputs. The supervisor provides the training data as a data frame and informs the machine learning agent about the modeled variables of interest.

We implement regression based learning with a function that trains regression models with different degrees of polynomials. The function receives a training set that composes variables of interest as parameters and fits the models with configurable amount of degrees. We achieve polynomial regression by utilizing *polyfit* function from the NumPy library (Figure 21).

```
def TrainModels (x, y, degree):
  # Loop through degrees
  for z in range (1, degree):
    # Fit models
    model = np.polyfit(x, y, z)
```

Figure 21. Simplified regression model training function

The default configuration consists of degrees from one to seven which restrict higher degree models that are computationally more resource demanding. Generally, the accuracy of the regression increases as the degree increases but the computational complexity increases as well.

5.2.2 Conditional Learning

We implement machine learning in such a way that learning supports optimization of the state-specific objectives of an autonomous ship. In addition to state-specific optimization, machine learning agent is implemented as cloud-native service. Distributed clouds have variety in their capacity of resources which may restrict computationally heavy machine learning. For these reasons, machine learning is conditional.

Conditional machine learning operates differently depending on the state of the autonomous ship. We implement conditional machine learning by determining machine learning objectives related to the state of the ship. In our use case, conditional learning consist of two states: sailing and docking.

Conditional machine learning adjusts the operational complexity of the learning algorithm based on the deployment environment. The more constrained the environment is, the more simple the learning procedure will be. The agent receives conditional learning related information about the environment from Kubernetes when it deploys the agent. The deployment, in turn, can take place in a distributed cloud environment which consists of: edge, regional and central clouds. In order words, the agent may conditionally adjust the complexity of the regression algorithm as depicted in Figure 22.

```
# Reduce complexity of local learning
if(environment == "edge"):
    TrainModels (x, y, 3)

elif(environment == "regional"):
    TrainModels (x, y, 5)

elif(environment == "central"):
    TrainModels (x, y, 8)
```

Figure 22. Machine learning agent adjusts the complexity of local learning based on the deployment environment

In this thesis, the conditionality for machine learning supports as a service principles where an end-user can configure state-specific objectives for machine learning and set in

the environment related boundaries. The end-user can also configure the numerical boundaries of machine learning in such a way that the weight of learning is restricted in the decision logic.

5.2.3 Decentralized Learning

We implement machine learning to support decentralization which enables parallel computation. Parallelism accelerates the overall learning speed and it enables learning in an environment where the available amount of resources are constrained. Thus, we implement decentralization in such a way that machine learning can be distributed among multiple machine learning agents for different cloud environments.

Decentralized learning enables machine learning agents to request or provide assistance for learning. A machine learning agent that does not have sufficient amount of available resources can request assistance from other agents that are on a feasible range. Requesting machine learning agent sends a request that informs other agents about the objective of learning via HTTP. After assisting machine learning agents have finished computation and sent their results, the requesting agent aggregates the results of decentralized learning. The following program snippet depicts the functionality of a requesting agent. In the Figure 23, the requested parameters for the learning algorithm are the optimum of the best fitting model in a certain interval and a list of the sums of squared deviations (errors).

```
# IP addresses refer to the agents in feasible range
request= 'http://192.168.2.131:8080/requestAssistance'
results = 'http://192.168.2.130:8080/fetchErrors'
polling = True
# Request for decentralized learning
r = requests.put(request, data={'request': True, 'x': x, 'y': y,
                  'degree': requestDegrees})
# Poll for available agents
while(polling):
      r = requests.get(results).content
      # Fetch results of decentralized machine learning
      if(r['done'] == True):
            decentralOpt = r['optimum']
            decentralErrors = r['errorList']
            polling = False

# Aggregate the results
if(min(decentralErrors) < min(localErrors)):
      aggregatedOpt = decentralOpt
else:
      aggregatedOpt = localOpt
```

Figure 23. Machine learning agent can request assistance for decentralized learning

A requesting machine learning agent can poll other assisting agents in a feasible range to distribute processing for them. An assisting agent receives information about the objective of learning from the requesting agent and then the assisting agent begins to perform learning based on the received information. After the learning task is finished, the assisting agent sends the desired parameters to the requesting agent via HTTP. Figure 24 depicts the functionality of the assisting agent.

```
# The IP addresses refer to the agents in feasible range
assist = 'http://192.168.2.131:8080/requestAssistance'
results = 'http://192.168.2.130:8080/results'
polling = True
# Poll for available agents
while(polling):
  r = requests.get(assist).content
  # Train models that are requested
  if(r['request'] == True):
    parameters = TrainModel(r['x'],r['y'],r['degree'])
    # Send the results of learning
    r = requests.put(results, data={'decentralOpt':
          parameters[0], 'decentralErrors': parameters[1]})
    polling = False
```

Figure 24. In our implementation, the role of agent (requesting vs assisting) is configurable

5.3 Implementation of Deduction Logic

Without the ability to perform justifiable deductions, a machine learning agent could end up producing inconsistent results. In this section, we describe the implemented deduction logic that enables the agent to conduct statistical inferences. Deduction logic relies on the findings of the learning procedure and they aim to improve machine learning based optimization. The agent utilizes fitness evaluation, model selection, global optimum and efficiency aggregation in deduction.

5.3.1 Fitness Evaluation and Model Selection

The supervisor, in a machine learning agent, trains regression models with different degrees in order to find the models that produce a satisfying result. Despite multiple feasible solutions would be available, the goal of machine learning is to find and select the most accurate model with the best fit. We implement continuous fitness evaluation during the learning phase which enables the machine learning agent to select the best fitting model during early stages.

In this thesis, we evaluate potential regression models by utilizing least squares estimation where we calculate the sums of the squared deviations of the models. The models that results the smallest sums of squared deviations are regarded as the best fitting models, and the agent utilizes these models to support deductions in global optimization (Figure 25).

```
def TrainModels (x, y, degree):
  # Train models
  for z in range (1, degree):
    model = np.polyfit(x, y, z)
    # Select the measured values
    for i in set(x):
      subset = (x == I)
      # Loop through values
      for j in range(0, len(subset)):
        if(subset[j] == True):
          values.append(y[k])
          # Least squares estimation
          error = error+(np.mean(values)-model(subset[k]))**2
      # Append errors
      errorlist.append(error)
  # Select the model with the smallest sum of squared deviations
  bestFittingModel = 1 + errorlists.index(min(errorlist))
```

Figure 25. Extended regression model training function that enables evaluation of learning by taking advantage of least squares estimation

If the optimization objective is multidimensional, it would be necessary to aggregate the information of the selected models that describe the objective. As an example, optimization of engine power usage could be regarded as a multidimensional task where the traveled distance, with a certain load, should be maximized and the energy consumption should be minimized. In this thesis, we implement multidimensional optimization by taking advantage in efficiency aggregation and selection as explained in the next section.

5.3.2 Global Optimum and Efficiency Aggregation

We derive global optimum by conducting a root analysis of the derivative of the model function. The root analysis describes the machine learning agent whether the model

function is monotonic or not, and what are the local minima or maxima. In addition to the root analysis, the machine learning agent discovers if the feasible function values are in a closed or an open interval. Discovery of the intervals is crucial because these boundaries may also be the global optimum solution. The machine learning agent obtains the global optimum from the results of root analysis and by testing boundaries (Figure 26).

```
def RootAnalysis(model, interval):
  # Find roots of the derivative
  modelDerivative = model.deriv(1)
  roots = modelDerivative.roots
  # Find global maxima
  if(max(interval) > max(roots):
    globalMaxima = max(interval)
  else:
    globalMaxima = max(roots)
  # Find global minima
  if(min(interval) < min(roots):
    globalMinima = min(interval)
  else:
    globalMinima = min(roots)
```

Figure 26. Root analysis function that derives global minima and maxima

In this thesis, we utilize functions from the NumPy library to calculate the derivative of the model function and the roots of the derivative. The root finding algorithm, in NumPy, relies on computing the eigenvalues of the companion matrix, and the algorithm is one of the fixed point methods (Numpy v1.13 Manual 2018).

Efficiency aggregation refers to the utilization of use of the selected regression model. We utilize the efficiency aggregation in multi-objective optimization to find a solution that satisfies various learning objectives simultaneously. We aggregate efficiencies by calculating the ratio between multiple model functions of global optimum (Figure 27).

```
def GlobalOptimum(x1, x2, y, interval):
  # Train models
  model1 = TrainModels (df.loc[:, y], df.loc[:, x1])
  model2 = TrainModels (df.loc[:, y], df.loc[:, x2])
  # Find global optima
  globalopt1 = RootAnalysis(model1, interval)
  globalopt2= RootAnalysis(model2, interval)
  # Calculate efficencies
  efficiency1 = model1(globalopt1) / model2(globalopt1)
  efficiency2 = model1(globalopt2) / model2(globalopt2)
  # Select the best efficiency
  if(efficiency1> efficiency2):
    print('Choose value: ', globalopt1)
  else:
    print(' Choose value: ', globalopt2)
```

Figure 27. Deduction logic for multi-objective optimization that calculates and aggregates efficiencies between trained models

The machine learning agent selects the global optimum that provides the highest aggregated efficiency to foster the efficiency of multi-objective optimization. This way, decisions of the machine learning agent strive to optimize the performance of the system by extracting the hidden information from the collected data.

5.4 Virtual Implementation

In this section, we containerize the implemented machine learning agent and data preparation module by utilizing Docker container platform. Containerized versions of the implemented applications are a convenient way of running applications in a distributed cloud environment since the containers package the applications together with their runtime environments. After containerization, we deploy the containerized applications in a cloud-based environment by employing Kubernetes.

5.4.1 Container Implementation

We bundle the data preparation module and the machine learning agent into Docker containers. The containers include software components, required libraries and execution runtime environments. Containerization facilitates the portability of applications in Docker-capable environments.

Docker daemon builds containers from the configured image. Respectively, docker builds images automatically by following the instructions from a Dockerfile which describe the additional software libraries and other dependencies to be installed on top of the base image (Figure 28).

```
# Dockerfile of the machine learning agent / preparation module
FROM python:2.7-slim
WORKDIR /base
ADD . /base
RUN pip install --trusted-host pypi.python.org -r require-
ments.txt

CMD ["python", "Main.py"]
```

Figure 28. Simplified Dockerfile that defines the working directory, required libraries and the dependencies of an image

Moreover, we build and store the resulting images into a Docker registry from where the Docker daemon can pull a desired image. Figure 29 illustrates how an image of a machine learning agent is built from previously shown Dockerfile.

```
Docker build -t edgeagent:v1 .
```

Figure 29. Docker build command builds an image that is based on the instructions from a Dockerfile

Docker has its own container orchestration system (called Docker Swarm) but instead we utilize Kubernetes from Google to deploy containers in this thesis.

5.4.2 Deployment with Kubernetes

Management of containerized applications can be laborious without using a proper orchestration system, especially if the usage of the application suddenly peaks out and then service is expected to scale out rapidly. For this reason, we employ Kubernetes to facilitate the orchestration of containerized applications in a cloud-based environment.

We can manually run images, as containers, on a Kubernetes cluster by utilizing kubectl run command. Kubectl run command builds a container based on a certain image and creates a so called "Deployment" to manage the constructed container. The created Deployment, on the Kubernetes cluster, ensures that the desired functionality of the container and also recovers the container from malfunctions based on the defined policies. With kubectl run command, we can define multiple parameters which include, e.g., networking, restart and image pull policies. Figure 30 shows how we can implement a deployment consisting of two containerized machine learning agents from the command line.

```
kubectl run edgeagent ––image=edgeagent:v1 --port=80
                      --replicas=2 ––restart= Always
                      –image-pull-policy = IfNot-Present
```

Figure 30. Kubectl run command builds a container in the Kubernetes cluster

In the previous example, the command becomes more the manually typed command becomes complex and more difficult to remember if we employ more specific parameters. To facilitate easier deployment, we utilize YAML (YAML Ain't Markup Language) files. YAML is a data serialization standard supported by all programming languages (The Official YAML Web Site 2018). The complex parameters can be described in the YAML file in order to make command line usage easier. In Figure 31, we define an YAML file that implements similar functionality as listed in the previous example.

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: edgeagent
spec:
    replicas: 2
spec:
    containers:
    - image: "edgeagent:v1"
      imagePullPolicy: IfNotPresent
      name: edgeagent
      ports:
         - containerPort: 80
      restartPolicy: Always
```

Figure 31. Parameters can be stored in a YAML file to facilitate the deployment

We manage the deployment of containerized applications by employing kubectl create command that constructs a resource based on the defined YAML file. Figure 32 depicts a deployment of two machine learning agents in the edge cloud using the previous YAML file.

```
kubectl create -f edgeagentdeployment.yaml
```

Figure 32. Utilization of YAML files reduces the complexity of deployment in a command line

Usage of the YAML file format enables a flexible development, deployment and management of containerized applications with Kubernetes. We utilize Kubernetes in orchestration to facilitate the deployment of containerized applications in a cloud-based environment. Moreover, Kubernetes guarantees the desired functional performance of the applications.

5.5 Testbed

We test the performance of the machine learning agents and a data preparation module in a testbed which emulates the key features, such as a distributed cloud environment and

deployment of a miniature version of an autonomous ship. The testbed includes a miniature version of the autonomous ship which sails in a water pool. The ship collects sensor and location data into a database where the data preparation module processes the collected data. In the testbed, we utilize MongoDB as a database solution which is an open-source Non-Structured Query Language (NoSQL) database program.

5.5.1 Autonomous Ship Prototype

We build a prototype of an autonomous ship from an extruded polystyrene (XPS) foam material. The ship carries a Raspberry Pi (RPI) and a portable battery. RPI is a single-chip computer which is responsible for controlling the ship and collecting the engine, sensor and location data. In this testbed, we emulate an edge cloud with the on-board RPI.

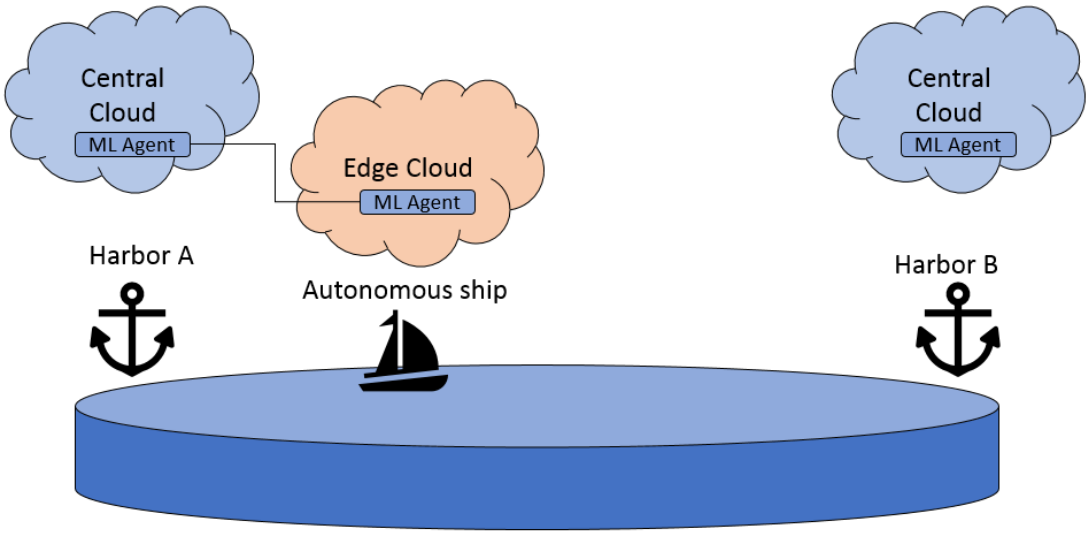


Figure 33. Autonomous ship carries a Raspberry Pi that emulates edge cloud

We use a spherical robot called Sphero as the engine of the prototyped ship. Sphero has an application programming interface (API) for JavaScript which enables programmable control (Sphero Docs 2018). Control unit, in the RPI, utilizes the API to adjust the performance of the Sphero according to the results of the machine learning. In addition to Sphero, we facilitate the control of the autonomous ship by installing a servomotor that controls a rudder attached to the end of the ship. The RPI in the ship has a control

application (“control unit”) that utilizes the rudder to turn the ship and to stabilize ship’s movement.

5.5.2 Functionality of the Prototype

We program the prototyped ship to travel between two harbors in a pool. The prototype recognizes harbor areas in advance so that the prototype has time to correct and adjust its movement using the Sphero and the rudder. Outside of the harbor area, the prototype sails towards the next harbor in an energy-efficient manner.

The ship prototype has two functional states: sailing and docking. In the sailing state, the prototype travels longer periods with optimized engine performance. Respectively, in docking state, the desired actions of the prototype are careful and accurate so that the ship docks safely to the harbor and tries to avoid collisions. These states enable the testing of the state-specific learning of the machine learning agents. It is worth noting that the states are expandable to cover multiple other scenarios such as emergency and low energy states.

5.5.3 Distributed Cloud Environment

In this testbed, we simulate the distributed cloud environment by utilizing a Raspberry Pi (RPI) and a Dell’s Latitude e7470 Ultrabook laptop. The RPI emulates the edge cloud. The laptop, on the other hand, is capable of providing larger amount of computational resources than the RPI so the laptop emulates a central cloud. A machine learning agent is able to utilize both of the emulated cloud environments in decentralized machine learning if the demand for additional computational resources increases.

5.6 Summary

In this chapter, we implemented the machine learning agent, the data preparation module and the testbed. We programmed the components in Python by utilizing two main

libraries: NumPy and pandas. These libraries provided us the tools for scientific computation which facilitated the implementation of machine learning and deduction logic. The functionality of the implemented applications is based on the mathematical methods that were introduced in section 4.4.

We also virtualized the machine learning agent and the data preparation module by composing the programmed software components, required libraries and execution runtime environments in Docker containers. Containerization facilitates the portability of the software and efficient deployment via Kubernetes.

In the next chapter, we evaluate the implemented machine learning agent and data preparation module by utilizing the testbed. The testbed is a test environment which emulates the main features of a production environment. The testbed also enables comprehensive evaluation of the implemented components and their functionality.

6 ANALYSIS AND EVALUATION

In this chapter, we test and evaluate the functionality, optimization and interoperability of the machine learning agent and the data preparation module. We utilize the implemented testbed to evaluate sailing and docking of a single autonomous ship in a water pool. The ship monitors and collects the measured sensor, engine and location data which we use to evaluate the machine learning agent and the data preparation module. In the collected sensor data, the power level of the ship engine alternates periodically which increases the coverage of the response events in data.

We evaluate the features of the machine learning agent and the data preparation module by comparing the real outputs to the predictions that are based on learning. In addition to the comparison, we also evaluate the fitness of operational performance.

6.1 Evaluation of Functionality

We evaluate the functionality of the data preparation module by comparing the prepared data to the corresponding raw data. We regard the functionality to be successful if the data preparation module fulfils the criteria of providing clean and complete data.

Evaluation of machine learning, on the other hand, demonstrates the veracity of the deduction results. We evaluate that the machine learning agent is able to utilize and validate regression based learning as intended. In validation of regression, the agent automatically calculates the least squares estimation of the learned regression model and selects the model that produces the smallest sum of squared deviations. In this section, we support and visualize the evaluation by utilizing graphs.

6.1.1 Evaluation of Data Preparation Module

We test and evaluate the functional performance of the data preparation module by utilizing raw and unprocessed data from the testbed. We specify the following evaluation

criteria to evaluate the correctness of the data preparation module: 1) The prepared data should have reduced amount of redundant information 2) Prepared data should be not corrupted 3) The amount of noise should be decreased in prepared data 4) The pattern of the data should be preserved and enhanced.

We evaluate the data preparation module by using value 24 as the interval (order) of moving average filter because the autonomous ship collects data three times per second, and the data preparation module calculates the average value of data within 8 seconds intervals. Figure 34 depicts the voltage and velocity data before and after being processed by the data preparation module.

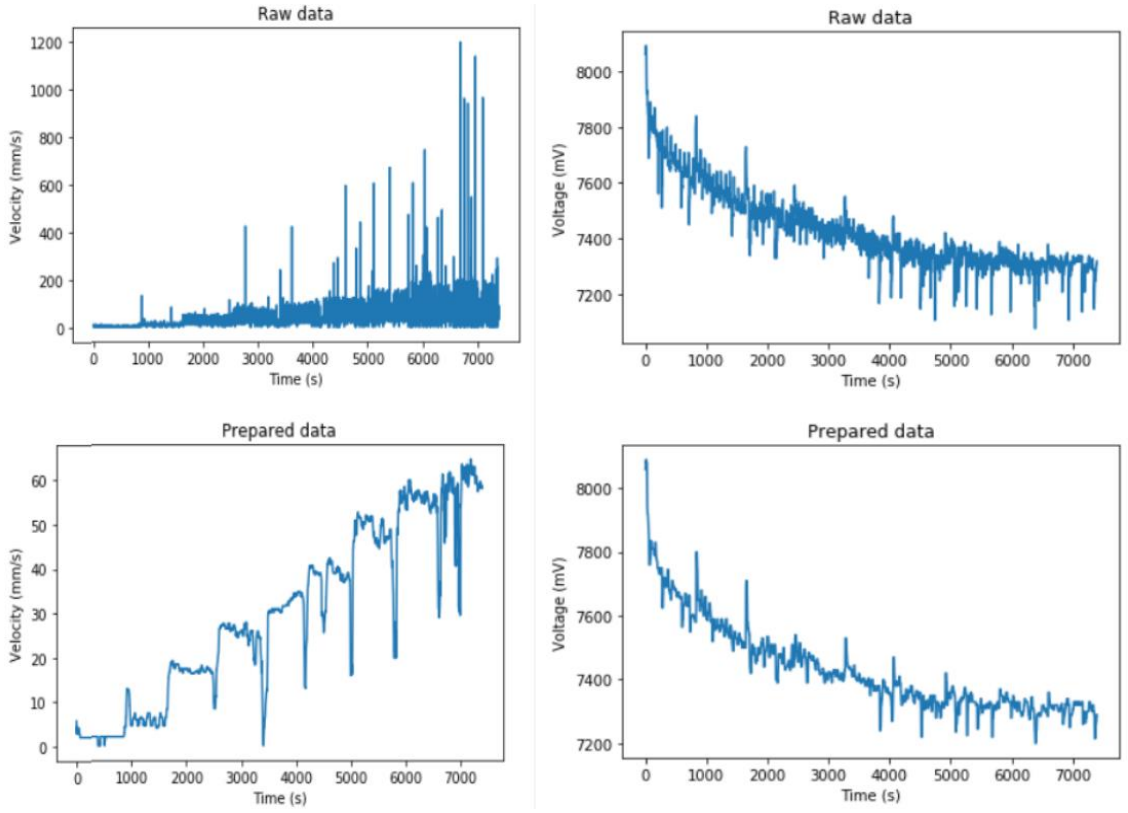


Figure 34. The upper graphs represent raw system data and the lower graphs represent the results of data preparation

Figure 34 depicts that the graphs with prepared data have no gaps and simultaneous occurrences of data points do not exist at the same moment in time. In other words, the prepared data has only unique values and the prepared data does not include any missing

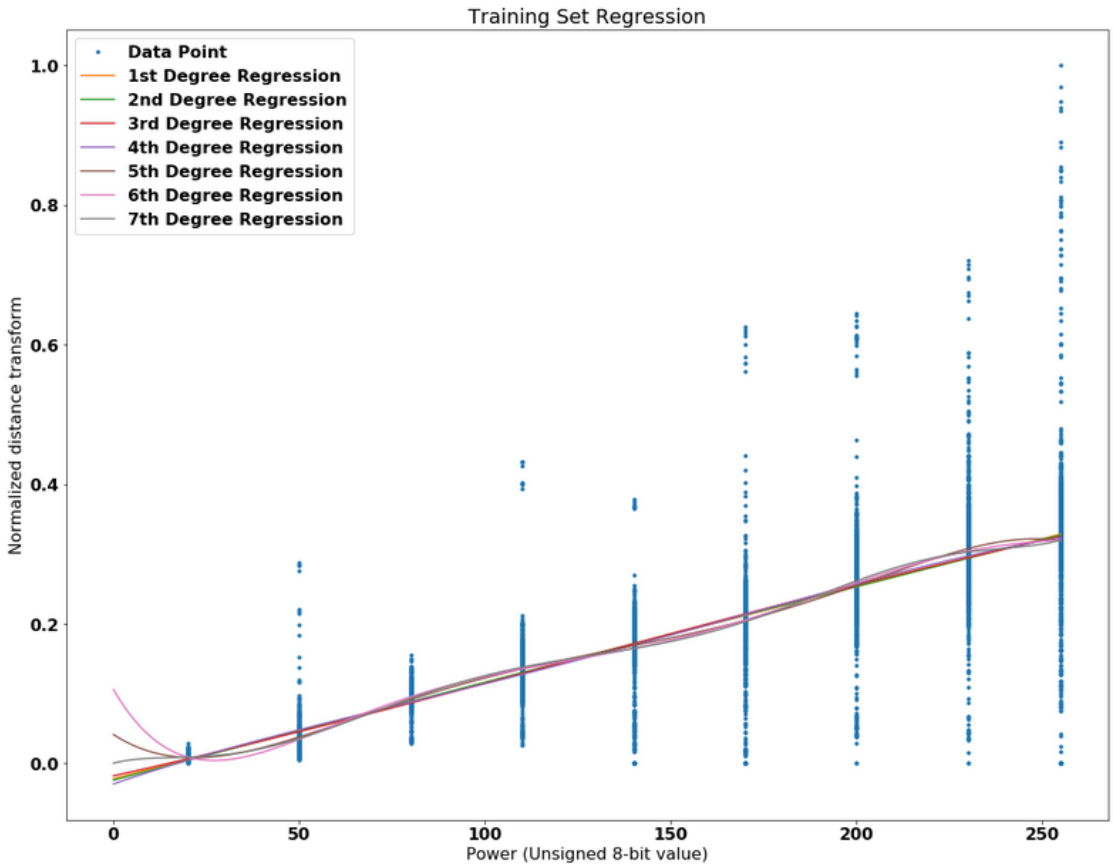
values. In the prepared data, the noise filtering has also been successful due to reduction of unwanted anomalies. Especially, in the Figure 34, the velocity in the raw data includes multiple unrealistic values that do not exist in the velocity of the prepared data.

The pattern of the raw data is not clearly recognizable, especially in the case velocity data, which hinders the reliability of machine learning and analytics. However, the data preparation module has preserved but actually enhanced the pattern of prepared data which is now significantly recognizable than in raw data. The velocity graph of the prepared data includes clear and repeating downward spikes that indicate when the power level of the ship has changed (Figure 34). The spikes show the effect of gliding where the ship had brief moments of free motion especially after acceleration when the engine was turned off.

6.1.2 Evaluation of Machine Learning

We evaluate the correctness of machine learning by comparing the selected model of the machine learning agent to multiple pre-defined regressions models. The machine learning agent strives to find the best fitting model by utilizing least squares estimation where the agent selects the model which results the smallest sum of squared deviations. The agent utilizes prepared data, that is provided by the data preparation module, in regressions.

We evaluate machine learning by training the agent with collected data that consists of information about the achieved velocity and the power usage of the tested ship. We validate that the agent selects the best fitting model by calculating sums of squared deviations of the regression models that have degrees from one to seven. We chose to evaluate degrees only from this range but it is a configurable setting in the agent. Figure 35 depicts graphs of regressions and the sums of squared deviations.




```
Sum of squared deviations with degree 1: 0.43011080281100345
Sum of squared deviations with degree 2: 0.4375230033365617
Sum of squared deviations with degree 3: 0.416220805878262
Sum of squared deviations with degree 4: 0.45635504594212467
Sum of squared deviations with degree 5: 0.25090833928219913
Sum of squared deviations with degree 6: 0.14131552654346244
Sum of squared deviations with degree 7: 0.3582108580154373

Best fit found at: 6th degree
```

Figure 35. By comparing multiple regression models, the machine learning agent selects the best fitting regression model with a degree of six

The agent selects correctly the sixth degree regression model which produces the smallest sum of squared deviations (Figure 35). One may also observe that the selected regression model has a slightly exceptional behavior between the known data points which can be fixed by expanding the training set with additional power values.

6.1.3 Evaluation of Decentralized Learning

We evaluate the correctness of the functionality of the decentralized learning by comparing the results of decentralized learning to the results of a single machine learning agent. In this comparison, we provide identical data sets for decentralized and local machine

learning systems and we expect exactly the same results. Both machine learning systems have the same learning objective where they strive to find the optimum value for power usage of the engine. Figure 36 depicts the learning procedure of a single machine learning agent that does not utilize decentralization.

```
#-- Machine Learning Agent -
Best fit found at 7th degree
Found optimal power level value: 205.76511509951138
```

Figure 36. Results of a single machine learning agent

In this evaluation, we distribute learning to be deployed in two machine learning agents that are located in edge and central clouds. The agent in the edge cloud requests assistance from the agent in the central cloud. Figure 37 depicts how the agent in central cloud detects a requesting agent in a feasible range.

```
#-- Machine Learning Agent of a Central Cloud -
Detecting agents in feasible range..
Request found!
Starting decentralized learning of degreed 3 to 7
Best fit found at 7th degree
Found optimal power level value: 205.76511509951138
Sending parameters to requesting agent..
Parameters sent successfully
```

Figure 37. Results of a machine learning agent that is deployed in the central cloud

The requesting agent sends information about the learning objective to the assisting agent. In addition, the requesting agent is responsible for dividing the learning task among the agents that contribute in decentralized learning. Finally, the requesting agent receives and aggregates the results of the learning task which is depicted in Figure 38.

```
#-- Machine Learning Agent of an Edge Cloud -
Requesting assistance from other agents..
Found assisting agent in address: http://192.168.2.131:8080/re-
questAssistance

Allocating local learning of degrees from 1 to 2
Allocating decentralized learning of degrees from 3 to 7
Best fit found at 1st degree
Found optimal power level value: 255

Waiting for other agents to finish learning..
Aggregating learning results..
Best fit found at 7th degree
Result of decentralized learning for optimal power usage:
205.76511509951138
```

Figure 38. Results of a requesting machine learning agent that is deployed in the edge cloud

As we observed previously, a single machine learning agent found the best fitting model in 7th degree and the optimal value of 205.765 by utilizing local learning. These results match with the results of decentralized learning so we can conclude that the distribution and aggregation of learning tasks works as expected.

6.2 Evaluation of Optimized Performance

In this section, we evaluate the effect of optimization of the machine learning agent. We utilize data collected from the testbed where an autonomous ship was sailing for four hours. The autonomous ship sailed between two harbors and the ship took advantage of two different states: sailing and docking. In both states, we alternated the power of the engine in every five minutes to broaden the coverage of the response events in the collected data.

The machine learning agent learns the optimum degree and parameters of the best fitting regression model from the training sets of examples. We evaluate the optimized performance by comparing learning results to measured values. From the findings of the comparison, we can evaluate if machine learning truly improves the performance of the ship.

6.2.1 Evaluation of State-Specific Machine Learning: Sailing

In sailing state, a machine learning agent strives to optimize the power usage of a ship engine in such a way that the energy consumption is minimized and the velocity is as high as possible. This multi-objective optimization (Pareto optimization) prolongs the sailing distance and supports overall energy efficiency.

The machine learning agent learns the best fitting regression models that describe the structural patterns of changing distance and voltage. In learning, the agent utilizes a training set of examples which consists of sailing with different power levels in fixed periods of time. The electric engine, of an autonomous ship prototype, utilizes unsigned 8-bit values to represent the magnitude of the power usage where 0 is the minimum power usage and 255 is the maximum. Figure 39 depicts the best fitting regression models that the agent derives from the training set, and also two additional iterations of learning.

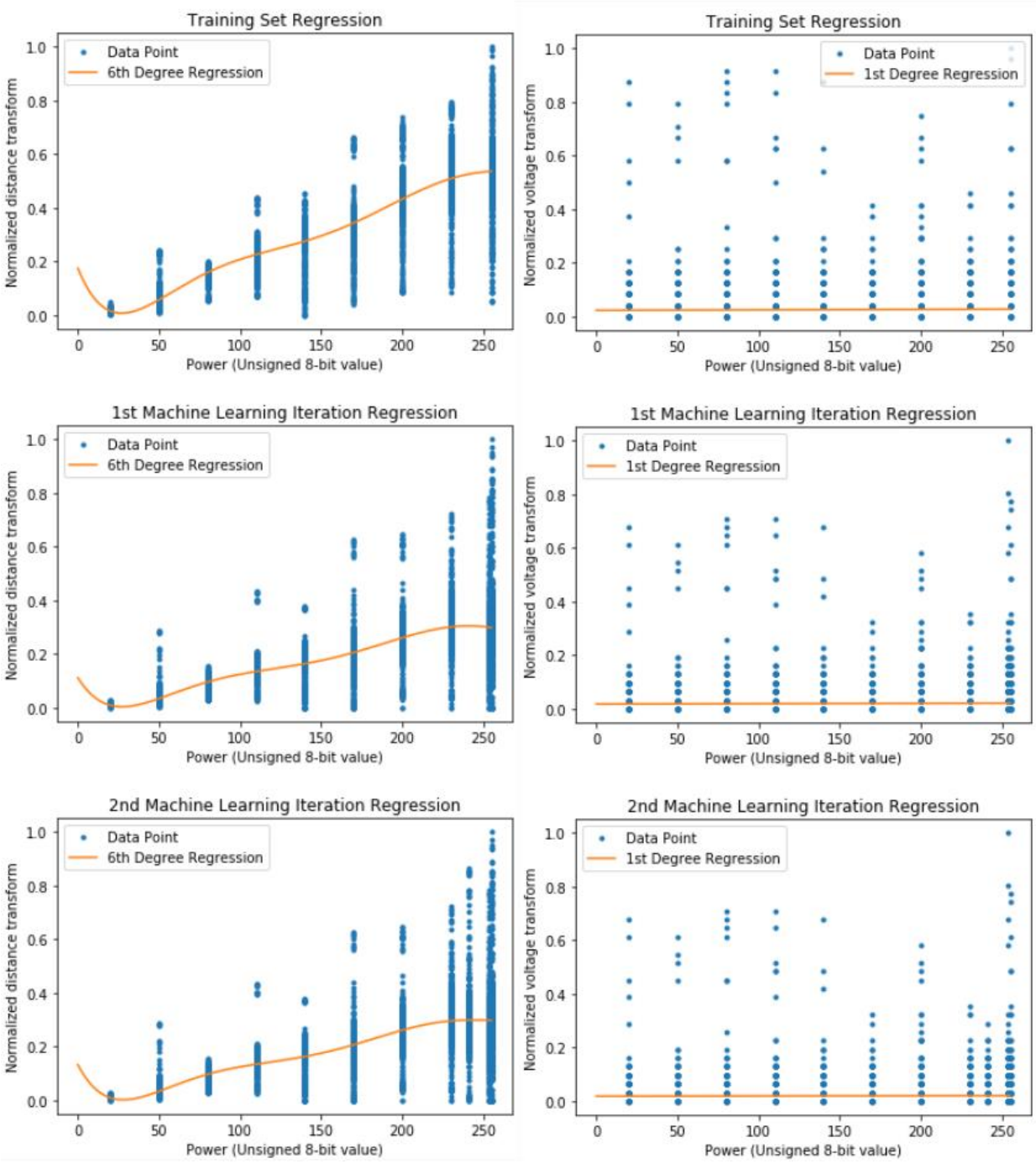


Figure 39. The global optimum convergences during iterations of machine learning

The agent finds best fitting regression models for the distance transform, which is the depended variable, when the degree has the value of six in every iteration. The agent finds the root derivative in order to discover the global optimum with maximum distance transform. The agent finds the global optimum value of 254 from the regression calculated

from the training set but the global optimum value converges to 241 after couple of machine learning iterations (Figure 39).

The agent finds the best fitting regression models respectively for voltage transform when the degree has the value of one. In this optimization, the objective is to find the global optimum where the voltage transform is minimal. As a result, the agent finds the global optimum value that converges to zero (Figure 39).

After the best fitting regression models are found, the agent aggregates the different distance-power efficiency ratios to find the most optimal solution in this multi-objective optimization. Generally, the agent aggregates the efficiency ratios by calculating and comparing the distance-power efficiency ratios of multiple feasible regression model functions of global optimum, but, in this evaluation, we calculate ratios for each tested power level (Figure 40).

```
Efficiency ratios of trained power levels:

Efficiency ratio with power level 20: 0.5087251892035706
Efficiency ratio with power level 50: 1.8381502685268921
Efficiency ratio with power level 80: 5.0414485985267292
Efficiency ratio with power level 110: 6.939491120090184
Efficiency ratio with power level 140: 8.234854280434233
Efficiency ratio with power level 170: 10.10325595371303
Efficiency ratio with power level 200: 12.53758467667819
Efficiency ratio with power level 230: 14.09660897191777
Efficiency ratio with power level 255: 13.84067482366743

Efficiency ratios of learned power levels:

Efficiency ratio with power level 254: 13.89075916124045
Efficiency ratio with power level 241: 14.27218972427181
```

Figure 40. Comparison of distance-power efficiency ratios with different power levels

Efficiency ratio is the highest when the best fitting regression models receive a value that converges to 241. The power value of 241 is the most optimal if we want to maximize the distance transform with the minimized voltage transform in the tested ship. If the shape or weight of the ship changes, additional machine learning would be necessary to perform

so that the most optimal power value for a certain ship (possibly with different amount of cargo) can be found.

6.2.2 Evaluation of State-Specific Machine Learning: Docking

In docking state, an autonomous ship detects a nearby harbor and the ship adjusts its movement to support smooth docking. A machine learning agent pursues to optimize the power usage of the ship engine in two ways: docking time is minimized and the ship does not sail past the harbor. If the ship collides or sails past the harbor, the ship receives a penalty which the agent adds in the measured docking time.

Similarly as in the previous section, the machine learning agent learns the best fitting regression model that depicts the relation between different power levels and the corresponding docking times. In this evaluation, we collected data using different power levels where we ran four docking procedures per selected power level. Figure 41 illustrates the best fitting regression models that the agent discovered by learning.

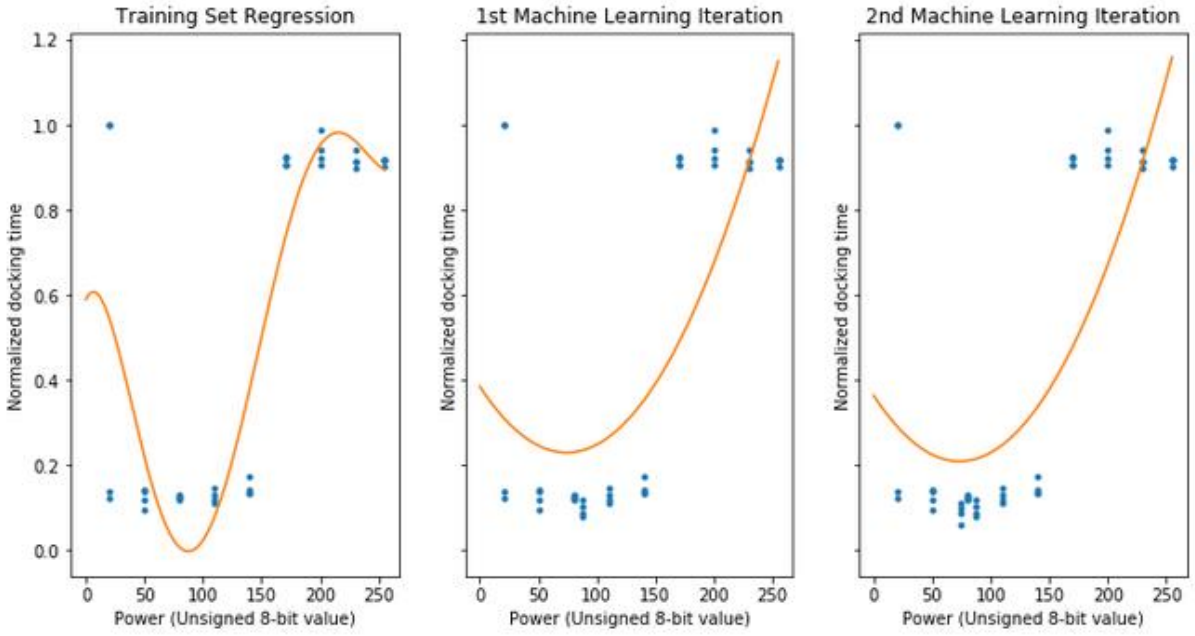


Figure 41. The degree of the best fitting regression model stabilizes when iterations in learning increases

The agent learns during training the best fitting regression model when the degree has value of five, but, after couple of iterations of learning, the agent finds the best regression model when the degree has value of two. Hence, we see that the regression model stabilizes when the global optimum converges. In this evaluation, the agent finds the initial global optimum value of 87, but, after the regression model stabilizes, the global optimum value converges to 74 (Figure 41).

Figure 42 depicts coefficients that represents the combination of minimized docking time and risk of sailing past harbor. The smaller the coefficient is, the more selected power level fulfils the learning objective.

```
Coefficients of trained power levels:

Coefficient with power level 20: 0.29131543243480684
Coefficient with power level 50: 0.22514691115577656
Coefficient with power level 80: 0.21074554392906975
Coefficient with power level 110: 0.2481113307546865
Coefficient with power level 140: 0.33724427163262677
Coefficient with power level 170: 0.47814436656289044
Coefficient with power level 200: 0.6708116155454777
Coefficient with power level 230: 0.9152460185803886
Coefficient with power level 255: 1.1584857082327833

Coefficients of learned power levels:

Coefficient with power level 87: 0.21483394318703364
Coefficient with power level 74: 0.20948444505022523
```

Figure 42. Comparison of coefficient values with different power levels

Coefficient has the lowest value when the selected regression model receives the value of 74. Consequently, we can state that the power value of 74 is the most optimal value when we desire to reduce the docking time and avoid the risk of sailing past the harbor.

6.3 Evaluation of Virtualized Agent

We evaluate that the virtualized machine learning agent and virtualized data preparation meet their design criteria in this section. To evaluate virtualization, we run containerized machine learning agents, and observe if the agents are able to interoperate in different environments and conduct decentralized machine learning by utilizing prepared data. Containerized applications run in parallel where they utilize HTTP -based communication so that, in addition to functional interoperability, we evaluate the agent's ability to perform inter-process communication (IPC).

We also evaluate the performance of the selected orchestration system. We implemented the deployment of containerized applications by employing Kubernetes which we evaluate also in this section. Furthermore, we evaluate Kubernetes' capability to restore the desired status of the containers upon their failure.

6.3.1 Evaluation of Container Interoperability

Container interoperability refers to operational reliability of containerized applications in different deployment environments. We evaluate the functionality of the containerized applications and that building of docker images works as expected. Figure 43 illustrates that the images are built successfully and published in the Docker repository and they can be pulled by worker nodes.

```
REPOSITORY      TAG       IMAGE ID       CREATED      SIZE
centralagent    v1        754435a5e51d   11 min        259MB
edgeagent       v1        b4651e59b841   12 min        259MB
datamodule      v1        e03ad24323eb   13 min        258MB
```

Figure 43. Built docker images of the machine learning agents and the data preparation module are published in the Docker repository

We evaluate the performance of the decentralized machine learning by running containerized machine learning agents in distributed cloud environment. We regard the machine learning results to be successful if decentralized learning (in containers) produces the same optimum target value of 241 for power usage in sailing similarly as described

previously in section 6.2.1. If the produced value differs from this target, we would notice interoperability problems either in decentralized learning or in the data preparation module. Figure 44, depicts the functionality of the containerized machine learning agent located in an edge cloud.

```
#-- Containerized Agent in an Edge Cloud
Requesting assistance from other agents..
Found assisting agent in address: http://192.168.2.131:8080/re-
questAssistance

Allocating local learning of degrees from 1 to 2
Allocating decentralized learning of degrees from 3 to 7
Best fit found at 1st degree
Found optimal power level value: 255
Waiting for other agents to finish learning..
Aggregating learning results..
Best fit found at 6th degree
Result of decentralized learning for optimal power usage:
241.00216610416206
```

Figure 44. Results of a containerized machine learning agent that is deployed in the edge cloud

Figure 45 describes, respectively, the functionality of the containerized machine learning agent located in a central cloud.

```
#-- Containerized Agent in a Central Cloud –
Detecting agents in feasible range..
Request found!
Starting decentralized learning of degreed 3 to 7
Best fit found at 6th degree
Found optimal power level value: 241.00216610416206
Sending parameters to requesting agent..
Parameters sent successfully
```

Figure 45. Results of a containerized machine learning agent that is deployed in the central cloud

Containerized machine learning agents perform in a similar way independently of whether they are running at the edge or the central cloud. From these results we can see that the result of decentralized learning matches with the target value and we can conclude that the behavior of the machine learning agents is identical in the containerized and non-containerized versions.

6.3.2 Evaluation of Orchestration

We evaluate container deployment and tolerance against failures in Kubernetes based distributed cloud environment. In order to evaluate the operational success, Kubernetes should be able to deploy the containers that we have defined in YAML files and ensure their proper operational functionality. Moreover, Kubernetes should be able to recover containerized applications after malfunctions. Figure 46 describes how we are able to deploy containers from YAML files by utilizing Kubernetes.

```
NAME          DESIRED  CURRENT  UP-TO-DATE AVAILABLE
centralagent  1        1        1          1
edgeagent     1        1        1          1
datamodule    1        1        1          1
```

Figure 46. Overview of containers that are deployed and orchestrated by Kubernetes

Next we observe the status of Kubernetes pods and how the Kubernetes responses when unwanted events occur. Figure 47 depicts the information of Kubernetes pod immediately after the deployment.

```
NAME          READY  STATUS     RESTARTS  AGE
centralagent  1/1    Running    0         23 min
edgeagent     1/1    Running    0         24 min
datamodule    1/1    Running    0         27 min
```

Figure 47. Overview of the deployed containers in the Kubernetes pod

We cause connectivity and environmental failures, on purpose, to test how Kubernetes manages orchestration when malfunctions occur. In Figure 48, Kubernetes handles a malfunction when the containers cannot be run.

```
NAME          READY  STATUS         RESTARTS  AGE
centralagent  0/1    ErrImagePull   2         33 min
edgeagent     0/1    Error          2         34 min
datamodule    0/1    Error          2         37 min
```

Figure 48. Overview of the runtime failures in containers

After the connectivity and the environmental states are in a stable, Kubernetes manages to restart pods and recover from the malfunction as illustrated in Figure 49.

```
NAME          READY   STATUS     RESTARTS   AGE
centralagent  1/1     Running    3          37 min
edgeagent     1/1     Running    3          38 min
datamodule    1/1     Running    3          41 min
```

Figure 49. Overview of the recovered containers that are running again

Kubernetes is autonomously able to deploy, maintain and recover containerized machine learning agents and data preparation module successfully. Under these circumstances, we can evaluate that the selected orchestration system supports the deployment of containerized applications in a distributed cloud environment.

6.4 Summary

In this chapter, we evaluated the functionality, optimization capability and virtual implementation of the machine learning agent and the data preparation module. We evaluated basic functionality of the implemented applications by comparing the results of the different agent configurations to the base line scenario. According to our tests, the machine learning software components work as expected.

After we had evaluated the functionality, we evaluated the optimized performance of an autonomous ship by utilizing a machine learning agent. The agent was assigned to handle two state-specific machine learning tasks where we evaluated that the machine learning agent was able to find the most optimal solution. We compared the results of machine learning to the measurement results. Based on comparison, we evaluated that machine learning improved the state-specific performance and thus machine learning optimizes the overall system performance.

Finally, we evaluated the virtualized implementation of machine learning agent and data preparation module. In evaluation of the virtualized implementation, we evaluated the functional interoperability, inter-process communication and orchestration of the containerized applications in a distributed cloud environment. The results of evaluation of the virtualized implementation indicate that the agent and the data preparation module

operate and communicate as expected in containers and they can be deployed by utilizing Kubernetes. In addition to deployment, Kubernetes is able to automatically restore the containerized applications after malfunctions and maintain the desired states of the containers.

7 CONCLUSION AND DISCUSSION

In this chapter, we discuss about the advantages and disadvantages of the decentralization of machine learning in a distributed cloud environment based on the findings in this thesis. We review the efficiency of machine learning based optimization, the potential in decentralized learning and improved portability and fault tolerance introduced by containerized deployment.

In this thesis, we extracted hidden information from the data by utilizing machine learning. The extracted hidden information provided more knowledge about the system performance than we could deduct from the data sets with heuristic approximations. In addition to the extraction of the hidden information, machine learning can optimize the performance of the autonomous ship in both two states better than the best performance that the training set could introduce. However, a drawback is that learning is relatively resource demanding which reduces the amount of available resources for other operations.

We have run decentralized machine learning in parallel in two environments, one of which was a constrained environment, which shows that machine learning can also be scaled down. Decentralization allowed us to balance the load between two environments and spread the load according to the computational capabilities of the environments. Even though decentralization facilitates learning in multiple environments, it can suffer from increased overhead due to inter-process communication.

Containerized deployment of the machine learning agents and data preparation module fostered portability of the applications in distributed cloud environment. In addition to portability, containerization enabled us to use an orchestration system to automate the deployment, maintenance and fault tolerance of the running applications. Kubernetes, as an orchestration system, served the purpose of Docker container orchestration which enabled the deployment of our containerized applications in an automated way.

In this thesis, we proposed an alternative solution for decentralized and parallel machine learning by utilizing a distributed cloud environment. We defined requirements for

decentralized machine learning that fosters microservice and “As a Service” -oriented development. In addition to development of the related requirements, we introduced a real life use case where decentralized machine learning was required to be interoperable in a multi-cloud environment and to provide optimized performance. Since decentralized machine learning was deployed in the distributed cloud environment, we developed decentralized machine learning software to support containerization which facilitates the portability and supports Kubernetes based orchestration.

We designed and implemented a machine learning agent and a data preparation module which are independent software applications that can be run in containers. Containerization of applications facilitates the portability in the distributed cloud environment and enables the orchestration via Kubernetes. The machine learning agent is responsible for the performance of decentralized machine learning operations and maintaining inter-process communication among other agents. Data preparation module, on the other hand, ensures that the data for machine learning is filtered, nonredundant and complete.

We evaluated success in functional and virtual performance of the implemented machine learning agent and data preparation module. In addition to functional and virtual performance, the machine learning agent was also able to improve overall system performance of an autonomous ship by taking advantage of machine learning. The ship had two possible functional state where the agent could distinguish different states and conduct state-specific machine learning. State-specific learning enables the agent to perform more advantageous deductions since the agent can separate the state-specific learning objectives. In this thesis, we managed to improve the overall system performance by utilizing regression which is one of the supervised learning techniques.

For further development, we would propose following improvements to be considered: support for additional machine learning techniques, support for additional data filtering techniques, enhanced ability to monitor cloud metrics, alternative solution with MapReduce framework and support for unikernel deployment. Support for additional technologies would increase the coverage of use cases for machine learning based optimization. Enhanced ability to monitor cloud metrics would benefit machine learning agent to detect

the capability of deployment environment more precisely. Finally, machine learning agents could be developed to support unikernel deployment which fosters light-weight execution of the applications.

REFERENCES

Allison Paul D. (2001) *Missing Data*

Alpaydin Ethem (2010) *Introduction to Machine Learning*.

Andreas Hammar (2014) *Analysis and Design of High Performance Inter-core Process Communication for Linux*.

Anthony Martin (2011) *Undergraduate study in Economics, Management, Finance and the Social Sciences.* [online]. [Referred on 4.6.2018]. Available at : http://www.met.edu/Institutes/ICS/NCNHIT/papers/39.pdf

Bass Len, Clements Paul & Kazman Rick (2003) *Software Architecture in Practice*.

Bijuraj L. V. (2013) *Clustering and its Applications*.

Brink Henrik, Richards Joseph W., Fetherolf Mark (2017) *Real-World Machine Learning.*

Docker Guide Documentation (2018). *About images, containers, and storage drivers*. [online]. [Referred on 22.5.2018]. Available at: https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/

Docker Product Manuals (2018). *Docker Enterprice Edition Platform.* [online]. [Referred on 21.5.2018]. Available at: https://docs.docker.com/ee/

Ericsson (2018) *The Ericsson Mobility Report*. [online]. [Referred on 24.7.2018]. Available at: https://www.ericsson.com/en/mobility-report/mobility-visualizer?f=1&ft=1&r=2,3,4,5,6,7,8,9&t=8&s=1,2,3&u=1&y=2017,2023&c=1

Gilbert Strang (1991) *Calculus*.

Google Cloud Platform Documentation (2018) *Kubernetes Engine: Container Cluster Architecture*. [online]. [Referred on 26.5.2018]. Available at: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture

Hwang Kai & Chen Min (2017) *Big-Data Analytics for Cloud, IoT and Cognitive Computing*.

Juniper Networks (2018) *What is a Docker container*. [online]. [Referred on 4.6.2018]. Available at: https://www.juniper.net/us/en/products-services/what-is/docker-container/

Katyal Mayanka & Mishra Atul (2015) *Orchestration of Cloud Computing Virtual Resources*. [online]. [Referred on 7.6.2018]. Available at: https://ieeexplore.ieee.org/document/7019756/

Kena Alexander, Choonhwa Lee, Eunsam Kim & Sumi Helal (2017) *Enabling End-to-End Orchestration of Multi-Cloud Applications*. [online]. [Referred on 29.5.2018]. Available at: https://ieeexplore.ieee.org/document/8008766/

Kubernetes Documentation (2018) *Concepts: Overview*. [online]. [Referred on 26.5.2018]. Available at: https://kubernetes.io/docs/concepts/overview/

Lucky R.W. (1968) *Adaptive redundancy removal in data transmission*.

Lukša Marko (2018) *Kubernetes in Action*.

Marinescu Dan C. (2013) *Cloud Computing: Theory and Practice*.

NumPy v1.13 Manual (2018) *Numpy and Scipy Documentation*. [online]. [Referred on 27.7.2018]. Available at: https://docs.scipy.org/doc/

Pandas 0.23.2 Documentation (2018) *pandas: powerful Python data analysis toolkit*. [online]. [Referred on 27.7.2018]. Available at: http://pandas.pydata.org/pandas-docs/stable/

Press William H., Teukoisky Saul A., Vetterling William T. & Flannery Brian P. (2007) *Numerical Recipes: The Art of Scientific Computing*.

Rawlings John O., Pantula Sastry G. & Dickey David A. (1988) *Applied Regression Analysis: A Research Tool*.

Rodger Richard (2018) *The Tao of Microservices*

Rolls-Royce (2016) *Autonomous Ships: The Next Step*. [online]. [Referred on 2.6.2018]. Available at: https://www.rolls-royce.com/~/media/Files/R/Rolls-Royce/documents/customers/marine/ship-intel/rr-ship-intel-aawa-8pg.pdf

Scott Charlie, Wolfe Paul & Erwin Mike (1999) *Virtual private networks*.

Singh Jatinder, Bacon Jean, Crowcroft Jon, Madhavapeeddy Anil, Pasquier Thomas, Hon W. Kuan & Millard Christofer (2014): *Technical Report of University of Cambridge: Regional clouds: technical considerations*. [online]. [Referred on 7.6.2018]. Available at: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-863.pdf

Smola Alex & Bishwanathan S.V.N (2008) *Introduction to Machine Learning*.

Sosinsky Barrie (2011*) Cloud Computing Bible*.

Sphero Documentation (2018) *Javascript API*. [online]. [Referred on 3.7.2018]. Available at: https://sdk.sphero.com/community-apis/javascript-sdk/

Stefan Poslad & Patricia Charlton (2001) *Standardizing Agent Interoperability: The FIPA Approach*. [online]. [Referred on 11.6.2018]. Available at:

http://www.eecs.qmul.ac.uk/~stefan/publications/2001-LNCS-standardising-agent-in-teroperability.pdf

Stuart J. Russel & Peter Norvig (1995) *Artificial Intelligence A Modern Approach*.

Sutton Richard S. & Barto Andrew G. (2017) *Reinforcement Learning: An Introduction*.

Tan Li & Jiang Jean (2013) *Digital signal processing: fundamentals and applications*.

The Official YAML Web Site (2018) [online]. [Referred on 12.7.2018]. Available at: http://yaml.org/

Wang Can, Zhang Sheng, Zhang Huyin, Qian Zhuzhong & Lu Sanglu (2017) *Edge Cloud Capacity Allocation for Low Delay Computing on Mobile Devices*. [online]. [Referred on 29.7.2018]. Available at: https://ieeexplore.ieee.org/document/8367279/

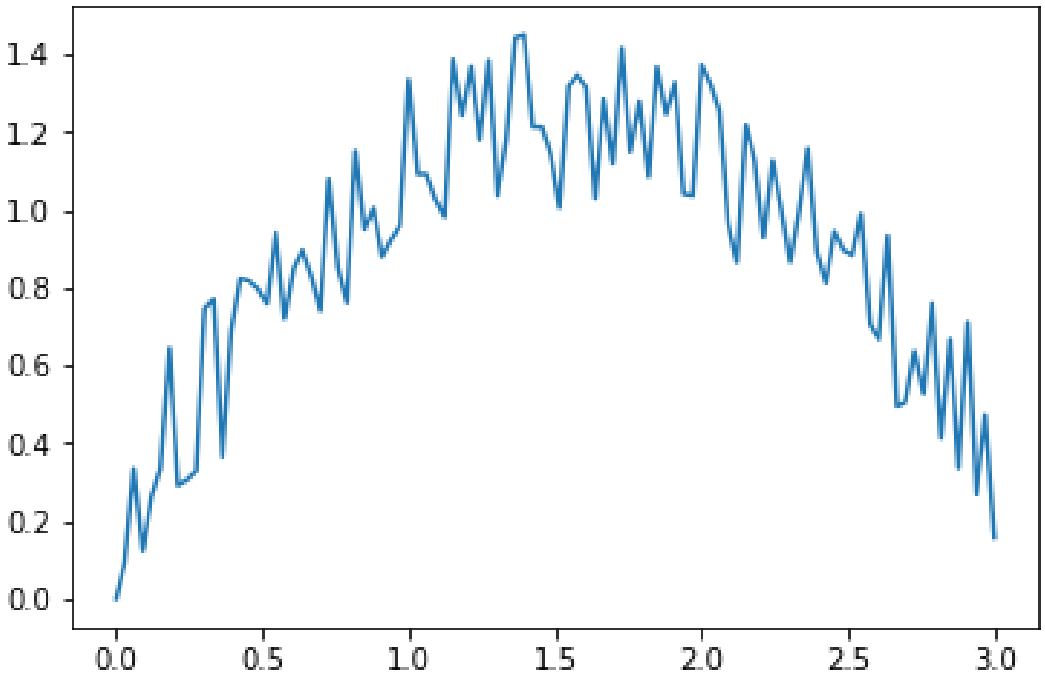
Zhang Xiujun, Liu Keqin, Liu Zhi-Ping, Duval Béatrice, Richer Jean-Michel, Zhao Xing-Ming, Hao Jin-Kao Hao & Chen Konan (2013) *Bioinformatics*.

Zhu Hong, Bayley Ian (2018) *If Docker Is The Answer, What Is The Question*. [online]. [Referred on 29.5.2018]. Available at: https://ieeexplore.ieee.org/document/8359160/

8 APPENDIX

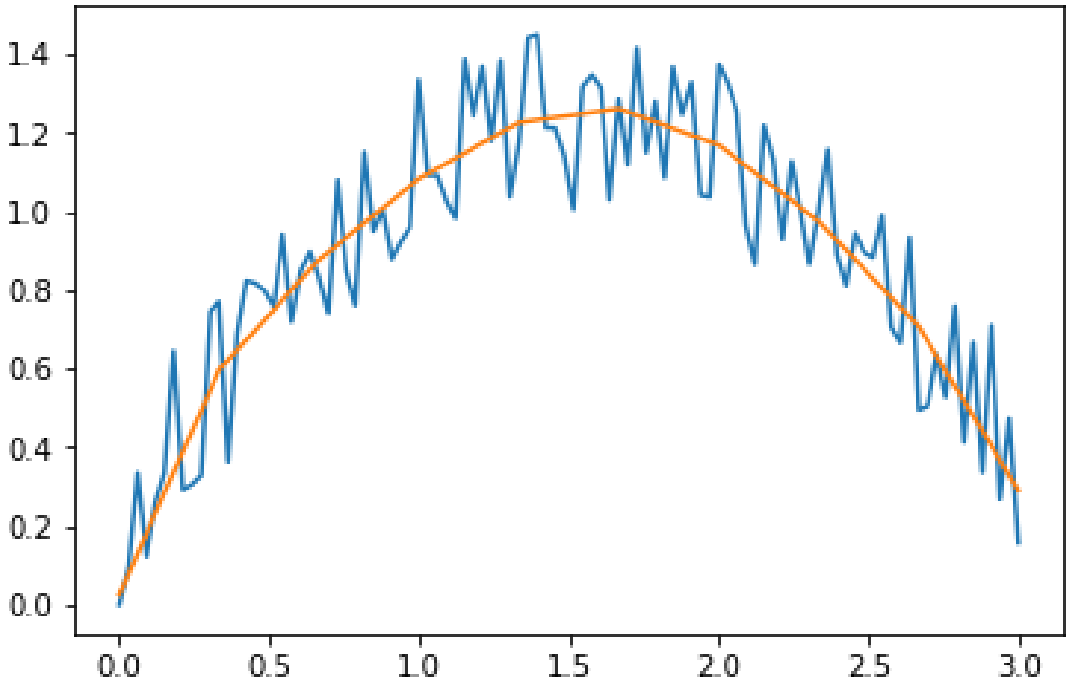
A. Alternative Solution with TensorFlow

```
# Joel Reijonen (joel.reijonen@ericsson.com)

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import time

# -- Initalization of input data --

# initialization of 100 observations
n_observations = 100

# xs -> linspace returns evenly spaced numbers over a specified
# interval
xs = np.linspace(0, 3, n_observations)

# ys -> sin wave with random uniform that draws samples  #from a
uniform distribution
ys = np.sin(xs) + np.random.uniform(0, 0.5, n_observations)

#plot data
plt.plot(xs,ys)
```




```
# Let's define a function how to train our best fitting
# regression model
def TrainModel(x, y, interval):
  # This set extracts unique values from the data set
  distinctX = set(x)
  # And let’s initalize errolist for evaluation
  errorlist = []
  # Let’s make a simple for loop which goes through degrees
#that we want to observe
  for degrees in range (1, 7):
    # initialize error and the index of this degree to be zero
    error=0
    index = 0
    # fit a regression by using np.polyfit
    z = np.polyfit(x, y, degrees)
    # convert the model as polynomial
    p = np.poly1d(z)
   # Let’s make another loop to go through measurement points
   for measurement in x:
     # Least square errors:
     # Sum the squared errors
     # errors are calculated from
     error= error + (y[index]-p(measurement))**2
     #increment index -> to go through all output values
     index+=1
     #--- endloop
```

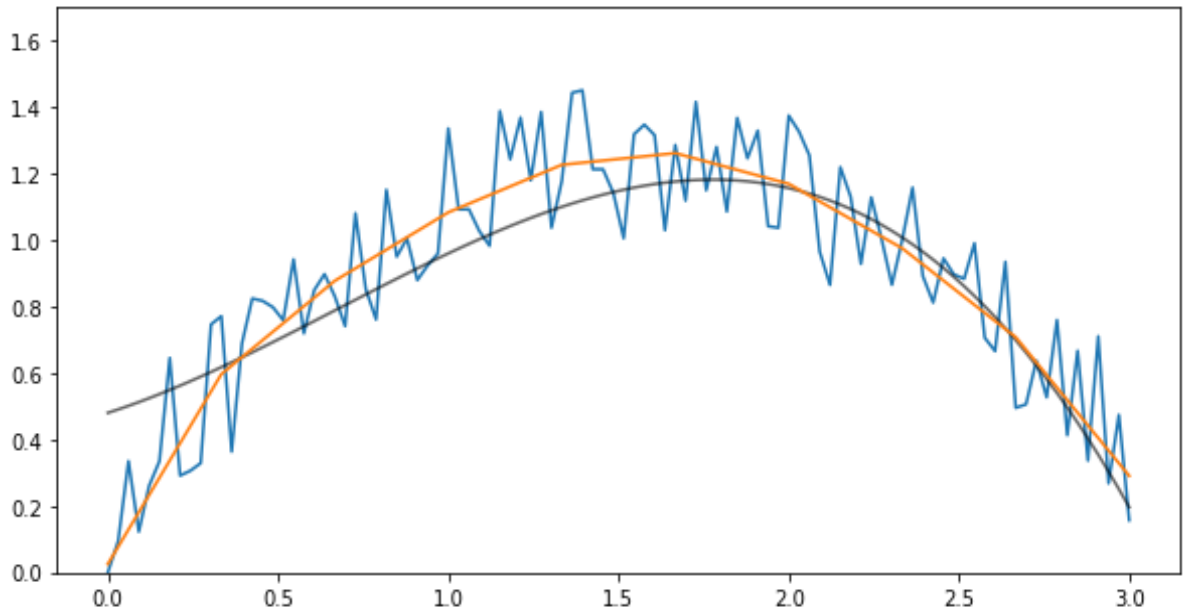
```
    # add received error to the list
    errorlist.append(error)
  #--- endloop
  # Get the degree of best fitting model that produces min
     #error
  degree = errorlist.index(min(errorlist)) + 1
  #Receive the model
  z = np.polyfit(x, y, degree)
  model = np.poly1d(z)
  #return polynomial and errorlist
  return model, errorlist;
#--- endFunction

#Lets train the model and start the timer
start= time.time() model, errorlist = TrainModel(xs,ys, [1,8])
end=time.time()

time1 = end-start
#Lets define a linspace set data to be used in plot-ting temp-
data = np.linspace(0,3,10)

#plot the original data
plt.plot(xs, ys)

#plot the regression over the original data
plt.plot(tempdata ,model(tempdata))
```



```
# Plot initial data
plt.plot(xs,ys)
# Start timer
start= time.time()
# tf.placeholders for the input and output of the network.
# Placeholders are variables which we need to fill in when we
# are ready to compute the graph
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
# Instead of a single factor and a bias, we'll create a
# polynomial function of different polynomial degrees
# We will then learn the influence that each degree of the
#input (X^0, X^1, X^2, ...) has on the final output (Y).

# Initialization of the variables of the graph (net-work)
# Let's initialize 4 weights
# In neural networks we initialize weights stochastically

# Let’s utilize tf.variables:
Y_pred = tf.Variable(tf.random_normal([1],
                     mean=0.5, stddev=0.1), name='bias')
# Utilize tf.variables through loop:
for pow_i in range(1, 4):
  W = tf.Variable(tf.random_normal([1],
                   mean=0.5, stddev=0.1),
                   name='weight_%d' % pow_i)
  # Initialize predicted output values
  Y_pred = tf.add(tf.multiply(tf.pow(X, pow_i), W), Y_pred)
  #--- endLoop

# Cost function will measure the distance between our
# observations and predictions and average over them.

# This is somehow similar cost function as we programmed
# previously in our 1st version
cost = tf.reduce_sum(tf.pow(Y_pred - Y, 2))
                      / (n_observations - 1)
# Use gradient descent to optimize W,b
# Performs a single step in the negative gradient
learning_rate = 0.05

#Lets define a gradient descent optimizer via tf.train
# Here we minimize the sum of cost
optimizer = tf.train.GradientDescentOptimizer(
                     learning_rate).minimize(cost)
```

```
# Let’s define epochs, that define size of loop in
# training
n_epochs = 10000

# We create a session to use the graph
with tf.Session() as sess:
  # Here we tell tensorflow that we want to initialize
  # all the variables in the graph so we can use them
  sess.run(tf.global_variables_initializer())

  # Fit all training data
  # Initialize previous training cost
  prev_training_cost = 0.0

  # Loop through epochs
  for epoch_i in range(n_epochs):
    # Use zip to loop through both sets of our initial data
    for (x, y) in zip(xs, ys):
      # Utilize run from initialized session
      # Feed our set of data together with optimizer
      sess.run(optimizer, feed_dict={X: x, Y: y})
      #--- endLoop
    # Update training cost by run and using cost, now feed
    # all data
    training_cost = sess.run(cost, feed_dict={X: xs, Y: ys})
    # Print updated training cost (cost should descent!)
    print('Current training cost: ', training_cost)
    # Plot in every 100 epochs
    if epoch_i % 100 == 0:
      plt.plot(xs, Y_pred.eval(feed_dict={X:
               xs}, session=sess), 'k', alpha=0.6)
    # Allow the training to quit when we have reached a
    #reasonable accuracy
    if np.abs(prev_training_cost - training_cost) < 0.001:
       break
    # Update cost
    prev_training_cost = training_cost
    #--- endSession
# Stop timer
end=time.time() time2= end-start
# Plot both of the graphs into same figure
plt.plot(tempdata, model(tempdata))
# Print ratio: Network training time divided by training
# time of our solution
print("RATIO: ", time2/time1)
('Current training cost: ', 0.03191368)
```

```
('Current training cost: ', 0.031566862)
('RATIO: ', 5.1100263354363396)
# In following graph, gray line is the regression that is
# obtained from this neural network (TensorFlow) and orange
# line is obtained from the solution that was used in this
# thesis
```




```
# Conclusions:
# Despite the fact that the neural network approach is slower
# less accurate and needs to be informed about the degree of
# regression model in advance, the neural network approach
# supports more general use cases where we do not necessarily
# know what kind of algorithm would provide feasible results.
#
# In addition, the decentralization of neural network based
# model training would be challenging to implement since
# training is dependent on the result of previous training
# iteration
```