Scalable cloud-based platform for distributed machine learning workloads
Group FAAM

Albert Gejr*, Anne Cathrine Kirkegaard*, Frederikke Lan*, Michael Fuglsang†,
*Software Engineering, University of Southern Denmark, Odense, Denmark
†Drones and Autonomous Systems, University of Southern Denmark, Odense, Denmark
Email: *{albni20, akirk20, fjoer20, mifug13}@student.sdu.dk

***Abstract*—The field of robotics has become more complex, resulting in a growing need for computing power which necessitates scalable and cost-effective solutions. This project addresses the industrial challenge of scalable computing for advanced robotic systems, with a focus on distributed machine learning tasks. This paper presents an experimental prototype that employs distributed computing on Google Cloud Platform, utilizing Kubernetes and Kubeflow. This implementation prioritizes scalability and performance through different tactics and design patterns. The functionality of the prototype is demonstrated through a specific use case by training a machine learning model using the Fashion MNIST dataset which is used for inference on embedded hardware in the form of a Raspberry Pi 4 showing performance results of X, Y, Z. The platform infrastructure was successfully provisioned by using Infrastructure as Code within the required timeframe. Tests of the training platform revealed that it was possible to timely and automatically scale according to the needs of the computing workloads. However, the utilization of the requested resources of the pods running the machine learning training process was not as high as required.**

***Index Terms*—Scalable computing, Unmanned Aerial Vehicles, Distributed computing, Machine Learning, Computer vision**

I. Description of the industrial challenge

This project addresses the *Scalable Computing for Robotic Solutions* use case. One of the challenges in the use case is the advancement in making robotic solutions, such as drones, able to handle an increasing amount of computation. The robotics companies must be able to handle the increasing computational load optimally and cost-effectively.

A. Introduction and Motivation

Robotics is an industry that is growing rapidly with robots being embodied in multiple different markets and they need to be increasingly complex to solve advanced tasks [1]. These tasks range from autonomous navigation in drones to precision control in robotic surgery, and they all require an increasing computing workload. As robotic systems become more refined, the need for a scalable computing solution to manage these increasing workloads is growing. Traditional computing methods, such as sequential computing, are insufficient to handle the number and intensity of the workloads which are required by modern robotic applications [2]. Therefore, scalable computing and distributed systems have become essential to support the growing complexity of robotic systems. It allows for the efficient management of computational resources, enabling robots and drones to perform tasks with higher precision and speed while adapting to new challenges dynamically [1].

To assist the project work for this paper the objective is formulated into the following problem: *How can a software solution optimize robotic workloads within machine learning?* Through the utilization of scalability, it should be possible to minimize the computing cost for robotics companies. This project seeks to investigate the challenge of performing various machine learning tasks in drones by looking into how distributed computing can be automatically scaled and leveraged to train machine learning models optimally and cost-efficiently.

The paper is structured as follows: Section II presents related work in the form of state-of-the-art in the field. Section III describes the approach for the project, including the project use case and requirements for the system. Section IV describes the elements and design of the experimental prototype. The results of the design and implementation as well as the performed tests and experiments are presented in Section V, and lastly the project is concluded in Section VI which also presents future work.

B. Background

The term cloud computing was born after a new paradigm was introduced suggesting that not all workloads had to be deployed on a singular device. The computation could be split into different parts and thereby the work was distributed, resulting in a new concept called distributed computing [3]. The workloads are containerized and deployed in the cloud using technologies such as Kubernetes [4] or Nomad [5].

Unmanned Aerial Vehicles (UAV) commonly known as drones have gained an increasingly relevance for research and real world application with the development within microelectronics and computing efficiency [6]. In relation to UAVs, research is being conducted within areas such as drone navigation, obstacle detection, and edge computing. Applications and use cases for drones especially include monitoring, surveying, and inspection within areas such as environment, agriculture, and infrastructure along with delivery, and logistics.

This project will combine these key domains into an implementation of an experimental prototype which addresses the industrial challenge of Scalable Computing for Robotic Solutions.

II. State-of-the-art

In 2005, W. F. McColl stated that *scalable computing* would become the new traditional form of computing. From the beginning it was clear that sequential computing would be superseded by parallel computing [7]. Since then, scalable computing has developed rapidly and resulted in a paradigm shift, meaning that workloads are no longer deployed on singular device as a monolith, but they are split into multiple parts [7]. In the paper "An Edge Computing Sizing Tool for Robotic Workloads" [3], Ahmad Rzgar Hamid describes an increasing automation in robotics that requires new sensors, data storage, and artificial intelligence based data processing. All these new technologies cause a need for more computing workloads, and these workloads can be handled by using scalable computing. The paper does not propose a solution to solve a similar industrial challenge, but it presents an edge computing sizing tool for robotic workloads.

In the paper "Towards high performance robotic computing" [8], the authors Leonardo Camargo-Forero et al. introduce an approach for a multi-robot system. A high performance computing cluster is deployed and used for the scalable multi-robot system. This includes the novel concept of *High Performance Robotic Computing* which is based on the traditional High Performance Computing and used to fit and enhance the world of robotics. Zhi Liu et al. [9] propose a solution for scalable and cooperative computing in the area of aerial video streaming called *UAVideo*. They address the limits of UAV systems in terms of computational and communication resources and their influence on the overall computing performance of the system. The study concludes that the *UAVideo* system considerably reduces the operation time compared to other systems.

In the paper "DynamoML: Dynamic Resource Management Operators for Machine Learning Workloads" [10] a set of runtime dynamic management techniques which should help in managing the expensive GPU resources for the increasingly complicated and resource-intensive Machine Learning workloads is proposed. Their proposed solution, *DynamoML*, is claimed to be one of a few to integrate and apply modeling jobs, autoscaling and scheduling together specifically for a Machine Learning pipeline workflow and demonstrates the benefits and importance of this dynamic resource management.

An example of a practical contribution to the state-of-the-art is Ray [11]. Ray is a compute engine for AI workloads which can manage, execute, and optimize compute needs. It is an open-source unified framework that aids in running distributed machine learning workloads which ultimately reduces the complexity for the user [12]. Ray provides a distributed and scalable architecture for machine learning computations [13, 14] which is simplified through their libraries by "providing a seamless, unified, and open experience for scalable ML" [15].

These practical and academic contributions demonstrate the state-of-the-art within some different areas of scalable computing for robotics. This foundation enables further research within the field. This paper will contribute to the literature by designing, implementing and evaluating an automatically scalable, distributed system for training machine learning models with a focus on limiting the costs and resource usage in order to provide an efficient and profitable solution for robotic companies.

III. Approach

The approach of the project is to elicit requirements from the industrial challenge "Scalable Computing for Robotic Solutions" [16] and from these design and implement the experimental prototype which can then be tested and evaluated in proportion to the requirements.

This section presents some robotic workloads which are relevant to this project's industrial challenge [16]. The functionality of the software solution is demonstrated through two use cases. This section also presents the functional and non-functional requirements for the experimental prototype along with relevant quality attribute scenarios. Lastly, the solution for the project case is presented.

A. Robotic workloads

The field of logistics and supply chains is evolving drastically because of the recent developments within Artificial Intelligence (AI) and Machine Learning which are becoming increasingly important in the industry to improve processes and adaptability as well as increase efficiency and thereby gain competitive advantages. This is emphasized in the paper "A Review on The Use of Artificial Intelligence and Machine Learning Technologies in The Logistics Sector" where it is concluded that "Deep Learning has enhanced capabilities to analyze and predict complex data structures, while optimization techniques serve to reduce the costs associated with logistics operations and improve overall efficiency." [17].

Computer vision makes it possible to provide real-time insights and help in automating tasks, while image recognition can be used to identify and categorize specific objects, thereby minimizing the required amount of manual human labor. Computer vision and image recognition, in general, allow businesses to handle large volumes of data and tasks faster and more accurately, which in the end can lead to cost savings.

B. Use Case

Two use cases have been created for this project. Together, they demonstrate essential parts of the usage of the software solution, but more specifically, they demonstrate how an actor utilizes the solution. Generally, the intention of the solution is to be able to deploy different Machine Learning models to the system which makes it possible to perform different workloads and tasks. The solution should be adaptable and able to optimize specific machine learning workloads by training distributed and automatically scalable, which should result in a reduction of costs.

The first use case, seen in Table I, describes how the software solution is utilized, while the second use case, which is available in Table II, describes how the trained machine learning model is utilized.

TABLE I
USE CASE: UTILIZE THE SOFTWARE SOLUTION

Primary Actor	Software Engineer with knowledge within machine learning in a robotics company
Short description	The Software Engineer should be able to provide the solution with an machine learning model and a dataset to be able to train the model distributed and scalable.
Pre-condition	• The cluster and infrastructure of the solution is running
Main scenario	1) The Software Engineer provides the solution with an implementation to train an ML model 2) The solution trains the machine learning model using the dataset 3) The trained model is saved and stored

TABLE II
USE CASE: UTILIZE THE TRAINED MODEL

Primary Actor	Software Engineer with knowledge within machine learning in a robotics company
Short description	The Software Engineer should be able to perform inference using a trained machine learning model
Pre-condition	• The trained model is uploaded to an UAV • The UAV has a camera
Main scenario	1) The Software Engineer sends off the UAV 2) The UAV transmits the data which it collects through the camera to a monitoring system 3) The Software Engineer monitors the incoming inference information when necessary to be able to act on different events

TABLE III
FUNCTIONAL REQUIREMENTS

ID	**Description**
F1	The solution should be able to train machine learning models
F2	The solution should be able to distribute the training workload across multiple nodes
F3	The solution should be able to save and store the trained machine learning model
F4	It should be possible to monitor the training and performance of machine learning models
F5	The solution's underlying infrastructure should be created using Infrastructure as Code (IaC)

TABLE IV
NON-FUNCTIONAL REQUIREMENTS

ID	**Description**
NF1	The trained model should be able to run on embedded hardware with less than 1 GB of memory
NF2	The solution should be able to automatically scale up within 5 minutes when a new workload is scheduled
NF3	The solution should automatically downscale unused resources within 5 minutes of inactivity to minimize costs
NF4	The solution's infrastructure should be provisioned in less than 15 minutes
NF5	During training, CPU and memory utilization should maintain an average above 75% to ensure efficient resource usage

C. Requirements

Requirements have been elicited from the description of the industrial challenge [2] to help design and implement the experimental prototype. The requirements cover the functionality of the system and its quality attributes, as defined by Len Bass et al. in [18]. The requirements describe specifically how the prototype should work and by which measures the solution can be evaluated. This experimental prototype has a focus on minimizing costs, and therefore the quality attributes accommodate this to ensure proper evaluation of the solution. The functional requirements have been used during the implementation and are listed in Table III.

From the non-functional requirements listed in Table IV it is possible to define which quality attributes the system should be evaluated. The quality attributes are *Performance* which is connected to the requirements NF1, NF4, and NF5 and *Scalability* which is connected to the requirements NF2 and NF3. The non-functional requirements have been used to create guidelines for the architecture and the design of the solution. The design choices for the systems are presented in Section IV and the evaluation of the quality attributes can be seen in Section V.

To ensure the basic functionality of the machine learning system, a set of requirements is outlined in Table V. These requirements focus on enhancing the training process, maintaining stability, and ensuring model robustness. Specifically, requirements ML1 and ML4 aim to optimize the training process by improving efficiency and configurability. Requirement ML2 addresses system stability by providing checkpoints to safeguard progress in the event of interruptions. Lastly, requirement ML3 ensures robustness by incorporating preprocessing capabilities, such as rotation and value normalization, to prepare data effectively for training. NML1 ensures that the output model achieves an accuracy of at least 90% to meet performance expectations

D. Quality Attribute Scenarios

Based on the quality attributes, *Performance* and *Scalability*, two Quality Attribute Scenarios (QAS) have been made. QASes are used to specify the QA requirements and should help in determining architectural decisions for the software solution. The two QASes have been created to illustrate the flow of the scenarios from their source to their response measure, and they are illustrated down below. In Fig. 1, the QAS for Performance is illustrated. In order to make this QAS,

TABLE V
ML REQUIREMENTS

ID	Description
ML1	The solution should automatically stop training after XX epoch if the validation loss doesn't improve.
ML2	The solution should save the weights while training
ML3	The solution should be able to do prepossessing and data augmentation e.g. Rotate, nominalize values,
ML4	The solution should have an option selector eg batch size
NML1	The output model should have a accuracy on 90%

the General Scenario for Performance [18, pp. 202-203] has been used as a source of inspiration.

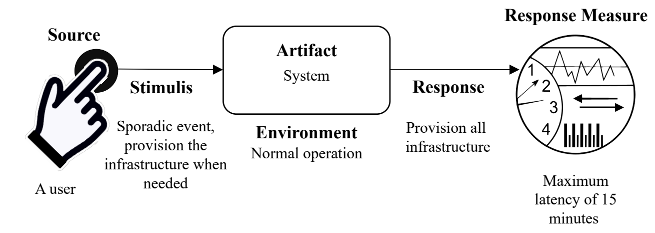

Source
A user
Stimulis
Sporadic event, provision the infrastructure when needed
Artifact
System
Environment
Normal operation
Response
Provision all infrastructure
Response Measure
Maximum latency of 15 minutes

Fig. 1. Quality attribute scenario for performance

In Fig. 2 the QAS for Scalability is illustrated and in contrast to the previous scenario, this scenario has not taken inspiration from a template in the book ”Software Architecture in Practice” by Len Bass et al. [18], since it does not provide one for scalability.

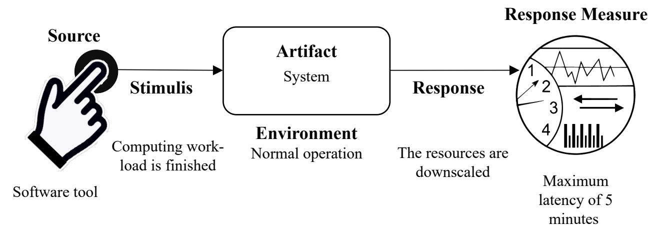


Fig. 2. Quality attribute scenario for scalability

The environment is in both cases set to ”Normal operation” which means there are no component or system failures, anomalies, or interruptions for the artifact. The response measure is used for evaluating the degree to which the software solution achieves the intended QA with its architecture.

E. Solution

The solution for this project includes a software solution that was developed to perform distributed and automatically scalable computing. It was designed based on the knowledge presented in SectionII, the industrial challenge, and the requirements. Improvements for the presented software solution are described in Section VI. The solution is different to other solutions presented in state-of-the-art in the way that it is specifically concerned with the machine learning task image recognition. The presented state-of-the-art is generally concerned with scalability but does not have specific emphasis on being *automatically* scalable contrary to the solution for this project.

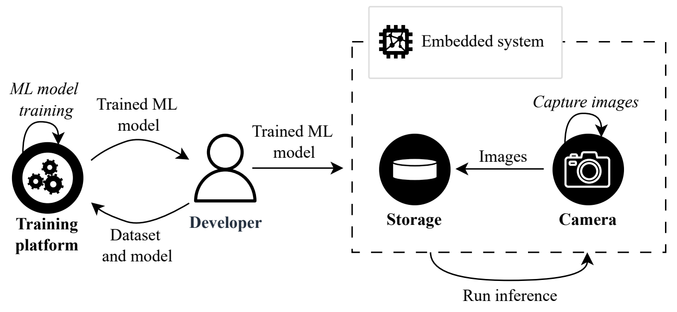


Fig. 3. System sequence diagram

Fig. 3 illustrates how a developer can use the experimental prototype. The developer uploads a dataset and a machine learning model to the platform, where the model is then trained. After training, the developer uploads the trained model to the embedded system where images are captured, inferred, and stored. Images can be transferred in bulk from the embedded system to the training platform for further training of the model.

IV. EXPERIMENTAL PROTOTYPE

This section contains a description of the experimental prototype, including what requirements the platform fulfills. The used machine learning model and the embedded system are also described.

A. The machine learning model training platform

The experimental prototype for this project is a training platform designed and built to scale the training process of machine learning models with an emphasis on cost efficiency. Moreover, the experimental prototype includes a trained machine learning model which is deployed to a Raspberry Pi 4 to demonstrate the model's capabilities when running on hardware that is realistic to this specific use case. The training platform uses a distributed strategy to allow multiple workers to train a machine learning model on Kubernetes using Kubeflow and Pytorch in Google Cloud.

The training platform uses Google Cloud as cloud provider because it provides functionality that supports both the functional and non-functional requirements of this project. The main quality attributes of this project are *Performance* and *Scalability*. The requirements regarding these attributes have been fulfilled through the selected technologies and design choices since they utilize different tactics and design patterns.

In Google Cloud, an autopilot Google Kubernetes Engine (GKE) has been provisioned. An autopilot GKE is a managed Kubernetes cluster that implements autoscaling. This fulfills the requirements NF2 and NF3 concerning scalability, through the use of tactics like horizontal scaling and containerization. The requirements NF1, NF4, and NF5 regarding performance have also been fulfilled by the use of the autopilot GKE. It allows for managing resources through tactics such as ”Increase Resources” and ”Schedule Resources” as described by Len Bass et al. [18, pp. 215-218].

The implemented architecture of the experimental prototype is illustrated in Fig. 4. The architectural diagram shows all the different resources and technologies which have been provisioned by the use of the Infrastructure as Code (IaC) tool Terraform in order to create the training platform. To be able to perform distributed training, the Kubeflow training operator was used because it simplified the distribution process. These resources along with the ones depicted in Fig. 4 fulfill the remaining functional requirements, F1, F2, F3, F4, and F5.

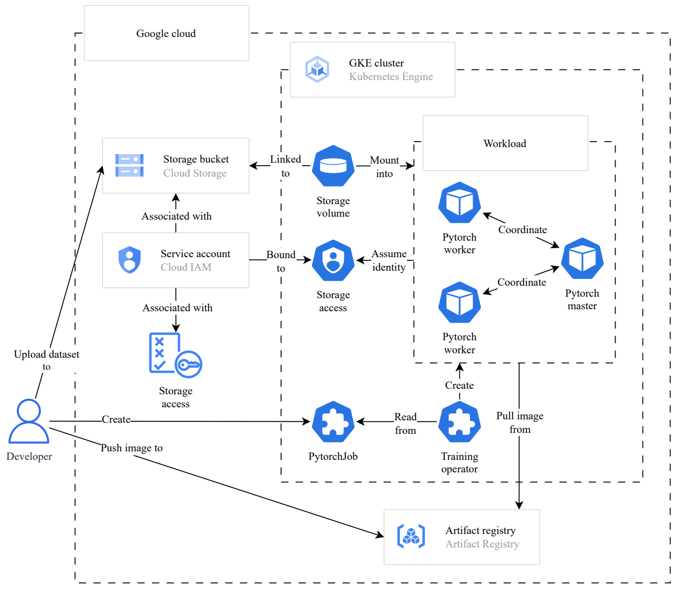


Fig. 4. Architecture diagram

Fig. 5 illustrates the workflow of the training platform at the time it is being used by the developer as described in the use case in Table I.

In the workflow a dataset is ingested to the storage bucket, an image is pushed to the container registry and a workload is deployed to the Kubernetes cluster. The cluster scales up the nodes to accommodate the workload and when there is sufficient capacity the workload is deployed to the cluster. The pods in the workload are created using the image stored in the artifact registry and use the permission bound to the service account to mount the storage into the pods. The pods can then use the dataset to train the machine learning model using a distributed approach. When the model has finished training, it is stored in the storage bucket along with performance data which can be used to evaluate the model. As the workload has finished, the cluster scales down the number of nodes to avoid wasted capacity. This describes how the different

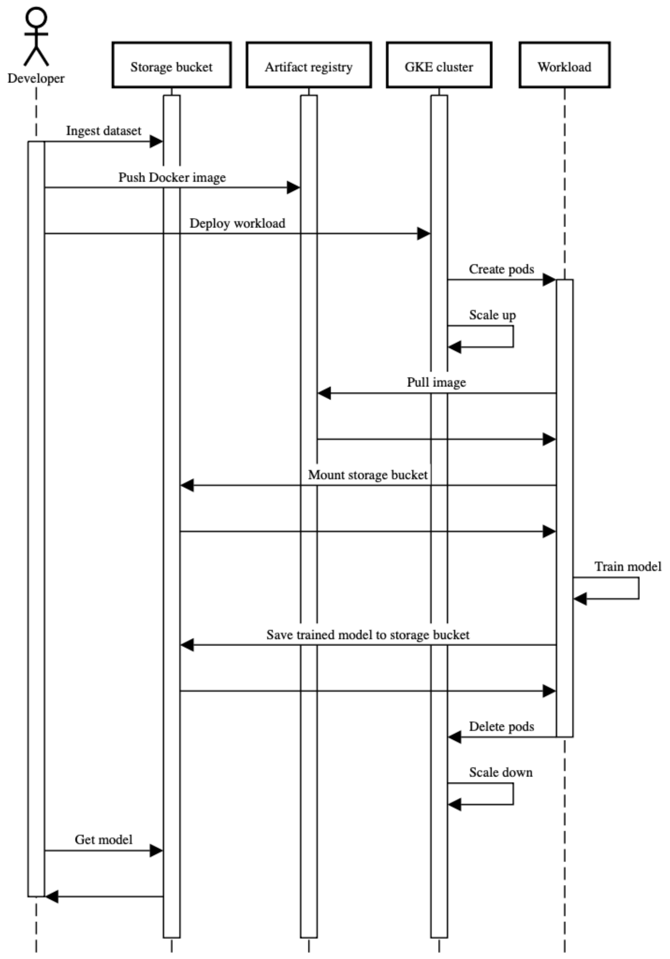


Fig. 5. Workflow sequence diagram

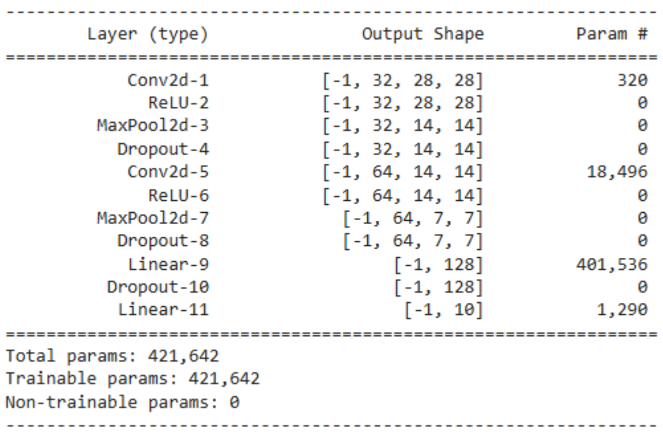
components have been realised in the implementation of the software solution.

B. The machine learning model

The machine learning model used in this prototype is designed to perform image classification tasks using deep learning techniques, by using Convolutional Neural Network (CNN). The model is trained on the Fashion MNIST dataset, which consists of 28x28 labelled, gray-scale fashion images of clothing items [19].

A simple CNN architecture has been designed and used to demonstrate the functionality of the training platform. The design balances complexity and performance, which makes it qualified for deployment and training on the cloud-based infrastructure and the embedded systems for inference [20, 21]. No requirements were made for the CNN architecture, since it should only act as a demonstration of the training platform.

The layers and total number of parameters of the implemented CNN model architecture is illustrated in Fig. 6. The output of the model is a 10-component vector corresponding to the 10 different classes of the Fashion MNIST dataset.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 28, 28]             320
              ReLU-2           [-1, 32, 28, 28]               0
         MaxPool2d-3           [-1, 32, 14, 14]               0
           Dropout-4           [-1, 32, 14, 14]               0
            Conv2d-5           [-1, 64, 14, 14]          18,496
              ReLU-6           [-1, 64, 14, 14]               0
         MaxPool2d-7             [-1, 64, 7, 7]               0
           Dropout-8             [-1, 64, 7, 7]               0
            Linear-9                  [-1, 128]         401,536
          Dropout-10                  [-1, 128]               0
           Linear-11                   [-1, 10]           1,290
================================================================
Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0
----------------------------------------------------------------
```

Fig. 6. CNN architecture

In order to make use of the distributed training provided by the training platform, it is necessary to implement the package `torch.distributed` from PyTorch [22] in the model architecture.

The following steps were included in the training of the machine learning model:

- **Pre-processing**: The Fashion MNIST dataset is pre-processed to normalize the images so the pixel value is between -1 and 1. The dataset is split into training and validation subsets.(ML3)
- **Data Augmentation**: To further improve the generalization of the model and reduce overfitting, basic data augmentation techniques such as random rotation was applied during training. (ML3)
- **Loss function**: The model uses cross-entropy loss, a standard loss function for multi-class classification task, to optimize the network during training
- **Optimizer**: The Adam optimizer was chosen for training, because of its adaptive learning rate.

C. *The embedded system*

In this project, the embedded system is built around a Raspberry Pi 4 with a connected camera. This is built to demonstrate the use of the machine learning model after training for the inference tasks, which could be performed by the end user. The embedded system could both be a drone or a stationary robot, solving inference tasks for companies within logistics and supply chain management.

The Raspberry Pi 4 is used in the experimental prototype to demonstrate the use case from Table II. It gives a concrete example of a low-cost, edge-computing device which can handle inference with machine learning models.

V. Results

Currently, the training platform only uses CPUs for training due to budget constraints, but GPU resources could be requested and used by specifying a nodeselector on the workloads.

The experiment which was carried out to validate the non-functional requirements listed in table IV involved deploying the training platform, training an ML model as described in Fig. 5, and testing the trained model on a Raspberry Pi.

The training platform was deployed to Google Cloud using Terraform. The provisioning process took 10 minutes and 14 seconds. This part of the experiment confirms that the non-functional requirement NF4 is fulfilled.

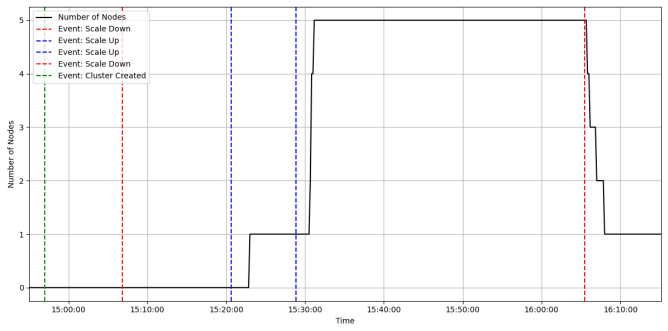


Fig. 7. Number of nodes in cluster

After deploying the training platform, the cluster scales down to zero nodes as no workload has been deployed. That is depicted in Fig. 7 where at 14:57 the cluster is created and scales down to zero nodes at 15:06. The Kubeflow training operator is deployed to the cluster which triggers a scale-up at 15:20. After 3 minutes, the cluster is scaled to one node, and the Kubeflow training operator is running. The machine learning workflow is then deployed which triggers a scale-up at 15:29. The cluster scales to five nodes within 2 minutes. This confirms that the solution fulfills requirement NF2.

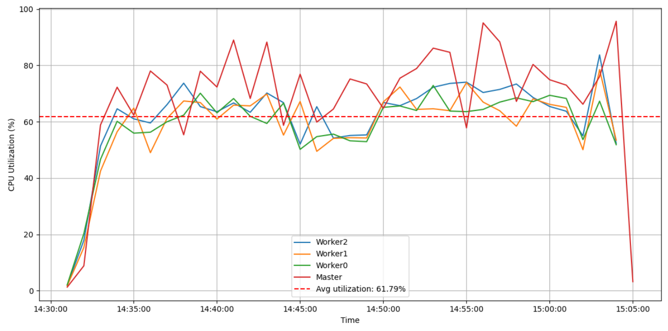


Fig. 8. Pod CPU request utilization

The workload runs from 15:31 to 16:05. The workload runs the distributed training process of the machine learning model using one master and three workers. During the training of the machine learning model, the workers, and master consume on average 61.79% of the requested CPU resources as visualized in Fig. 8. Simultaneously, the memory usage of the workers is around 30% and the memory usage of the master is around 45% with the overall average being 33.83%. Neither the CPU nor memory utilization percentage is as high as NF5 requires, and the requirement is therefore not fulfilled. Better resource

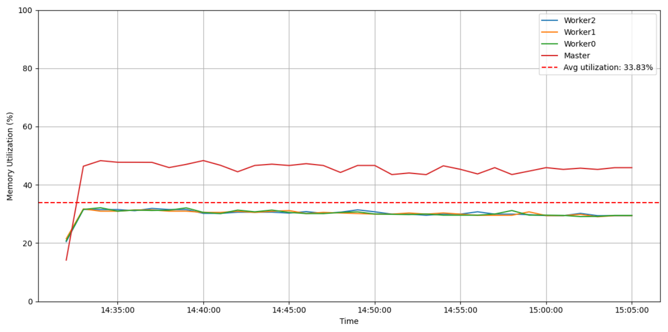


Fig. 9. Pod memory request utilization

utilization could likely be achieved by adjusting the resource requests.

After the workload has terminated, the cluster triggers a downscale. The downscale event is triggered at 16:05 and the cluster is scaled down to 1 node at 16:08. The downscale occurs within 3 minutes of the workload terminating, therefore requirement NF3 is fulfilled.

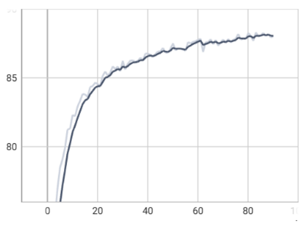


Fig. 10. Training Accuracy

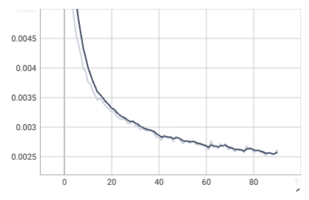


Fig. 11. Training Loss

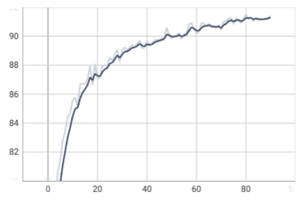


Fig. 12. Validation Accuracy

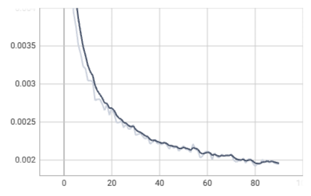


Fig. 13. Validation Loss

The graphs in Fig. 10, 11, 12 and 13, provide insights into the training and validation process of the machine learning model:

Training accuracy (Fig. 10): The training accuracy steadily increase over the epochs, Starting at 58,79% and reaching 88,05% at the end of training. Indicating that the model is progressively learning to classify the training data correctly.

Training loss (Fig. 11): The training average loss decreases progressively from 0.0089 to 0.0026 throughout the epochs, demonstrating that the model is minimizing the error on the training data as expected.

Validation accuracy (Fig. 12): The validation accuracy improves alongside training accuracy, starting at 73.13% and reaching 91,4%, reflecting the model's ability to generalize to unseen data. Any significant gap between validation and training accuracy would indicate potential over fitting, where the model performs well on the training data but struggles with new data.

Validation loss (Fig. 13): The validation average loss decreases initially from 0.0056 to 0.0019, indicating that the model is learning patterns that generalize well. A subsequent plateau or increase in validation loss could signal overfitting, where the model starts to memorize the training data rather than learning generalizable patterns. Alternatively, a slight increase may also suggest the model has reached an optimal balance point, where further training does not yield significant improvement in generalization.

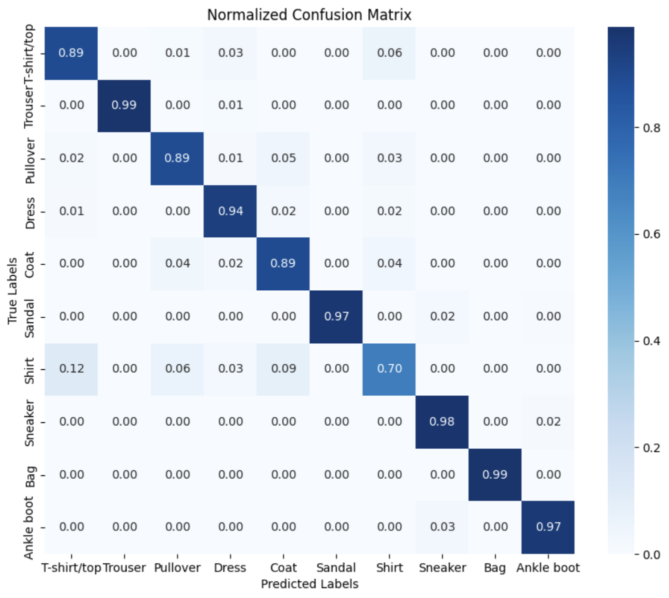


Fig. 14. Confusion matrix of the test dataset

The normalized confusion matrix (fig. 14) show the proportion of predictions for each class relative to the total true instances of that class. Key observation from the matrix are as follows:

Correct predictions (Diagonal values): The highest proportion of correct predictions are observed for the classes *Trouser* (0.99), *bag* (0.99), *sneaker* (0.98) and *sandal* (0.97). The lowest proportion of correct predictions is for the class *shirt* (0.70).

Misclassification (Off-Diagonal values): The *shirt* class has notable proportions of misclassification into *T-shirt/top* (0.12), *pullover* (0.06) and *coat* (0.09).

VI. Conclusion

The purpose of this project was to optimize robotic workloads within machine learning. This has been done by careful design and development of a cloud-based platform, which can be used for training machine learning models.

The results of the experiment demonstrated that the platform fulfills the requirements for rapid provisioning and automated scaling. The resource utilization was insufficient to fulfill the requirement for effective CPU and memory utilization.

The project included a demonstration of how to use the training platform where a model was implemented and trained using the Fashion MNIST dataset which provided THESE inference results.

References

[1] António J. R. Neves et al. “A Personal Robot as an Improvement to the Customers’ In-store Experience”. In: vol. 921. Communications in Computer and Information Science. Springer International Publishing, 2019, pp. 296–317. ISBN: 1865-0929.

[2] FAAM. *Industrial Challenges*. URL: https://github.com/AGejr/scalable-computing/blob/main/Industrial%20challenges.pdf. (accessed: 11.12.2024).

[3] Ahmad Rzgar Hamid and Mikkel Baun Kjærgaard. “An Edge Computing Sizing Tool for Robotic Workloads”. In: *2024 IEEE/ACM 6th International Workshop on Robotics Software Engineering (RoSE)*. 2024, pp. 43–46.

[4] Kubernetes. *Kubernetes Documentation - Concepts*. URL: https://kubernetes.io/docs/concepts/. (accessed: 10.10.2024).

[5] HashiCorp. *Nomad Documentation*. URL: https://developer.hashicorp.com/nomad/docs?product_intent=nomad. (accessed: 10.10.2024).

[6] Abdulla Al-Kaff anf David Martín et al. “Survey of computer vision algorithms and applications for unmanned aerial vehicles”. In: *Expert Systems with Applications* 92 (2018), pp. 447–463.

[7] W. F. McColl. “Scalable computing”. In: *Computer Science Today: Recent Trends and Developments*. Ed. by Jan van Leeuwen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 46–61. ISBN: 978-3-540-49435-5. DOI: 10.1007/BFb0015236. URL: https://doi.org/10.1007/BFb0015236.

[8] Leonardo Camargo-Forero, Pablo Royo, and Xavier Prats. “Towards high performance robotic computing”. In: *Robotics and Autonomous Systems* 107 (2018), pp. 167–181. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2018.05.011. URL: https://www.sciencedirect.com/science/article/pii/S092188901830232X.

[9] Zhi Liu et al. “Robust Edge Computing in UAV Systems via Scalable Computing and Cooperative Computing”. In: *IEEE Wireless Communications* 28.5 (2021), pp. 36–42.

[10] Min-Chi Chiang et al. “DynamoML: Dynamic Resource Management for Machine Learning Pipeline Workloads”. In: *SN Computer Science* 4.5 (Sept. 2023), p. 665.

[11] INC ANYSCALE. *Ray is the AI Compute Engine*. URL: https://www.ray.io. (accessed: 14.11.2024).

[12] The Ray Team. *Overview*. URL: https://docs.ray.io/en/latest/ray-overview/index.html. (accessed: 14.11.2024).

[13] Ray Team. *Ray v2 Architecture*. URL: https://docs.google.com/document/d/1tBw9A4j62ruI5omIJbMxly-la5w4q_TjyJgJL_jN2fI/preview?tab=t.0. (accessed: 14.11.2024).

[14] Ray. *ray-project/ray*. URL: https://github.com/ray-project/ray?tab=readme-ov-file. (accessed: 14.11.2024).

[15] The Ray Team. *Ray for ML Infrastructure*. URL: https://docs.ray.io/en/releases-2.9.3/ray-air/getting-started.html. (accessed: 14.11.2024).

[16] SDU. *Industrial Challenges*. URL: https://sdu.itslearning.com/ContentArea/ContentArea.aspx?LocationID=35207&LocationType=1. (accessed: 07.11.2024).

[17] S. Oğuz and D. Yalçıntaş. “A Review on The Use of Artificial Intelligence and Machine Learning Technologies in The Logistics Sector”. In: *Trends in Business and Economics* 38.4 (2024), pp. 218–225.

[18] Len Bass, Paul Clements, and Rick Kazman. “3. Understanding Quality Attributes”. In: *Software architecture in practice*. 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2021, pp. 72, 202–203, 215–218. ISBN: 9780136886099;

[19] PyTorch. *FashionMNIST*. URL: https://pytorch.org/vision/stable/generated/torchvision.datasets.FashionMNIST.html. (accessed: 05.12.2024).

[20] Nico Klingler. *Convolutional Neural Networks (CNNs): A 2025 Deep Dive*. URL: https://viso.ai/deep-learning/convolutional-neural-networks/. (accessed: 11.12.2024).

[21] Ousmane Youme et al. “Evolution under Length Constraints for CNN Architecture design”. In: New York, NY, USA: ACM, 2023, pp. 23–31.

[22] PyTorch. *Distributed communication package - torch.distributed*. URL: https://pytorch.org/docs/stable/distributed.html. (accessed: 05.12.2024).

Individual contribution

Lecture report

All group members worked equally on the tasks from each lecture.

Technical report

Albert:

- Requirements
- Experimental prototype
- Results
- Conclusion

Anne Cathrine:

- Description of the industrial challenge
- State of the art
- Approach
- Conclusion

Frederikke:

- Description of the industrial challenge
- State of the art
- Approach

- Conclusion

Michael:

- Requirements
- Experimental prototype
- Results
- Conclusion

Prototype:

Albert:

- Training platform

Anne Cathrine:

- Training platform

Frederikke:

- Training platform

Michael:

- Machine learning
- Embedded system