DECENTER

Decentralised technologies
for orchestrated Cloud-to-Edge intelligence

D4.3

Second release of application's Artificial Intelligence methods and solutions

31/07/2020

Revision history

Administration and summary		
Project acronym:	DECENTER	
Document identifier:	D4.3: Second release of application's Artificial Intelligence methods and solutions [M24]	
Leading partner:	KETI	
Report version:	1.0	
Report preparation date:	31.07.2019	
Classification:	PU (Public)	
Nature:	R (Report)	
Author(s) and contributors:	Seungwoo Kum, Jaewon Moon (KETI) Seokhee Lee (SNU) Orlando Avila Garcia (ATOS) Vlado Stankovski, Uroš Paščinski (UL) Guillem Garí (ROB) Marco Piazzola (FBK) Sofia Kleisarchaki (KENT)	
Status:	-	Plan
	-	Draft
	x	Final

The DECENTER Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

Date	**Edited by**	**Status**	**Changes made**
19.05.2020	Seungwoo Kum	Plan	ToC
19.05.2020	Seungwoo Kum	Draft	Added Chapter 3 and 6
22.05.2020	Seokhee Lee	Draft	Added Chap. 2
29.05.2020	Orlando Avila Garcia, Jaewon Moon, Valdo Stankovski, Guillem Garí, Marco Piazzola	Draft	Added Chap. 5
05.06.2020	Seokhee Lee	Draft	Edited Chap. 2
15.06.2020	Seungwoo Kum	Draft	Edited contents and formats, added Chap 1 and executive summary.
22.06.2020	Sofia Kleisarchaki	Draft	Added chapter 2

26.06.2020	Sofia Kleisarchaki, Uroš Paščinski, Gullem Gari, Marco Piazzola	Draft	Added subsections of section 4.2
26.06.2020	Seungwoo Kum, Vlado Stankovski	Draft	Added chapter 2 (new chapter)
07.07.2020	Orlando Avila Garcia	Draft	Editorial comments
17.07.2020	Uroš Paščinski	Draft	Updated Section 6.3.3.
27.07.2020	Orlando Avila Garcia, Petar Kochovski	Draft	Updated Section 6.3.
30.07.2020	Seungwoo Kum	Final	Final version

Notice that other documents may supersede this document. A list of latest *public* DECENTER deliverables can be found at the DECENTER Web page at https://www.decenter-project.eu/.

Copyright

Acknowledgements

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 815141 (DECENTER: Decentralised technologies for orchestrated Cloud-to-Edge intelligence)

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT/IITP) (No. 1711075689, Decentralised cloud technologies for edge/IoT integration in support of AI applications).

The partners in the project are FONDAZIONE BRUNO KESSLER (FBK), ATOS (ATOS), COMMISSARIAT À L'ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES (CEA), COMUNE DI TRENTO (TN), ROBOTNIK (ROB), UNIVERZA V LJUBLJANI (UL), KOREA ELECTRONICS TECHNOLOGY INSTITUTE (KETI), GLUESYS (GLSYS), DALIWORKS (DW), LG U+ (LGUP), SEOUL NATIONAL UNIVERSITY (SNU).

The content of this document is the result of extensive discussions within the DECENTER © Consortium as a whole.

More information

Public DECENTER reports and other information pertaining to the project are available through DECENTER public Web site under http://www.decenter-project.eu.

Contents

Revision history 1
Copyright 2
Acknowledgements 3
More information 3
Executive Summary 8
1 Introduction 9
2 Challenges and prospects for AI on the edge 10
2.1 Differences between edge resource and cloud resources 10
2.2 Delivering AI as a service 11
2.3 Security and privacy 11
2.4 Distributed data generation 12
3 Methods for AI delivery from cloud to edge 14
3.1 Survey on Deep Neural Network Compression and Acceleration 14
3.2 Channel pruning on face detector 18
3.3 Semi-supervised learning on the edge 20
3.4 Deep adversarial training to keep privacy 23
3.5 Remarks 24
4 Updates on Containerization of AI methods 25
4.1 AI as a Microservice - Deployment and Management 25
4.2 New features of the DECENTER AI package 26
4.3 Design Pattern of AI Microservice with DECENTER AI Package 28
4.4 Implementation of AI service with DECENTER AI package: UC4 33
4.5 Remarks 34
5 Updates on Digital Twin representation 35
5.1 Interaction between DT and AI at microservice level 35
5.2 Data types of the DT representations 36
5.2.1 UC1: Smart City Crossing Safety 36
5.2.2 UC2: Logistics Robotics 37
5.2.3 UC3: Smart and Safe Construction App 39
5.2.4 UC4: Ambient Intelligence for Office Environments 41
5.3 Updates of SensiNact to support DT 42
5.4 Remarks 44
6 Updates on AI solution design 45
6.1 Machine Learning Workflow 45
6.2 AI Solution Design 49
6.2.1 AI Application reference architecture 49

6.2.2 Level of containerization 51
6.2.3 Interaction with Microservices 53
6.3 Use case AI application description 53
6.3.1 Smart City Crossing Safety (UC1) 53
Crossing Safety - Application Services 54
Crossing Safety - Architecture Views 55
6.3.2 Logistics Robotics Optimization (UC2) 56
Logistics Optimization - Application Services 56
Logistics Optimization - Architecture Views 57
6.3.3 Smart and Safe Construction (UC3) 59
Smart and Safe Construction - Application Services 59
Smart and Safe Construction - Architecture Views 60
6.3.4 Ambient Intelligence for Office Environments (UC4) 61
Ambient Intelligence - Application Services 61
Ambient Intelligence - Architecture Views 62
6.4 Interaction with Platform 63
6.4.1 Adding resources to the infrastructure 63
6.4.2 Request resources for the deployment 64
7 Federated Learning with DECENTER 65
7.1 Introduction to Federated Learning 65
7.2 AI Service and Federated Learning 65
7.3 Design of Federated Learning Architecture with DECENTER 66
7.3.1 Model Management for Inferencing and Learning 66
7.3.2 Resource orchestration 67
7.3.3 Architecture 68
7.3.4 Data Flow 68
7.4 Evaluation on MNIST data set 71
7.5 Remarks 74
8 Conclusion and future works 75
9 References 78
10 Abbreviations 80

List of Figures

Figure 1 Goal of pruning is to produce a Y' that resembles Y as much as possible.....16
Figure 2 Three step pipeline of pruning and comparison of neurons before and after pruning [5].....16
Figure 3 Overview of channel pruning [4].....16
Figure 4 Overview of standard convolution and decomposed convolution methods.....17
Figure 5 Architecture of DSFD [3].....18
Figure 6 Detection performance of DSFD [3].....18
Figure 7 A conceptual illustration of semi-supervised learning.....20
Figure 8 An illustration of semi-supervised learning scheme.....21
Figure 9 An example of semi-supervised learning process.....21
Figure 10 t-SNE visualization of GPQ.....22
Figure 11 t- The graph of semi-supervised learning results (mAP score), by varying the amount of unlabelled data; blue line for CIFAR-10 and orange line for NUS-WIDE.....22
Figure 12 Adversarial training for image retrieval.....23
Figure 13 The process of adversarial example generation.....23
Figure 14 Software stack for AI application and virtualization.....25
Figure 15 First version of the DECENTER package, from Y1. The AI method or logic is implemented in a child class of decenter.ai.baseclass and only HTTP protocol is supported.....27
Figure 16 New version of the DECENTER AI package, from Y2. The AI method execution, which is supported within the BaseClass, is now separated from the Interface modules, which now support HTTP and MQTT protocols.....27
Figure 17 Configuration of client-server design pattern.....30
Figure 18 Configuration of event-driven design pattern.....31
Figure 19 Microservice configuration of UC4, including Digital Twin and Data Management.....33
Figure 20 Components of Digital Twin.....35
Figure 21 Use of DECENTER Digital Twin application service by UC1 AI application.....36
Figure 22 Use of DECENTER Digital Twin application service by UC2 AI application.....37
Figure 23: Use of DECENTER Digital Twin application service by UC3 AI application. Services related to digital twin are shaded with grey background.....39
Figure 24 Use of DECENTER Digital Twin application service by UC4 AI application.....41
Figure 25 Digital Twin Data Model, UC4.....43
Figure 26 Continuous workflow of activities in the MLOps systems development life cycle. The activities of the ML training phase (in dark blue) are followed by the activities of the ML operations phase (light blue).....45
Figure 27 Technology stack of KFServing, the supported multi-framework model serving system of Kubeflow (extracted from KFServing GitHub Repository).....47
Figure 28 Adaptive behaviour control loop of an intelligence (adaptive) system; is has been taken as the reference AI process model in DECENTER.....50
Figure 29 Comparison of containerization of an AI microservice software stack. Containers including all the software stack have benefits on effective virtualization (e.g. environment isolation), with the cons of a larger size.....52
Figure 30 Example deployment of AI microservices with DECENTER.....53
Figure 31 Process view of UC1 AI Application.....55
Figure 32 The Application Components view shows the control flow between microservices (enclosed in each Application Service), in UC1.....55
Figure 33 Example of deployment of UC1 containerised microservices on Kubernetes-based Fog Platform.....56

Figure 34 Process view of UC2 AI Application 57
Figure 35 The Application Components view shows the control flow between microservices (enclosed in each Application Service), in UC2 58
Figure 36 Example of deployment of UC2 containerised microservices on Kubernetes-based Fog Platform 58
Figure 37 Process and data flow of UC3 60
Figure 38 Process view of AI microservices in UC3 60
Figure 39 Application components view shows the control flow between microservices in UC3. The consecutive steps of the workflow are described in D2.2 60
Figure 40 Process view of UC4 AI Application 62
Figure 41 The Application Components view shows the control flow between microservices (enclosed in each Application Service), in UC4 62
Figure 42 Example of deployment of UC4 containerised microservices on Kubernetes-based Fog Platform 63
Figure 43 AI-container building process with DECENTER AI package 64
Figure 44 FL service configuration with DECENTER on cloud-edge environment.....68
Figure 45 Algorithm of FL Server.....70
Figure 46 Algorithm of FL Client 70
Figure 47 Process between instances of FL process 71
Figure 48 Microservice deployment configuration on Kubernetes cluster 72

Executive Summary

This document reports activities carried out by WP4 until M24. These activities focus on delivering AI intelligence to the edge and investigating various aspects of AI on the edge to provide suitable methods and facilities to accomplish it. Each task represents one of those aspects: T4.1 on delivering AI onto resource-constrained devices; T4.2 on making available representations produced by and fed into AI methods; T4.3 on data management; and T4.4 on best practices for AI solution design to deploy AI methods along the cloud-to-edge continuum. This document describes updates on each activity accomplished until now, with the exception of T4.3, whose last report was given within D4.2. The outcomes of WP4 are very important for the DECENTER platform. Firstly, to make it able to run AI methods in resource-constrained devices (typically located at the edge), a novel network pruning method was investigated in T4.1. The method shows efficient resource usage on the edge compute node with smaller models, less computational loads and faster execution. The method is implemented on one of models of use cases (UC4). Second, in T4.2, the Digital Twin service was improved to provide placeholder for the features and data in AI application processes. Data types and appropriate protocol are designed in the activity. Third, T4.4 i) investigated AI solution good practices to help implement end-to-end AI application processes; ii) updated the DECENTER AI Package, which is a python library to develop AI microservices, to fulfil the requirements with a few design patterns; iii) updated the DECENTER use case solution designs. Last but not least, an AI application which uses Federated Learning was implemented to make an initial validation of the DECENTER Platform to support distributed AI model training processes.

This document describes those outcomes in detail as well as the benefits coming from them.

- Network pruning method for AI model. It ensures less computational resource usage with faster execution time.
- Digital Twin module. It provides placeholder for data and features in AI data flow to be used in various applications such as Digital Twin.
- DECENTER AI package. It provides methods to build an AI microservice from an AI method in a systematic way.
- AI solution and design guideline. With the multiple use case implementations in DECENTER, it can provide insights on AI application design for edge-cloud environment.

1 Introduction

Deliverable D4.3 summarizes the work done within WP4 until month 24 of the project. The main focus of WP4 is to deliver intelligence onto edge on top of DECENTER Platform, and several methods and facilities to provide running environment of AI on the edge has been designed and implemented on the second project year. All the tasks have been started on the first year and this document provides update of AI methods and solutions provided by DECENTER.

As stated in the previous report (D4.1), AI applications often require a high amount of computational power, relying sometimes on some specific hardware, which result in varying requirements and constraints for cloud infrastructure as well as application architectures. WP4 investigated various aspect of AI application from AI model to cloud-edge deployment to provide a suitable environment for AI to run with edge. T4.1 focuses on AI model optimization, to deliver AI onto resource-constrained devices. In Y2 network pruning method has been investigated, along with methods to help privacy preserving for AI on the edge and cloud. T4.2 focuses on making available representations produced by and fed into AI methods, and SensiNact has been updated to provides methods for accessing extracted features from the AI on other applications such as Digital Twin. T4.3 is focussing on cross-border data management, and now being implemented according to the architecture which has been described in D4.1, while the details of cross-border data management are going to described in next deliverable (D4.4) in M30. The best practices on AI solution to help implement use case pilots has been investigated in T4.4.

The rest of this report is organised as follows: the environment on cloud-edge computing and AI has been updated in Chapter 2. Chapter 3 describes on AI optimization methods newly investigated on the second year. The network pruning method and its implementation on AI model of UC4 is presented in that chapter. Chapter 4 describes updates of DECENTER AI package. DECENTER AI package refers a facility to help containerisation of AI methods as a microservice and has been updated to apply more flexibility on service configuration. A few design patterns to build an AI microservice are provided for reference. Chapter 5 describes updates on Digital Twin, to make use of features and data on AI data flow for Digital Twin implementation. The AI design and implementation good practices are given in Chapter 6, including those of AI/ML Workflow. Chapter 7 describes the implementation of Federated Learning, a decentralized ML model building technique, on top of the DECENTER platform and the DECENTER AI facilities. Conclusions on the second years activities and plan for the third year are presented in Chapter 8.

2 Challenges and prospects for AI on the edge

In the last years, applying Deep Learning (DL) technology at the edge of the network has attracted much attention. The main research and engineering challenges are 1) to build lightweight AI service and 2) to enforce collaboration and use of infrastructure along the compute (cloud-to-edge) continuum. Regarding lightweight AI, the focus has been on hardware acceleration for DL workloads and making AI software stacks and runtime platforms lighter so to decrease the amount of required computing resources and energy (power) consumption on smaller (e.g. edge) devices. In this respect vendors are making good progress: Nvidia provides TensorRT as inferencing system on the Tegra-based SoC devices. For Google, there are TensorFlow Lite and Tensor Processing Unit devices. Rockchip from china has their own platform and acceleration enabled SoC. However, comparing to the progress inbuilding lighter AI, the collaboration between cloud and edge infrastructure to deploy workloads along the compute continuum, is being developed at a slower pace. Instead of running an AI on a stand-alone device, it needs focusing on how to collaborate the resources on the edge to those on the cloud to provide suitable solutions to bring a practical end-to-end AI service with optimized resource usage and resolving privacy issues.

In this section, the challenges and prospects for AI on the edge to be addressed in Y2 of DECENTER are presented.

2.1 Differences between edge resource and cloud resources

Many cloud platforms—including those from big technology companies such as Google, Amazon and Microsoft—are supporting edge computing with the help of cloud technologies. In their proposed solutions, the AI method is packaged with container technology, delivered and deployed on the edge devices with the help of resource orchestration tools. However, using the edge resource has quite different characteristics than using the cloud resources. According to the definition of Cloud Computing[1], the resources on the cloud are considered to be infinite and continuously accessible, which is not quite applicable to the edge computing devices. The main idea behind those devices is to place computing resources nearer the data source or the service endpoint, to offload computational loads from the cloud and provide better Quality of Service (QoS) in terms of security, response time, bandwidth or latency. The downside is that those resources cannot be considered as infinite nor homogeneous as cloud resources, they need to be placed in a specific location, they can have different computing architectures, and/or the wide area network (WAN) access (including Internet) can be limited due to the infrastructure configuration.

- Managing edge resources in DECENTER

DECENTER has been working on identification of edge resources on the second year in WP3 and WP4. Edge resources have quite different characteristics when compared to those of cloud. First, the edge resource will hardly be placed on the cloud infrastructure, and it needs description of locations or regions to make use of that. Second, the runtime environment would not be the same as the one in the cloud infrastructure. The edge resource will rely on various hardware architecture with different interfaces, for example ARM-based SoC integrated with GPU module.

An extensible description scheme for edge resources for hardware acceleration was defined to describe them with respect to the two characteristics identified above. The location information is described as ‘region’ to show where the resource is located and on which infrastructure (cloud or edge). Also, the type of acceleration resources are described with a

[1] NIST publication 800-145, The NIST Definition of Cloud Computing.

few directives, to show which acceleration device is there and how many memories are available for the device. The details of those descriptions can be found on D3.3. Those resource definitions can cover two dimensions of the heterogeneity of such resources: heterogeneity of cloud-edge resource allocation (geometric location) and heterogeneity of different architectures which each resource is based on.

2.2 Delivering AI as a service

It is essential to turn AI methods into services to make use of cloud technologies for cloud-edge collaboration by deploying them on proper resources; this means that communication APIs to operate those AI methods needs to be defined and implemented. Few DL frameworks and platforms already offer methods and tools for that. For example, frameworks such as TensorFlow Serving[2] provides accessing the deep-learning model with gPRC or REST protocol. KF (Kubeflow) Serving[3] or Seldon Serving[4] provide the same functionality but they are framework-independent (see more details in Section 6.1). Nevertheless, all these solutions focus on AI model serving itself, rather than providing an end-to-end service, that is, from data source to service endpoint. In other words, to use those platforms, it needs to transform (pre-process) raw data (such as an image) so to make it fit for the input layer of the model, which might consume more computational resources.

- AI as a Service in DECENTER

DECENTER provides DECENTER AI package, which helps implementation of an AI (micro)service from an AI method. By separating network interfaces from AI methods, the package can help AI developers to turn their AI methods into a service with less effort. Moreover, DECENTER provides several design patterns with the package which makes it able to deal with various network architectures to build a service.

One of the important aspect of this package which differentiate it from other model serving is that it provides AI as a service, going beyond model as a service to includes pre-processing of the raw data as well as post-processing of output to make it easy to build an efficient tailored AI service which also make an efficient use of network resources.

2.3 Security and privacy

Often, AI methods need to deal with security and personal sensitive information. These are two important requirements that must be addressed in modern software engineering.[5]

The goal of our work is to research how different Cloud service consumers and Cloud providers can achieve security and privacy compliance related to sensitive data management in the context when the information is communicated and processed (e.g. by use of AI methods) across different administrative domains. One aspect is the achievement of end-to-end security in the communication channels as well as processing nodes. Our work aims at promoting (1) certification, with compliance being monitored by an independent authority, such as a certification authority that verifies the fact that a Cloud provider uses secure SGX chips; and (2) permission-based processing, where the users may decide in what way their personal information is processed, for example, based on reputation models of Cloud providers, and (3) regulatory aspects for the management of information that can be imposed by governments of states, for example, by specifying when and where can specific information be processed

[2] https://www.tensorflow.org/tfx/guide/serving
[3] https://github.com/kubeflow/kfserving
[4] https://www.kubeflow.org/docs/components/serving/seldon/
[5] https://www.sciencedirect.com/science/article/pii/S1877050916320944

and used. Current focus of researchers in this area is to achieve greater degree of transparency, explainability, risk assessment and audits in the overall process of AI-related data management.[6]

- Data (and Model) Management in DECENTER

Another part of our research and innovation activities addressed AI methods which make assertions about specific human subjects. For example, AI models operating on video streams and/or sensor data can be used to identify specific persons, their behaviour, physical and mental conditions and the like. This implies the use of sensitive personal data. The DECENTER project aims at securing the overall process of using such AI methods as containerised software artefacts. With this approach, AI methods can be packed into containers (see Chapter 4), then they can be stored in a repository, moved from one Cloud provider to another (or otherwise managed), from one could-to-edge (compute) continuum tier to another, and they can be started in different administrative domains. The ability to gain access to such artefacts equals the ability to gain access to the characterisation data of the human subjects. Hence, our work focuses on the possibility to secure the overall process for managing data as well as AI models, which requires end-to-end security.

2.4 Distributed data generation

Data pipelines for building AI models (for example, by means of ML techniques) typically consist of data collection from the systems' interaction with its environment, their storage and preparation (for example, to create training and test sets for ML training), and its processing either in a monolithic or distributed process. In modern AI applications, the optimization of AI models is meant to happen continuously and hence the training pipeline is usually planned to run periodically, following either a fixed schedule (e.g. weekly) or a drop in the model inference performance (e.g. in terms of accuracy). Also, an advanced form of ML, Deep Learning (DL), requires a vast amount of training data as well as compute resources to achieve a good performance. Due to all those constraints, ML engineers tend to deploy training processes in a centralized cloud or data centre instead of physically closer to the data source, where the data is generated in the first place (e.g. sensors). However, the benefits coming from training ML models on compute nodes closer to the sensors are becoming more and more attractive in advanced (high value) scenarios: more security and privacy and bandwidth savings as you don't need to send data far away from the user, and cost savings as you can use clusters of edge computing nodes or a swarms of things.

Another problem is the distributed generation of AI features, that is, the output of AI model inference processes). In an AI-based system running in parallel many of such processes (e.g. classifications, regressions, action selection, etc.), which may have been deployed along the cloud-to-edge (compute) continuum to run closer where their input data is generated, it is a huge challenge for the application developer to build a consistent and complete situational awareness capability. In decentralised AI applications, this is certainly the case as most features will be consumed by processes running closer to the feature-producing service, and hence AI features exchanges between services may disappear locally and never be integrated in a more complete representation.

[6] https://www.brookings.edu/research/protecting-privacy-in-an-ai-driven-world/

- Machine Learning decentralization by means of Federated Learning

Following comments from reviewers in the review of Y1, we decided to incorporate in the project the investigation of decentralized Machine Learning techniques; specifically, Federated Learning (FL) as the most promising of these techniques. This is a novel ML training approach to Deep Learning (DL) which uses decentralized data and infrastructure resources for a distributed training process. The main advantages brought by this technique is a distribution of the training process such that most of the workload can be executed in a decentralized fashion, closer to the data source. On the devices (e.g. smart things) where the data is produced and/or physically closer edge computing nodes. The development of an architecture for FL over DECENTER platform captured an important part of our WP4 effort in Y2. We developed a first version of a reference architecture for FL with DECENTER, which takes advantage of the application services as well as platform services provided by our Platform. This work is extensively presented in Chapter 7.

- Feature distribution by means of Digital Twin

One of the focus of DECENTER is to integrate IoT architecture patterns and technologies. DECENTER now provides methods to using the output generated by AI services to build Digital Twin representations. More specifically, in most DECENTER use cases, the AI features (i.e. inferences outputted by operationalised AI methods) of the respective AI applications are registered in the DECENTER Digital Twin application service and used for further exploitation such as Knowledge Base or user-oriented representation (e.g. to be presented through a mobile app for the user to follow certain parameters of the AI application) (see Chapter 5).

3 Methods for AI delivery from cloud to edge

One of the objectives of mapping AI onto the DECENTER platform includes bringing AI near to the device or to the service point for ensuring quick response from the AI service. Usually lots of computational resources are required for training and inferencing with deep neural networks. We investigate so-called model compression and acceleration methods to make the best use of computational resources and to ease these burdens of computation during training and inference. Model compression and acceleration is the research area, where the goal is to reduce the computation of the network while keeping the performance such as accuracy or precision. To be detail, we categorized the existing model compression and acceleration algorithms into four categories, and briefly introduce the key idea of each category. Additionally, we conduct the experiment on optimizing face detection algorithm using one of the model compression and acceleration called channel pruning. We put the result of pruned network compared to the original network with respect to the detection performance and the size of model.

In addition, we can take advantage of DECENTER platform to upgrade AI models. To be specific, there exists unlabeled data collected on the edge which has potentials to improve the AI models. We introduce semi-supervised learning algorithm for DECENTER to utilize unlabeled data to update AI models on edge, without any expensive labeling process. We conduct experiments with our proposed method and achieve performance improvement in basic computer vision task.

Note that, AI models employ lots of multimedia data which includes biometric information such as fingerprint, iris, and face images. Those personal data must be protected as it cannot be reused if the original data is leaked. In this regard, we propose an adversarial training approach to make AI model robust from the original data, even if it is trained with personal information. Specifically, we perform a research to make adversarial examples and exploit it for training to prevent AI model from saving restorable representation learning. In the followings, we describe the details of the above proposals.

Last but not least, this delivery of AI is well-suited for recent researches on Federated or Distributed learning, which makes use of edge (or worker) for the training since those processes involves exchange of AI models between instances. DECENTER has investigated this aspect of the Federated Learning (FL), and applied the Platform to build an image classification service which makes use of updating model with FL. The details on FL implementation is given on a separate chapter – Chapter 7.

3.1 Survey on Deep Neural Network Compression and Acceleration

Deep Learning (DL) requires very large datasets to successfully train deep neural networks (DNN) to do the demanding tasks. This means the network needs to go through all the data points in the training set several times to learn the mapping function between the problem and the solution. This training process can take from hours to days and therefore it normally results in high computing infrastructure costs.

In the last decade, technical advances in graphical processing units (GPUs) along with constant scientific progress in DNN methods and tools have drawn huge attention toward AI-based systems powered by DL. One of the keys for this trend has been the availability of GPUs with high computational capability specialized on computing tasks demanding high memory bandwidth and parallelism. Specifically, the recent GPUs are more capable of doing computations with large amount of memory such as matrix multiplications, which is directly related to the ability of computing basic linear algebra operations, which are the basis of DL

methods. This results in DNN computing speed being boosted when running on a GPU instead of a CPU.

We need to consider several aspects when developing a DL method for a real-world application. On the one hand, operationalising a DL algorithm on portable devices or embedded systems with limited resources such as memory, compute, GPU, power, etc. presents important challenges. On the other, training a DL method to provide reasonably quality results (e.g. in terms of precision) is a very resource-consuming process. Furthermore, most state-of-the-art real-world DNNs need to possess docents of layers with millions or billions of parameters, with the amount of resources and operations needed directly related to the number of parameters of the model. Therefore, it is increasingly important to compress and to accelerate DNNs so that they can be deployed in real world applications.

There are numerous research papers on compressing and accelerating the DNNs. There may be different ways to classify existing approaches and it can be controversial among the researchers, but for the convenience of our research and innovation in DECENTER, we have classified them into four main categories.

1. Parameter pruning and sparse connection
2. Grouped convolution method
3. Quantization and binarization
4. Knowledge distillation

These methods can be applied on the DNN independently or in combination to magnify the performance of compression and acceleration, so some methods suggest combining these methods to complement each other. In this section, we will shortly introduce the methods listed above. Since the methods related to parameter pruning and sparse connection is used on the next section (Section 3.2) on a face detector algorithm to compress and accelerate the model, in the following subsection, we will explain in more detail this category than the other three.

Parameter pruning and sparse connection

The first method to compress and to accelerate the network uses the so-called pruning technique. The network has some redundancy over channels of the feature map, filters of the convolutional layer, etc and the key idea of pruning is to remove these redundant channels or parameters which are not critical to the model performance. A simple original neural network can be expressed as the output matrix Y being the result of the input matrix X multiplied by the weight matrix W. The pruned network can be expressed as a channel selection matrix that is multiplied onto the input or weight matrixes to output Y' (see Equation 1, Figure 1). The goal is to find a new weight matrix which makes Y' as close to Y as possible. Which channels or parameters to prune differs from method to method and it depends on the algorithm.

Equation 1 Objective function of pruning and sparse connection

$$\arg\min_{\beta,W} \frac{1}{2N} \left\| Y - \sum_{i=1}^{c} \beta_i X_i W_i^{\top} \right\|_F^2$$
$$\text{subject to } \|\beta\|_0 \leq c'$$

Y: Original output
X: Input
W: Weight
B: Channel selection

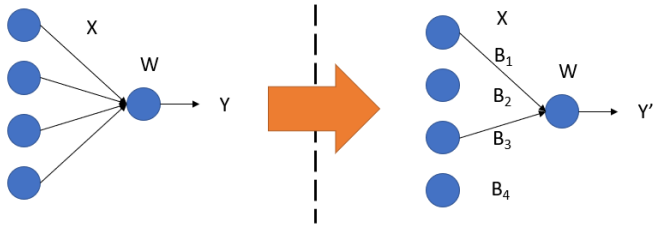


Figure 1 Goal of pruning is to produce a Y' that resembles Y as much as possible.

The typical training process would consist of a three-step training pipeline [5]. First, the standard network needs to be trained so it learns the weights of connections between neurons (see Figure 2). Second, we prune some connections whose contribution to the model performance is not considered critical. Lastly, it retrains the remaining weights so that the output of this new network closely reproduces the original output (with all the connections enabled). After that, the second and the last steps are iteratively repeated until the pruning process is done. After all of the process, the compressed and accelerated network can be run with fewer parameters and FLOPs.

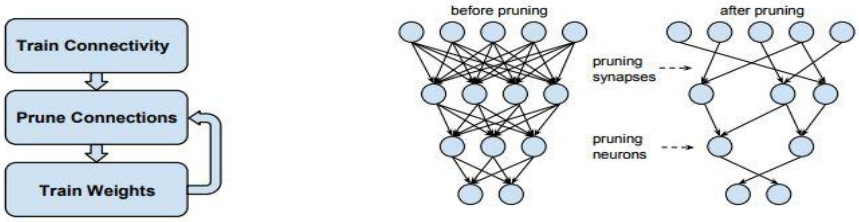


Figure 2 Three step pipeline of pruning and comparison of neurons before and after pruning [5]

The pruning technique is the so-called channel pruning [4]. This method tries to remove the redundant channels in the feature map. There are many algorithms have been researched to identify and to prune the redundant channels. One of well-known algorithm is to use modified LASSO regression for pruning. By removing the redundant channels, some parameters in the weight matrix that is multiplied onto those channels are not used anymore. Therefore, the weights matrix that is multiplied to compute the feature map and the weight matrix in the next layer to compute the next feature map have in common that not all the parameters in the matrices are used (see Figure 3). In other words, these useless parameters can be removed now from the network.

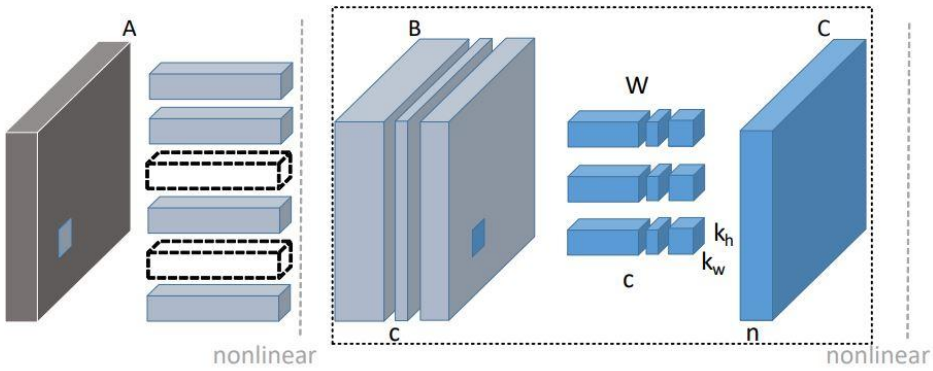


Figure 3 Overview of channel pruning [4]

Grouped convolution-based method

As mentioned above, convolution operations represent the bulk of most computation in DNNs. The key idea behind the so-called grouped convolution-based method is to replace the computation in convolution with other operation so that it would improve in compression rate as well as computation speed. The basic idea is similar to the singular value decomposition (SVD)[8]. A matrix can be decomposed as a product of three matrices. If we keep the largest singular value and drop the rest, which is well known as low-rank matrix approximation, the approximation of the original matrix can be achieved. The best-known method is to perform SVD on fully connected layers. The fully connected weight matrix can be decomposed using SVD so that the number of weight parameter drops.

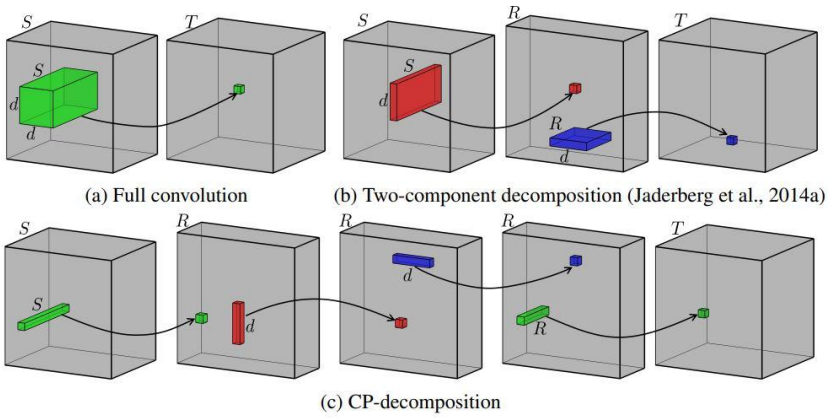


Figure 4 Overview of standard convolution and decomposed convolution methods

A tensor can be decomposed in the same way a matrix is decomposed with SVD. One of the best-known methods to do this is called CP-decomposition (see Figure 4.). It is very similar to SVD and the weight tensor can be decomposed in the multiplication of four different matrices. Lebedev et al. [8] use this idea to decompose the convolution operation and so significantly reduce the number of parameters.

MobileNet [9] is the best-known method to replace standard convolutions with depth-wise convolution and point-wise convolution, which are so-called separable convolution. It also keeps spatial invariant features of convolutional neural network, and it is much more computationally efficient than the standard convolution. This method significantly reduces the computation cost by 1/N+1/D^2, where N is the number of output channels and D is the kernel size. It is so efficient that object detector based on MobileNet is widely used on smartphones nowadays.

Quantization and binarization

The network can be compressed by quantization and binarization by reducing the number of bits required to represent each weight. This can result in a significant speed-up with minimal loss of accuracy; for example, Vanhoucke et al. show those benefits when quantizing 8-bit parameters [6]. Courbariaux et al. use binary representation for parameters so that network is consisted of binary weights and it is extreme case of binarization of the network [7].

Knowledge Distillation

Knowledge distillation is also known as teacher-student network [22]. A teacher network is a standard large and complex neural network and the student network is a shallow and light-weight network. The key idea behind this method is the student network learns to mimic the

teacher network. The student network can be trained to mimic the intermediate result such as feature map or class probability depending on applications.

3.2 Channel pruning on face detector

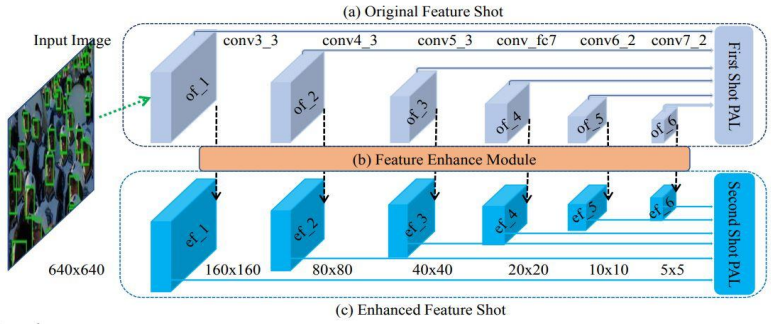


Figure 5 Architecture of DSFD [3]

In the experiment, the face detector called dual shot face detector (DSFD) [3] is used. DSFD achieves a high accuracy by performing two shots. In the first shot, the detector tries to find the faces which can be easily distinguished from the background or large faces compared to the original image size. In the second shot, it reuses the features created in the first shot to combine features from different scales and recreate contextually rich features. Thus, the rest of faces, for example those hardly distinguishable from the background or small faces compared to the original image size, can be found using those features identified in the first shot. The architecture of DSFD is depicted in Figure 5.

DSFD is trained with a publicly open dataset called WIDER FACE dataset [23]. It has 32,203 images with 393,703 faces. The images in that dataset present a high degree of variety in scale, pose, occlusion, blurry, expression, makeup, illumination, etc. Therefore, it gives high level of robustness to DNNs trained with it. Figure 6 shows the detection result of DSFD.

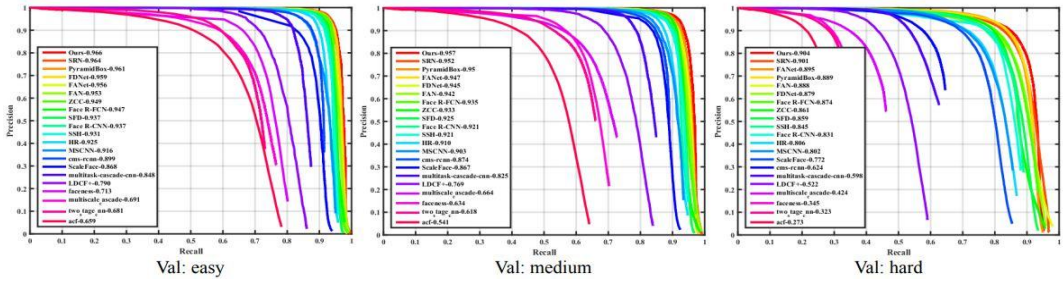


Figure 6 Detection performance of DSFD [3]

As shown in Figure 6, DSFD shows the state of art performance in face detection task. However, there are several aspects to be considered when using DSFD solely in DECENTER platform. Since it consists of a two-phases detection process, the computation is much heavier than in single shot detectors; moreover, it also has more parameters because more convolutions are involved in the second shot detection phase. Furthermore, it takes up a lot of memory because feature maps extracted from in the first shot detection phase are reused in the second shot.

In our experiment to use a parameter pruning and sparse connection method in DSDF, two objective functions are used for network compression and acceleration. The L1 norm is known as the tightest relaxation of the L0 norm, which has a direct relation with sparsity [24]. L1 norm as the objective function was used instead of an L0 norm since gradients can be calculated

with L1 objective function. In contrast, the gradient cannot be calculated on L0 norm. In equation 2 where the total number of layers is L, the size of the feature point map is C, the height is H, the width is W, and a pixel of the feature point map is represented by $x_{l,h,w,c}$, the first objective function applied the L1 objective function directly to each pixel of the feature map to sparsify it, which means to have more zeros on the feature map. For the second term of the objective function, L1 norm-based objective function was used to prune the channel. After finding the maximum value for each channel from each layer, L1 norm-based objective function was applied and the lowest 20% of channels among all channels C were used in calculation. That is, it is assumed that a channel with low activation has little effect on detection performance, and a channel with high responsiveness has a high effect on the result. Therefore, the objective function is applied to induce sparsity only on the channels with low activation. Finally, α in front of the second function was used as a hyperparameter that weights between the two objective functions.

Equation 2 Sparsification loss

$$\sum_{l}^{L}\sum_{h}^{H}\sum_{w}^{W}\sum_{c}^{C}|x_{l,h,w,c}| + \alpha\sum_{l}^{L}\sum_{c\in C'}|\max_{h,w}(x_{l,h,w,c})|$$

The neural network compression and acceleration training process was conducted in a two-stage pipeline. First, a standard neural network was trained with a WIDER FACE training dataset to find connectivity between parameters. In the second step, the compression and acceleration objective function in the equation 2 was additionally applied to the original detection objective function used in the existing neural network. The purpose of compression and acceleration objective function is to remove some connections that do not significantly affect the performance of the neural network. For hyperparameters, α in Equation 2 was set to 0.1, and C' was set to the lowest 20% channels with the lowest maximum value for training.

For validation set of WIDER FACE [2], performance was evaluated by calculating the size, the number of parameters and average precision (AP) as shown in Table 1 and 2. The size of pruned network reduced down to 134.6 MB, which is 78.85% of that of the original network. Also, the number of parameters has reduced down to 33.6 million from 42.7 million. However, the detection performance only showed 9.41% of degradation at most on hard WIDER FACE validation set, which was 0.7763 average precision. On easy and medium set, the performance degraded only about 4% compared to the original average precision.

For validation set of WIDER FACE [2], performance was evaluated by calculating the percentage of activations and average precision (AP) as shown in Tables 1 and 2. The highest case for the percentage of activated feature map was 30.33%, where both objective functions were applied. When only channel pruning was applied, the highest case for activated channel was 20.00%, but the activated feature map was the highest. When both objective functions were applied, the detection performance was the worst as the feature point map was the highest.

Table 1 The size and the number of parameters of the original network and the pruned network

	Size	The number of parameters
Original	170.7 MB	42,748,434
Pruned Network	134.6 MB	33,680,043

Table 2 Average precision on WIDER FACE validation dataset

	AP on easy set	AP on medium set	AP on hard set
Original	0.9431	0.9320	0.8569
Pruned network	0.9129	0.8966	0.7763

3.3 Semi-supervised learning on the edge

AI models that currently show the best performance in most computer vision tasks are based on deep supervised learning, which utilizes tons of elaborately labelled data. The more the labelled data exists, the better the AI model performs. However, there is limitations that producing labelled data is costly, and plenty of unlabelled data is not exploited for training. The concept of semi-supervised learning which employs both labelled and unlabelled data has getting more attention to alleviate this problem.

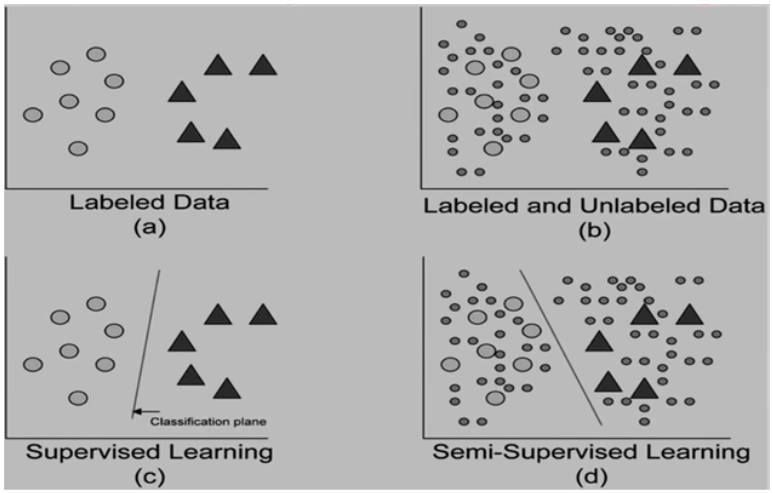


Figure 7 A conceptual illustration of semi-supervised learning

As shown in Figure 7 (a) and (c), the classification plane of AI model which discriminates two different class (circle and triangle) is determined by training with labelled data. Since this AI model classifies solely based on the labelled data, the performance can be degraded for the data distant from the training set. In contrast, as illustrate in Figure 7 (b) and (d), the AI model trained with both labelled and unlabelled data has more robust classification plane, which increases the classification accuracy on the variant test set.

From the viewpoint of DECENTER, data collected on the edge can be considered as unlabelled data, which contains potential to improve AI models. We develop a semi-supervised algorithm to explore unlabelled data to update AI models for DECENTER. Especially, we adopt deep image retrieval task [18] which requires accurate understanding of complex image representations as an example study.

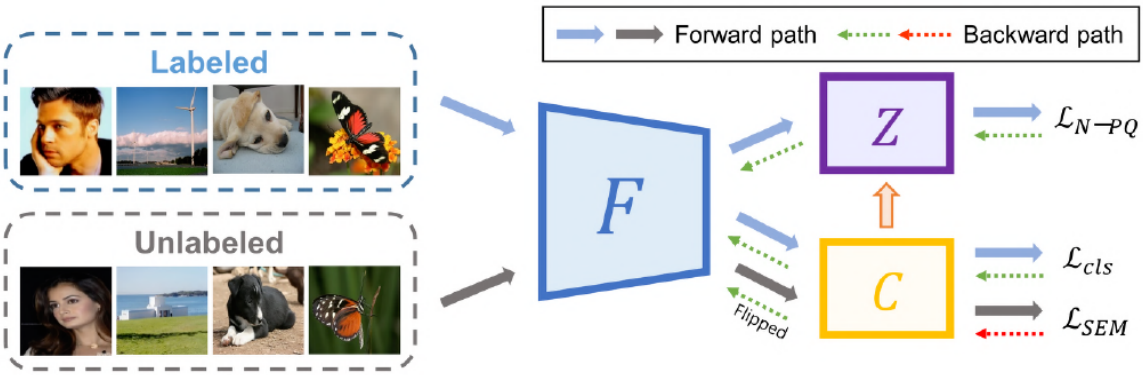


Figure 8 An illustration of semi-supervised learning scheme

As shown in the Figure 8, we develop an AI method for image retrieval (Generalized Product Quantization; *GPQ* [19]) by means of a semi-supervised learning algorithm. It contains three parts: *F (*feature extractor), *Z (*product-quantization codebooks), and *C (*classifier). Forward path shows how the labelled and unlabelled data pass through the network, and backward path shows the propagation of the gradients originated from the training objectives.

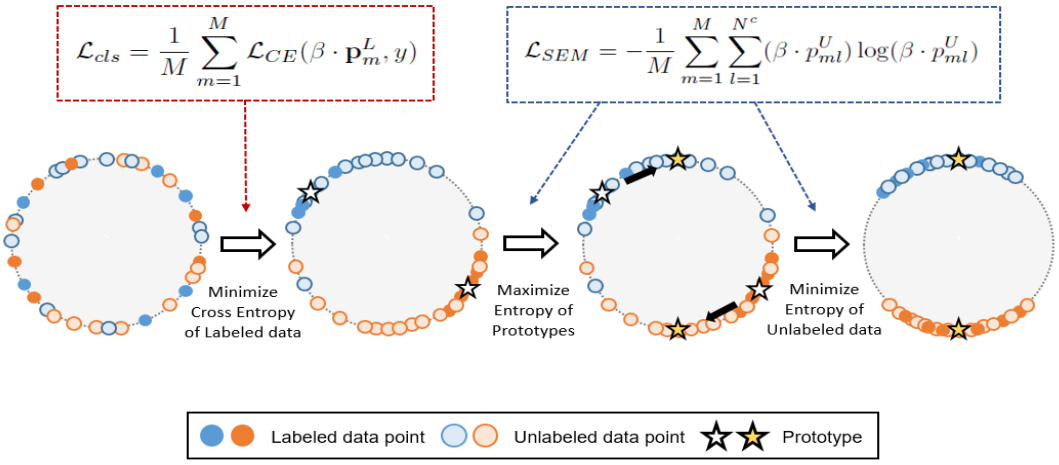


Figure 9 An example of semi-supervised learning process.

The process of semi-supervised learning is visualized in Figure 9. For the data points constrained on the unit hypersphere, the cross entropy of labelled data points is minimized to find prototypes (white stars). Then, the entropy between the prototypes and the unlabelled data points is maximized to move prototypes toward unlabelled data points and find new prototypes (yellow stars). Finally, the entropy of the unlabelled data points is minimized to cluster them near the new prototypes.

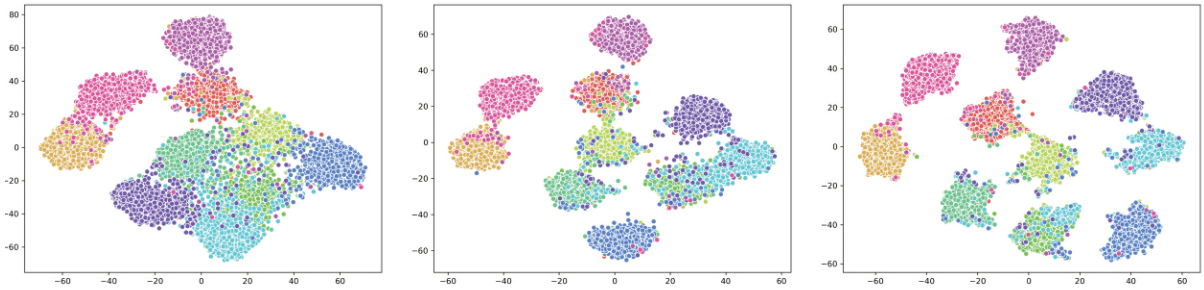
Table 3 mAP score comparison with other retrieval methods.

Concept	Method	CIFAR-10				NUS-WIDE			
		12-bits	24-bits	32-bits	48-bits	12-bits	24-bits	32-bits	48-bits
Deep Semi-supervised	GPQ (Ours)	**0.858**	**0.869**	**0.878**	**0.883**	**0.852**	**0.865**	**0.876**	**0.878**
	SSGAH [10]	0.819	0.837	0.847	0.855	0.838	0.849	0.863	0.867
	BGDH [36]	0.805	0.824	0.826	0.833	0.810	0.821	0.825	0.829
	SSDH [40]	0.801	0.813	0.812	0.814	0.783	0.788	0.791	0.794
Deep Quantization	PQN [38]	0.795	0.819	0.823	0.830	0.803	0.818	0.822	0.824
	DTQ [19]	0.785	0.789	0.790	0.792	0.791	0.798	0.808	0.811
	DQN [2]	0.527	0.551	0.558	0.564	0.764	0.778	0.785	0.793

We set up two benchmark datasets as [17], CIFAR-10 of 60,000 color images, which employs 5,000 labelled data, 54,000 unlabelled data and 1,000 test data, and NUS-WIDE of 169,643 color images, which employs 10,500 labelled data, 157,043 unlabelled data and 2,100 test data. The retrieval performance of hashing method is measured by mAP (mean Average Precision) with bit lengths of 12, 24, 32, and 48 for all images in the query dataset.

As reported in Table 3, we can observe that our proposed method shows the best result in deep image retrieval task which contributes from semi-supervised learning. In addition, our GPQ (on the right side of the Figure 10) shows the most discriminative visualized results than the variants of GPQ; GPQ-T and GPQ-H with the different training objectives, demonstrating the effectiveness of our work.

Moreover, as illustrated in Figure 11, the graph with hyper-parameter R which represents the ratio of the total unlabelled data used for training, shows that increasing the number of unlabelled data improves the retrieval performance of AI model. In summary, GPQ can fully utilize the unlabelled data and increase the robustness of AI model for improvement, which can be applied on various edge AI models in DECENTER.



(a) GPQ-T (b) GPQ-H (c) GPQ

Figure 10 t-SNE visualization of GPQ

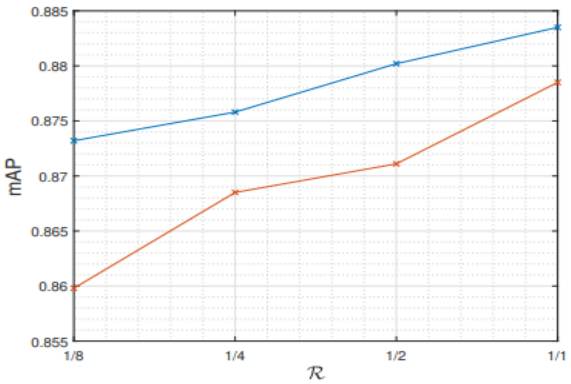


Figure 11 t- The graph of semi-supervised learning results (mAP score), by varying the amount of unlabelled data; blue line for CIFAR-10 and orange line for NUS-WIDE

3.4 Deep adversarial training to keep privacy

With a wide variety of biometric data, including face images, fingerprints and iris images, being increasingly exploited to train DNNs, therefore, the risk of compromising personal data included in AI model is rising rapidly. In DECENTER, we have conducted research on the use of adversarial training to keep privacy of AI models. Specifically, on defending attacks on deep image retrieval system to restore biometric information contained in AI model by introducing adversarial training [20] in the process of neural network learning.

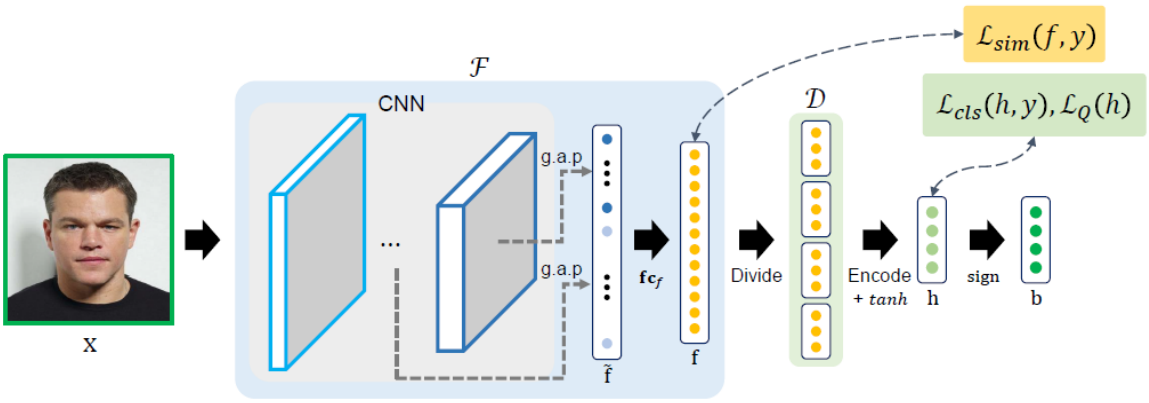


Figure 12 Adversarial training for image retrieval

The basic idea of adversarial training is including adversarially perturbed images into the training dataset. To do this, we first need to generate adversarial noise, which poisons original image when added and fools the AI model to output undesired outputs.

In this case, perturbation is generated with one of the most famous algorithms termed Projected Gradient Descent (PGD) [21]. We assign the same class labels to the adversarial examples with the original data and include them for AI model training to increase the robustness toward perturbed images.

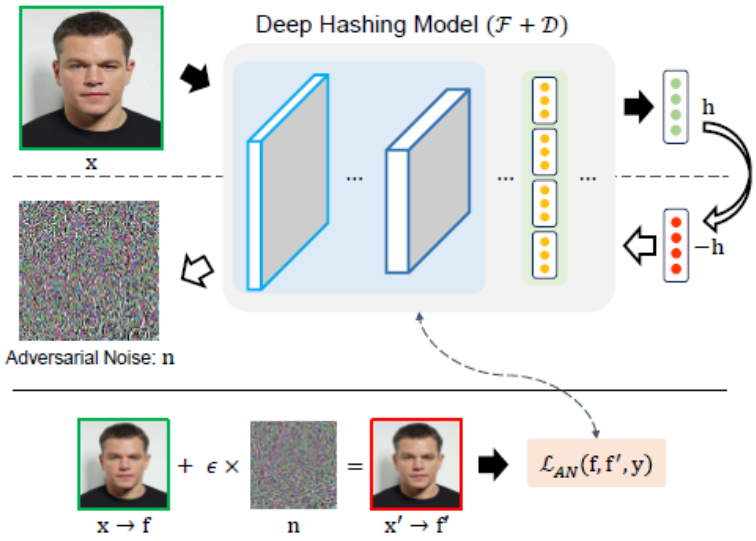


Figure 13 The process of adversarial example generation

In detail, we first train the network to output hash code vector h, and then by change the sign of it, we obtain the opposite hash code -h. We utilize gradients originate from -h to generate adversarial noise n, which may harm the original input data. Therefore, we train the network one more time with the noise added image (red squared image in Figure 13) to make the network robust toward adversarial attacks.

In this manner, we are able to build AI model that trained with both original data and adversarially perturbed data, which increases the robustness of AI model and results in containing least biometric information in itself.

If we properly using this algorithm, we can keep the personal information of each image from saving in AI model and also generating non-invertible feature representations. As a result, we can decrease the privacy concerns in AI models.

3.5 Remarks

In section 3.1, we have investigated the various methods of model compression and acceleration that can be reduce the burden of computation in the resource constrained devices. In section 3.2, the channel pruning, which is one of the model compression and acceleration method, was implemented on the face detector called Dual-Shot Face Detector (DSFD). The experiment was conducted on WIDER FACE dataset, and the pruned detector showed the reduced size of parameters while maintaining moderate detection performance compared to the standard detector. In section 3.3, the semi-supervised learning algorithm is explored for DECENTER. By utilizing both labelled and unlabelled data, AI models are upgraded to perform better. Especially, unlabelled data collected on the edge device can be exploited for AI model improvement without sending any personal data to the cloud. In section 3.4, the infringement of biometric data included in AI model is considered. Since biometric data cannot be replaced when leaked, the adversarial training approach is studied to increase the robustness of AI model, reducing privacy concerns.

In the third year, we are planning to conduct experiment on other model compression and acceleration methods such as quantization to reduce computation burden. Also, unsupervised and self-supervised training methods which need less personal information will be regarded for DECENTER AI model optimization.

4 Updates on Containerization of AI methods

This section describes the activities on the development of DECENTER toolset for AI method containerization into microservices carried out during the second year (Y2) of the project. The main asset if the so-called DECENTER AI package, a python library which facilitate ML model serving as microservices. The first version of the DECENTER AI Package is described in D4.1, Section 4 [11]. In Y2, the following activities have taken place to enhance it:

- AI Model management with AI model repository.
- Interfaces of AI microservice.
- AI container configuration methods for Docker and Kubernetes
- Labeling of GPU node

4.1 AI as a Microservice - Deployment and Management

The structure of AI containers was defined in D4.1[11]. DECENTER has designed a container that provides an AI microservice from an AI method by means of the DECENTER AI Package. The software stack for such microservice has been identified and how to containerize that software stack has also been defined. In the four DECENTER use cases, many AI microservices have already been developed and deployed as containers on the DECENTER platform (see D5.1 [17]).

In Y2, the hardware and software stack for an AI container have been updated as we extend our scope from the cloud to the edge nodes. Unlike the typical cloud infrastructure, where all the resources are considered infinite and homogeneous, the computing infrastructure usually has finite resources and the resources may differ from each other on the edge. It needs clear ways to identify different types of resources such as whether the architecture which it is based on is ARM64 or AMD64, type of GPU or the memory assigned for GPU computation. If the constrained resources of edge are not described properly, the deployed AI container will not run properly. To this end DECENTER has defined several types to describe custom resources which are related to AI computation. The way to request GPU nodes or other AI-related resources by an AI microservice is described in D3.3 [12], and the structure of AI container has been updated accordingly.

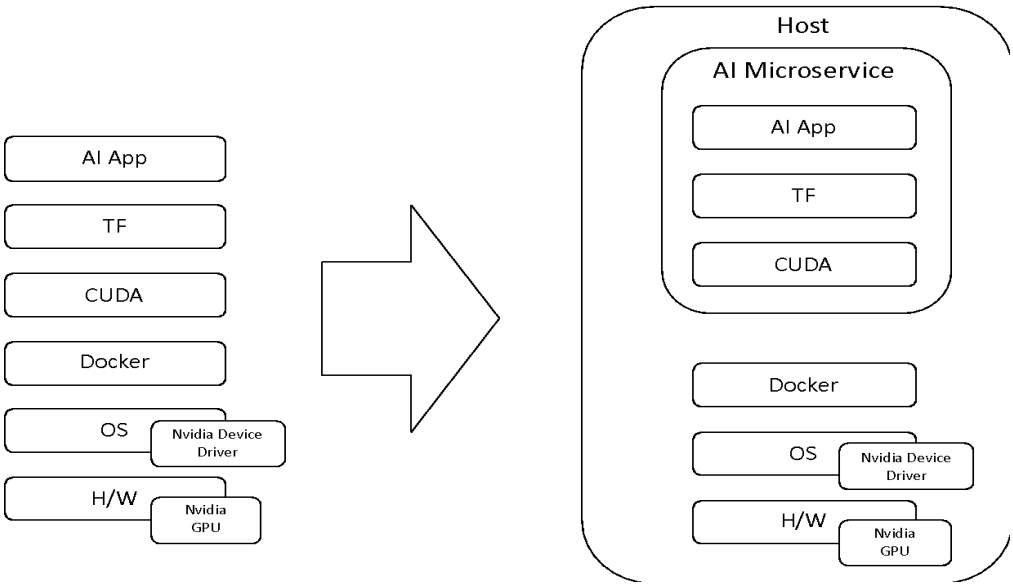


Figure 14 Software stack for AI application and virtualization

Figure 14 depicts the proposed container structure of an AI microservice in DECENTER. In Y1 we focused on the containerization of AI microservices; in Y2, the scope was extended to include support for tensor computing acceleration devices such as GPU (Graphics Processing Unit). On the left-hand side of the Figure, the software stack of AI microservice is depicted, with the example of Nvidia GPU device. The AI microservice is built on the top of a deep learning framework (TensorFlow or TF in the Figure). To use acceleration hardware, another software pieces must be inserted in various layers of the stack, including libraries for acceleration (CUDA), device driver (Nvidia Device Driver). The container structure is depicted on the right-hand side of the Figure. The structure is basically the same as the one of Y1; however, the relationship between container and host requirements are now identified and added for the container description.

Due to different architectures that GPU devices are based on, there are huge dependencies between acceleration libraries (CUDA in the figure) and device driver (Nvidia device driver in the figure). For example, to run an AI application with CUDA 10.0, it needs a host with device driver version higher than 410.48[7]. This dependency restricts on the deployment of an AI microservice since device drivers are part of the host resources and not of the container.

The memory of those GPU devices also needs to be considered when deploying an AI microservice requiring them. Usually, a dedicated memory is associated to a GPU device for faster and reliable execution of computation, instead of sharing memory with the host CPU. The size of GPU-memory is fixed by the host hardware installation. However, for the SoC (system on Chip) device where a GPU is integrated with a CPU, this is different: in that case the system memory is shared between CPU and GPU and there is no dedicated GPU-memory.

In summary, for an AI microservice to request a GPU node, it needs to specify at least three pieces of information within the deployment manifest of the AI service container on the host: GPU device installed on the node, device driver compatibility (version number), and the memory associated with GPU device. All three are needed to properly deploy AI microservices through the DECENTER platform, especially on resource-constrained edge infrastructure. They are fully defined in D3.3 [12].

4.2 New features of the DECENTER AI package

Figure 15 depicts the container configuration with the first version of the DECENTER AI package, developed in Y1. The interfaces to configure and operate the AI methods are implemented with Flask server[8], which is a Python implementation of HTTP service, and the AI method itself is implemented on a separate class which inherits from BaseClass (decenter.ai.baseclass). To implement an AI service with DECENTER package, AI developer needs to firstly build this new class inheriting from decenter.ai.baseclass and override a few methods with its own logic.

[7] https://docs.nvidia.com/deploy/cuda-compatibility/index.html
[8] https://flask.palletsprojects.com

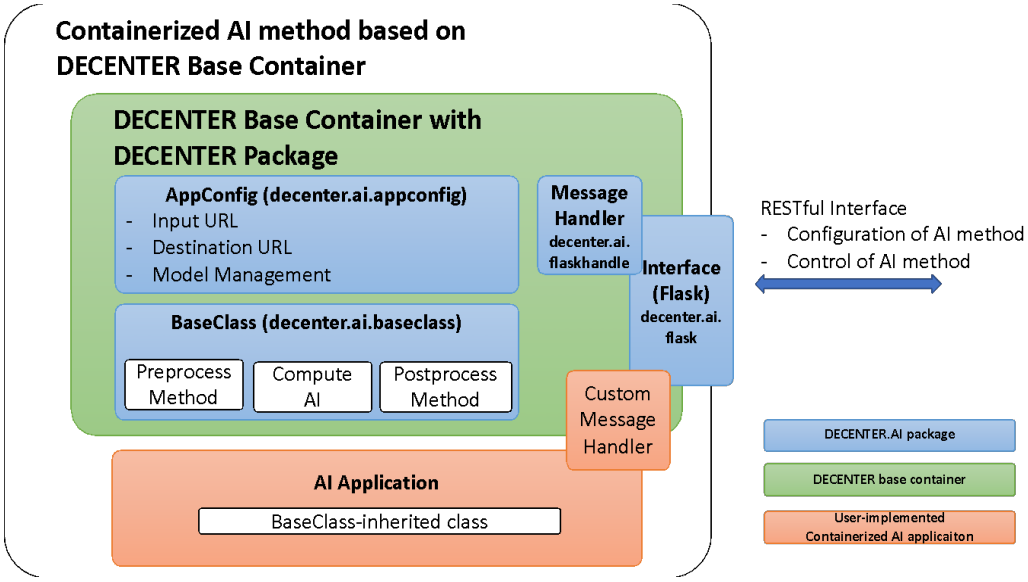


Figure 15 First version of the DECENTER package, from Y1. The AI method or logic is implemented in a child class of decenter.ai.baseclass and only HTTP protocol is supported

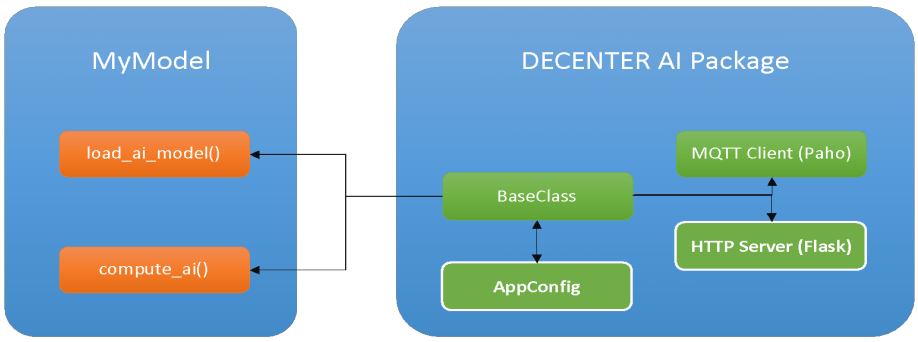


Figure 16 New version of the DECENTER AI package, from Y2. The AI method execution, which is supported within the BaseClass, is now separated from the Interface modules, which now support HTTP and MQTT protocols

Figure 16 depicts the structure of the new version of the DECENTER AI package, develop in Y2. The basic idea is the same: to store all the configuration variables on AppConfig, to manage interfaces through message handlers, and to implement the main logic to run the AI method in the BaseClass. The main difference is that, now the AI logic from DECENTER AI package. Instead of implement AI methods on the child class of BaseClass, which was the approach from the first year, the updated version uses a new skeleton class named as MyModel. The BaseClass will work as a bridge to call pre-defined methods on MyModel thus make it easier to implement the AI methods with DECENTER AI package. Also, the Interface modules have been extended to support MQTT protocol—for asynchronous operation of the AI method—along with the HTTP protocol—for synchronous operation. Furthermore, the DECENTER AI package interoperates now with DECENTER AI model repository to manage the update of the AI model, which is trained and versioned outside the AI microservice. The AI model repository is part of the cross-border data management capability and it is described in D4.2 [13]).

4.3 Design Pattern of AI Microservice with DECENTER AI Package

The DECENTER AI package offers few facilities to support a flexible configuration of AI methods into microservices. Some design patterns can be exercised by using those facilities; they are shown in this section.

These are the pre-requisites to use DECENTER AI package:

A. Design your application architecture using a microservices style. This includes the identification of the AI microservices' input and output data,as well as the communication protocol of their interfaces.
B. Prepare your AI model to be used. The AI model needs to be train (built), serialized into file(s). Those files can be packaged (built) into the microservice container image itself or downloaded from a model repository server at runtime.
C. The logic of the AI method needs to be implemented in the template class My_Model.py, the skeleton provided by the DECENTER AI package.

4.3.1 Flexible configuration for microservice

DECENTER AI package supports the following parameters for configuring an AI microservice.

- Input source URL: URL of input data to be analysed.
- Output destination URL: URL of destination where the analysis result needs to be delivered.
- AI model data: AI model information, e.g. the neural network weights.
- URL of the model repository server where to download the AI model from.
- AI model name and version number to be loaded by the AI method.

DECENTER AI package supports two types of configuration to cope with various running environments for the AI microservice: 1) the runtime configuration uses a RESTful API to read and write configuration parameters values, and 2) the deployment configuration uses OS environment variables to pass configuration parameter values to the container.

The RESTful API for runtime configuration is presented in Table 4. This type of configuration is useful, for example, when an AI microservice receives as input an image which is specified by a user through the AI application GUI.

Table 4 RESTful API structure of DECENTER AI package

Endpoint	Operation	Parameter	Description
/set_input	GET	url	Set input url ex) http://myservice/set_input?url=http://keti.re.kr/a.jpg
/get_input	GET	none	Retrieves configured input URL Ex) http://myservice/get_input
/set_output	GET	url	Set output url ex) http://myservice/set_output?url=http://keti.re.kr/a.jsp

/get_output	GET	none	Retrieves configured output URL Ex) http://myservice/get_output
/set_ai_model	GET	server url, model name, model version	Set AI model to be used Ex) http://myservice/set_ai_model?server=http://keti.re.kr/model?model_name=vgg16?model_version=0.1
/compute	GET	none	Request computation of AI method. The results will be returned with the response packet or delivered to the output destination.

If the AI microservice is static – input/output location (i.e. URL) does not change frequently during the lifetime of the service deployment –the deployment configuration is more convenient. In this way, all the configuration parameter values are written on a system variable with the name of 'MY_APP_CONFIG'. The DECETER AI package will try to load those values from 'MY_APP_CONFIG' variable when the AI microservice is started.

One straightforward way to specify the 'MY_APP_CONFIG' variable to a containerised AI microservice is in the Dockerfile itself, where environment variables can be specified in the following way:

```
MY_APP_CONFIG="{
      "input": { "url": "input_source_url"}, # HTTP of MQTT
      "output": {
            "url": {"name_of_output": "destination_url" # MQTT
            # repeat as needed
          },
      "ai_model": {
            "url": "http://servername:port", # string
            "model_name": "name_of_model", # string
            "model_version": "version_in_float", # float
            },
      "autostart": { "value": "True or False" }
}"
```

Nevertheless, when deploying the AI microservice on the DECENTER platform, we recommend using a Kubernetes ConfigMap such as the following:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: uc4-fd-config
data:
  appconfig:
    {
      "input": {
      "url": "http://decenter.keti.re.kr"
      },
      "output": {
```

```
    "url": {
            "TYPE_FI": "mqtt://uc4-mqtt:1883/face_image",
            "TYPE_FI_DT": "mqtt://uc4-mqtt:1883/face_image_dt"
            }
    },
    "ai_model": {
        "url": "http://182.252.132.39:5000",
        "model_name": "decenter_mnist",
        "model_version":"0.1"
    },
    "autostart": {
      "value": "True"
    }
  }
```

4.3.2 Design Pattern #1: Client-Server

You can build an AI application which follows a client-server architecture pattern through the built-in Flask server included in the DECENTER AI package; this offers a pre-defined RESTful API to configure and operate an AI method.

Figure 17 shows the client-server architecture of an application with an AI service based on the DECENTER AI package which is consumed by another microservice (AI Service Client). The service client will issue HTTP request messages to the API of the Flask server.

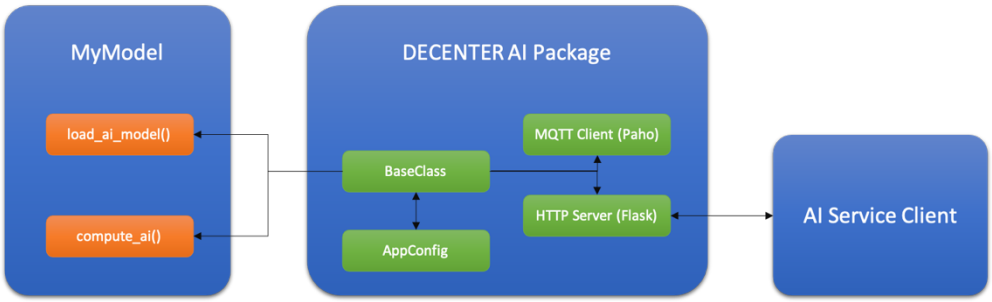


Figure 17 Configuration of client-server design pattern

This design pattern is good for a service which needs to make random synchronous requests to the AI service when needed. Image classification or face verification can be good examples for this design pattern. The client will make an HTTP request to the AI service, and the result will be returned in the response message.

- Configuration

The example configuration for this design pattern is as follows:

```
MY_APP_CONFIG="{
      "input": { "url": ""}, # Empty
      "output": {
            "url": {"name_of_output": "" # Empty
          },
      "ai_model": {
            "url": "http://182.252.132.39:5000",
            "model_name": "decenter_mnist", # string
            "model_version": 0.1, # float
```

```
            },
        "autostart": { "value": "False" }
}"
```

Note that the input and output fields have empty values since all the request will be made via RESTful interfaces (/compute).

- Control the AI method

The examples for RESTful APIs to set input and process AI are as follows:

http://myservice/set_input?input=http://decenter.keti.re.kr/a.jpg

http://myservice/compute

The classification result will be returned in the response packet of the second HTTP request (/compute).

4.3.3 Design pattern #2: Event-driven design pattern with MQTT Protocol

You can build an AI service which interacts asynchronously with other microservices of your AI application via a publish/subscribe mechanism such as MQTT[9]. MQTT is one of the famous implementations of Pub-Sub protocols. It is designed to support lightweight communication and is suitable for data transmission of IoT devices. Figure 18 shows this design pattern. In the figure the MQTT client is used for exchange of data for AI method or the output of it, while the service client can access the AI service via HTTP protocol.

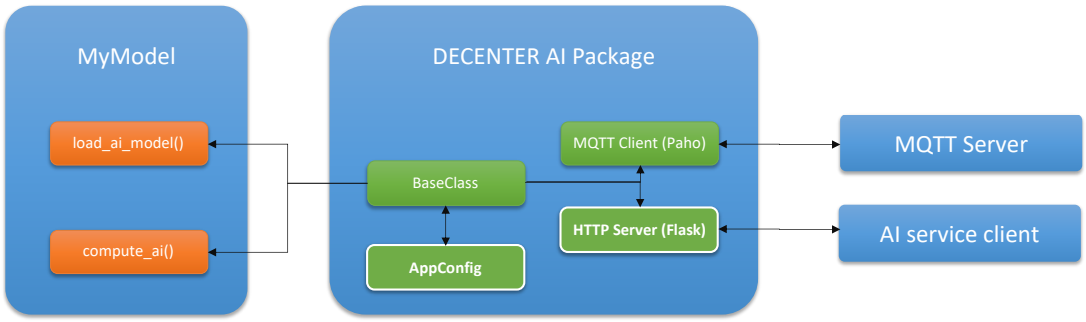


Figure 18 Configuration of event-driven design pattern

The input data will be fed into the AI service via MQTT, and the output of the AI method will be sent to the MQTT destination. This design pattern is suitable for the AI service which interacts with IoT applications.

- Configuration

The example configuration for this design pattern is as follows:

```
MY_APP_CONFIG="{
        "input": { "url": "mqtt://decenter/input_topic"},
        "output": {
              "url": {"result_type_b": "mqtt://decenter/output_topic_b"}
            },
        "ai_model": {
              "url": "http://182.252.132.39:5000",
              "model_name": "decenter_mnist", # string
```

[9] https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

```
            "model_version": 0.1, # float
            },
      "autostart": { "value": "True" }
}"
```

Note that the “autostart” parameter has value of True. Since there will be no explicit request to the AI method to start operations, this AI microservice runs automatically from the very moment it starts receiving messages from input URL.

- Operation

When it’s initiated, the DECENTER package will download the AI model specified in the configuration and will load it into memory. Later, it subscribes to the MQTT topic designated by the “input” parameter in the configuration. When an MQTT message is received, it will call MyModel.compute_ai() and sent an MQTT message with the result to the output destination topic.

4.3.4 Design pattern #3: Streaming data processing

If you want to build an AI service which receives and processes video stream, you can use the first design pattern (client-server) or use this design pattern. Basically, the configuration of this design pattern is the same as the first one, however this design pattern requires continuous output of AI computation from just one request from the client. This design pattern can be applied to many use cases which includes video stream. Suppose that there is an AI service which receives a video stream from an IP camera for some sort of analysis. Upon receiving request, the AI service will detect objects from the video stream and transmits output for every from in the video stream. To support this design pattern, DECENTER AI package can be applied as a separate thread in a process, with a message queue to interact with the AI method in MyModel.

- Configuration

The example configuration for this design pattern is as follows:

```
MY_APP_CONFIG=”{
      “input”: { “url”: “http://my_camera/stream_01”},
      “output”: {
            “url”: {“detected_object_class”: “mqtt://decenter/output_topic_class”},
            “url”: {“detected_object_coordinates”:
“mqtt://decenter/output_topic_coordinates”}
          },
      “ai_model”: {
            “url”: “http://182.252.132.39:5000”,
            “model_name”: “decenter_mnist”, # string
            “model_version”: 0.1, # float
            },
      “autostart”: { “value”: “True” }
}”
```

When the microservice is initiated, it will retrieve video stream from input URL, feed the AI model with individual photograms, and deliver the result to the designated MQTT topic.

- Operation

When it's initiated, the DECENTER AI Package will load the AI model to memory, and it will try to connect to the URL of the video stream, then feed the video input to the MyModle.compute(), which is implemented as an independent thread with shared message queue with DECENTER AI package. The AI computation result will be transmitted to the specified destination topics.

Comparing to the client-server design pattern, this design pattern eliminates HTTP messages between AI service and service client to handle continuous processing of AI computation for each frame. You need to make a request just once for the video stream, and the output will be generated for every frame.

4.4 Implementation of AI service with DECENTER AI package: UC4

To validate the use of the DECENTER AI package, the microservices of UC4 have been implemented and tested. The purpose of UC4 is to validate DECENTER benefits on smart office environment. The AI service on UC4 will provide place information to a person in front of a camera according to the membership of the person to different groups. If the person is a member of a specific group, the schedule or meeting room information will be displayed on the screen. Full description of UC4 can be found in D2.2, and the microservice configuration of UC4 is depicted in Figure 19. It consists of three types of AI microservices (Face Detection, Face Feature Extractor, and X Group Member Verification - MV), one MQTT broker, one model repository server (part of the data management capability of DECENTER), and other two regular microservices (Service Control and Content Serving).

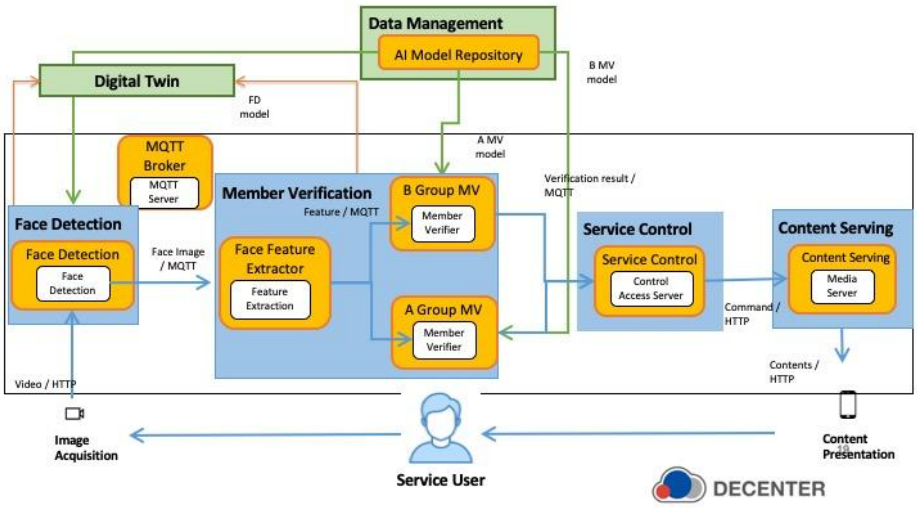


Figure 19 Microservice configuration of UC4, including Digital Twin and Data Management.

According to the application design, there are three types of AI microservice which are cascaded one from another; their interfaces are described in Table 4. UC4 implementation requires total six microservices, four of them are AI microservices. The AI microservices are all built with the DECENTER AI package. The details on UC4 are given on D5.1.

Table 5 Microservice descriptions of UC4

Identifier	Input	Input Type	Output	Output Type	Auto-start
UC4-FD	URL of Camera (HTTP GET)	Video Stream	mqtt://UC4-mqtt/face_image	TYPE_FI	True

UC4-FE	mqtt://UC4-MQTT /face_image	TYPE_FI	mqtt://UC4-MQTT /face_feature	TYPE_FF	True
UC4-MVA	mqtt://UC4-MQTT /face_feature	TYPE_FF	mqtt://UC4-MQTT /member_result	TYPE_MR	True
UC4-MVB	mqtt://UC4-MQTT /face_feature	TYPE_FF	mqtt://UC4-MQTT /member_result	TYPE_MR	True
UC4-SC	mqtt://UC4-MQTT /member_result	TYPE_MR	http://uc4-sc/contents (HTTP POST)	TBD	True

4.5 Remarks

DECENTER AI package has been updated on the second year to provide more intuitive ways to build an AI (micro)service from an AI method. It has now better integration with Docker and Kubernetes, and can deal with multiple protocols for service access. While updating the DECENTER AI Package, we had applied it to various configurations to see whether it is suitable fori providing AI with microservice architecture and were able to define three design patterns which can be exploited for service realisation on Kubernetes-compatible cluster. The other benefit of this package is that it encapsulates data processing along with model itself. With model serving, the service client needs to handle data processing and deliver it to the model serving, which increases network bandwidth for the service. This package receives raw data as an input, thus making service simpler and bandwidth efficient.

This AI package has been applied to AI microservices of UC4 in Y2. On Y3, more integrations with the Platforms scheduled to bring it more cloud- and edge- compatible. Custom Resource Definitions (CRDs) will be added for the configuration of AI service and integration of DECENTER platforms such as AppComposer. Also, more AI microservices will be implemented for the testbed and PoC with DECENTER UCs.

5 Updates on Digital Twin representation

The Digital Twin asset of DECENTER is able to represent concepts, events and situations of real life in a digital replica. The digital replica contains not only existing, real-life events but also non-existing, virtual AI entities. The integration of these heterogeneous sources of information has a great impact on industry. Particularly, it facilitates the monitoring, management, assessment, and decision making during the development and production of a system or product. This section is dedicated to describing the advancements of Digital Twin during the second year, as well as the integration plan with the different use cases.

5.1 Interaction between DT and AI at microservice level

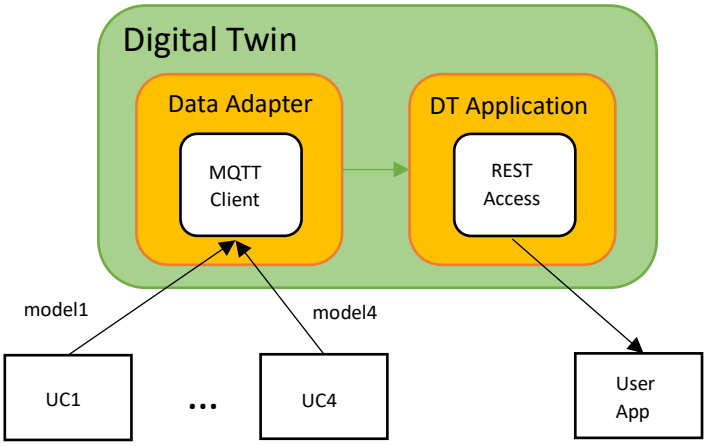


Figure 20 Components of Digital Twin

The DECENTER Digital Twin (DT) application service consists of two components, the *Data Adapter* and the *DT Application.* The main goal of these components is to offer certain common services to AI applications. In particular:

- *Data Adapter* encompasses an *MQTT Client* microservice, which enables the interactions with the UCs. The client subscribes to the MQTT entry point of any AI application, provided by the use cases (UC). After a lightweight configuration (e.g., port, IP) of the client, the Data Adapter retrieves any published information in real time. This information, described in detail at the next section, contains interesting AI features (i.e. inferences outputted by operationalised AI methods) and processed data. The Data Adapter maps this information to an internal data model, which is either given or pre-defined by the UC owners. Finally, the exchange of information respects the JSON syntax.
- *DT Application* provides a REST API in order to facilitate the exploration of Digital Twin's data. A swagger tool is deployed to enable an entry point to the REST API. This API can be used by any third-part application to further exploit or analyse the data. A lightweight configuration (e.g., port, IP) is also needed to achieve this communication.

These two components are deployed at the 'Application Services' layer of DECENTER Architecture (for more details see D2.2).

5.2 Data types of the DT representations

This section describes the data provided by each use case (UC) to the Digital Twin (DT) to build required representations of AI features in their AI applications. This set of data is capable of producing a digital representation of the real-world situation inferred by the AI applications, which is taking place in the use case environment. The set of data is chosen in such a way that it maximizes the situational awareness, but at the same time it does not sacrifice the privacy of the involved actors. It is worth mentioning that the set of defined data—i.e. the DT representation—can be easily enriched, by simply extending the formatted message of the MQTT communication protocol. The DT receives the message and then automatically and dynamically represents the new information within its data model (see Section 5.3 for more details).

5.2.1 UC1: Smart City Crossing Safety

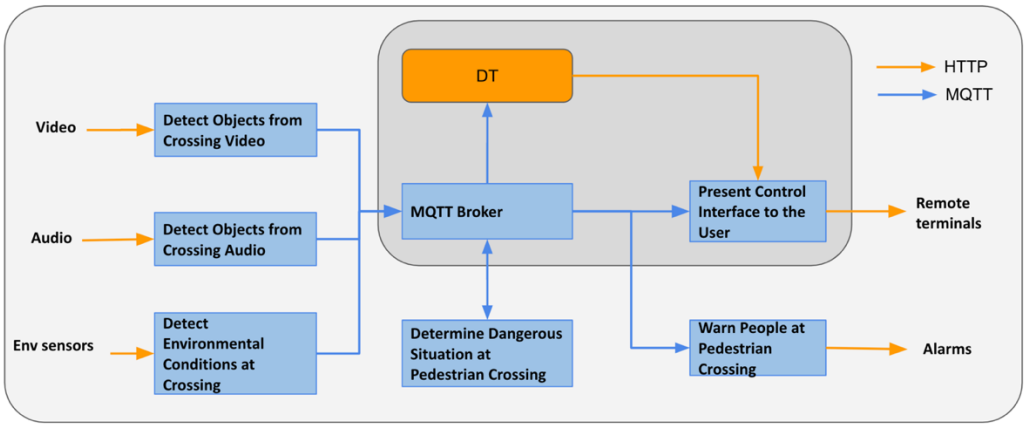


Figure 21 Use of DECENTER Digital Twin application service by UC1 AI application

As shown in Figure 21 the UC1 application is made by several microservices communicating through the MQTT protocol. The output of these microservices can be used for the implementation of a DT representation, in particular the messages related to detected object on the street and the actuation commands sent to give an alert to vehicles and pedestrians. The DT is used to make a representation of the road showing the main actors and the action produced by the UC1 application algorithms.

In this use case we can provide a list of JSON formatted messages that the Digital Twin application can obtain subscribing to these the following topics:

- **Topic:** /event

  Content (json):
  timestamp (number): UNIX timestamp when the event is sent
  event_type (string): type of event occuring at the crossing
  device (string): name of the device used to capture data
  data (object): event specific data

  List of possible event messages:

event_type	device	data
vehicle	cam_south	{ type: “bus”, speed: 55 }
pedestrian	mic_north	{ type: “person” }

weather	weather_station_1	{ temp: 20, hum: 75, bar: 1000, rain: 0 }

The client can subscribe to all events with topic:
event/#
or to a specific device with:
event/cam_south

- **Topic:** /actuation

Content (json):
timestamp (number): UNIX timestamp when device is actuated
event_type (string): type of event occuring at the crossing
device (string): name of the triggered alerting device
data (object): actuation specific data

List of possible actuation messages:

event_type	device	data
alert	vehicle_lights_north	{ duration: 10 }
alert	vehicle_lights_south	{ duration: 10 }
alert	pedestrian_lights	{ duration: 10 }
alert	pedestrian_buzzers	{ duration: 10 }

5.2.2 UC2: Logistics Robotics

Several AI and non-AI microservices collaborate in this use case application; some of them send information to an instance of DECENTER Digital Twin (DT) though an MQTT broker in order to keep an up-to-date representation of the warehouse in robotic intra-logistics scenarios. This representation would include 1) the pose of all robots operating on the warehouse floor and 2) the classification of objects detected through the robots' front cameras.

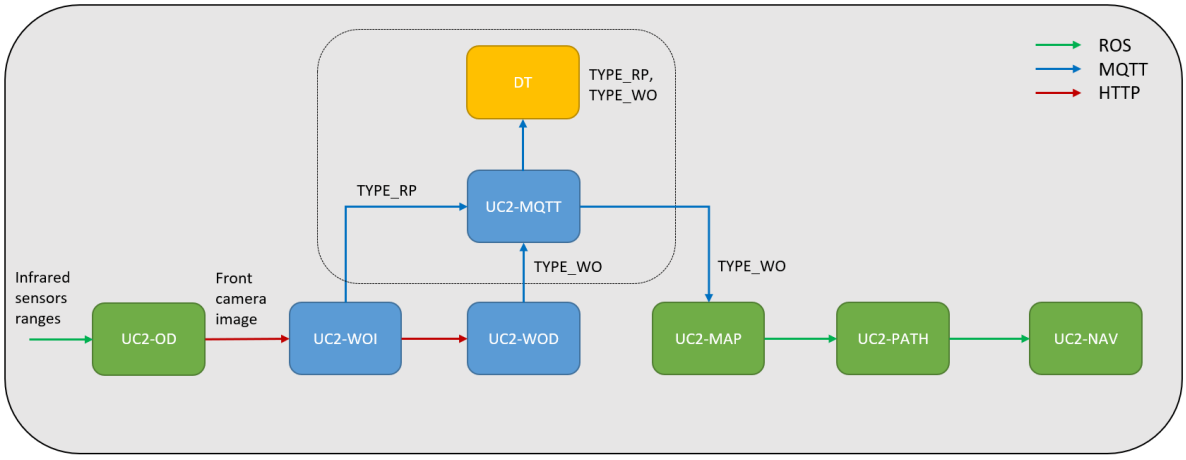


Figure 22 Use of DECENTER Digital Twin application service by UC2 AI application

Specifically, the Warehouse Object Identifier service or UC2-WOI sends the **robot pose** (i.e. position and orientation) periodically. Moreover, the UC2-WOI also sends requests to UC2-WOD to process the robot's front camera images when obstacles are detected in the proximity of the robot by the Range Infrared Sensors. The Warehouse Object Detector service or UC2-WOD (an AI microservice) process those images by means of a Machine Learning (ML) object detection model and sends the description of the **warehouse objects** detected to the DT.

- Topic: /robot
    - Description: This type describes the position and orientation of a robot within the warehouse. UC2-WOI (Warehouse Object Identifier) microservice receives the image captured from the robot's front camera along metadata, such as the location and robot identification. Then, it sends it to the DT.
    - Data Type: TYPE_RP

Name	Type	Example
Timestamp	Datetime (%Y-%m-%d %H:%M:%S)	2020-03-20 03-02-25
Robot identifier	Unit8	1
Position	Float x 3	-87.653274, 41.936172, 0
Orientation	Float x 3	170.1, -39.0, 1.0

- Topic: /object
    - Description: This type describes an object detected within an image captured through a robot's front camera. UC2-WOD (Warehouse Object Detector) microservice received a request from the UC2-WOI to process an image to detect warehouse objects. Then, it sends it to the DT.
    - Data Type: TYPE_WO

Name	Type	Example
Timestamp	Datetime (%Y-%m-%d %H:%M:%S)	2020-03-20 03-02-25
Robot identifier	Unit8	1
Class	String	person
Object position	Unit8 x4	50, 73, 65, 88
Confidence	Float	0.93

    - Topic name: warehouse_object
        - UC2-WOD (Warehouse Object Detection) microservice received a request from the UC2-WODB to process an image to detect warehouse objects.
        - UC2-WOD sends it to the DT.

5.2.3 UC3: Smart and Safe Construction App

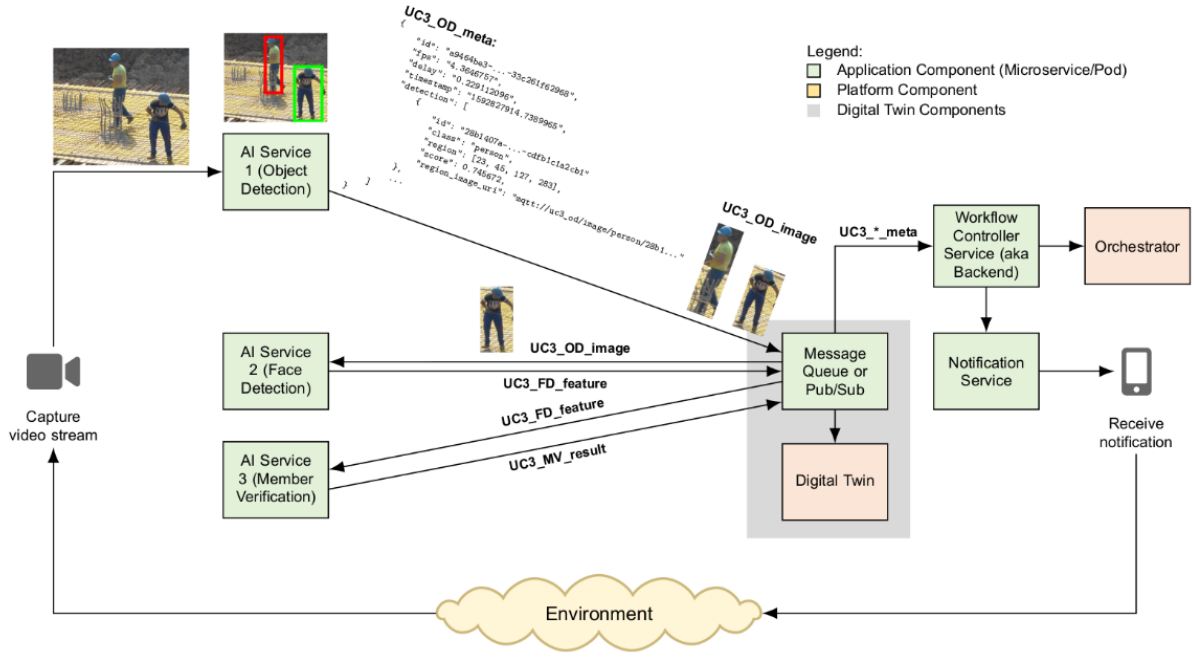


Figure 23: Use of DECENTER Digital Twin application service by UC3 AI application. Services related to digital twin are shaded with grey background

Similar to the other use cases (UC), UC3 also adopts a micro-service architectures style (see Figure 23). In the context of DT representations, of particular interest are components that perform AI inference operations on a video stream, namely AI Service 1, AI Service 2 and AI Service 3, while Message Queue Service and Workflow Controller Service respectively collect and distribute results, and perform control logic based on the result. The communication between the components is loosely coupled, but centrally relied upon Message Queue Service. Since mere exchange of the results is sufficient for the application to function properly, for any potential DT service, if present, it should suffice to read messages coming out of a message queue. In the following we briefly describe the application flow while focusing on the content and format of messages that are exchanged between the components.

A video camera located at a construction site captures physical phenomena and converts it into a continued sequence of digital images, exposed over a network. AI Service 1 intercepts and processes the stream frame by frame. On each frame it performs object detection; objects such as vehicles and persons are detected, which means that for a predetermined list of classes the AI method is able to find rectangular regions in an image and classify the object it perceives within each region. Therefore, in general, depending on the content of a video frame, it can produce several results, shaped into messages of various formats. All the detection results of a single image are formatted into a single JSON, as in the following example:

```
{
	"id": "a9464be3-b446-4531-bd6d-33c261f62968",
	"fps": 4.3646757,
	"delay": 0.229112096,
	"timestamp": 1592827914.7389965,
	"detection": [
		{
			"id": "ec5c6b07-0647-46e1-9818-ad475e10b545",
			"class": "person",
```

```
                "region": [23, 45, 127, 283],
                "score": 0.745672,
                "region_image_uri": "mqtt://uc3_od/image/person/ec5c..."
            },
            ...
        ]
}
```

This message, denoted as type UC3_OD_meta, contains several attributes, as detailed in Table 6 and Table 7. For each such meta message, a new topic in a message queue is generated and is of form: uc3_od/meta/id. Because IDs are dynamically generated by AI Service 1 upon creation of a message, a client can subscribe to a message queue broker with topic string “uc3_od/meta/#”, where “#” denotes wildcard topic. The recipient of this type of messages is Workflow Controller Service, and, possibly, a digital twin service.

Table 6: Description of type UC3_OD_meta

Attribute Name	Description	Type	Example Value
Id	Id of the meta description	UUID	a9464be3-b446-4531-bd6d-33c261f62968
Fps	Current frames per second rate	Float32	4.3646757
Delay	Current delay between frames (i.e. 1 / FPS)	Float32	0.229112096
Timestamp	Message creation time as a Unix timestamp since the epoch (1970-1-1) in UTC	Float64	1592827914.7389965
Detection	Array of objects describing detection results	Array of detection objects	See Table 7

Table 7: Description of detection object within type UC3_OD_meta

Attribute Name	Description	Type	Example Value
Id	Id of the detection	UUID	ec5c6b07-0647-46e1-9818-ad475e10b545
Class	Classification of detected object	String	Person
Region	Bounding box of detected object	Array of uint32 of length 4: upper, left, lower, right	[23, 45, 127, 283]
Score	Confidence of the AI method in the detection	Float32	0.745672
Region_image_uri	URI of extracted image region	URI of form protocol://topic_prefix/image/class/id	mqtt://uc3_od/image/person/ ec5c6b07-0647-46e1-9818-ad475e10b545

Besides the UC3_OD_meta type, possibly, several messages of type UC3_OD_image are generated and published into a message queue broker. For each detection of a particular target class, namely person, a new message is created. In other words, if AI Service 1 in an incoming video frame detects a person, a message of type UC3_OD_image is created. This message contains nothing but the extracted regional image, which enables AI Services 2 and 3 to continue the inference on extracted image parts, that possibly contain the expected content (i.e. on an image is a person and nothing else). However, while the meta message is relatively short, the cropped image is potentially much larger. Therefore, the cropped image is compressed into a JPEG and efficiently packed into a binary data. A multi-dimensional (banded) array is flattened into a one-dimensional array Based on the preliminary experiments, each such extracted regional image is of size up to 15 KB.

5.2.4 UC4: Ambient Intelligence for Office Environments

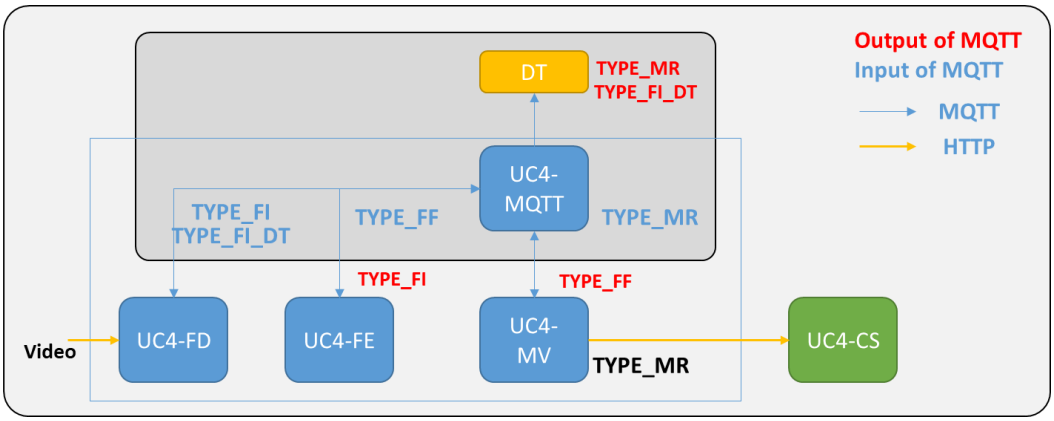


Figure 24 Use of DECENTER Digital Twin application service by UC4 AI application

UC4 integrates several AI microservices. Those AI microservices analyse the input to generate features, and some of the AI features (or inferences) can be used for the implementation of a Digital Twin representation. DECENTER provides a Digital Twin application service to store the features extracted from an AI method, to make it available to be used afterwards. In UC4, two data types are provided to the Digital Twin: TYPE_FI_DT and TYPE_MR. All message bodies are written in JSON format and sent via MQTT.

- TYPE_FI_DT
  - Description: This type describes the image coordinate of the source image where the face is detected. This message type does not include the face image itself to prevent sensitive data transmission.
  - Data Type: TYPE_FI_DT

Name	Type	Example
Timestamp	Datetime (%Y-%m-%d %H:%M:%S)	2020-03-20 03-02-25
Face_position	float16 x4 (Lefttop_x, Lefttop_y, Rightbottom_x, Rightbottom_y)	0.1, 0.2, 0.6, 0.7 (Each value is a coordinate value when the width and height of face image are normalized to 1.)

- Topic name: face_image_dt
    - It contains coordinate of face image and timestamp information.
    - UC4-FD(Face Detection) MS sends it to DT.

- TYPE_MR
    - Description: This type describes the final analysis result of this use case – the membership information. This type includes timestamp to identify to which image this results belongs.
    - Data Type: TYPE_MR

Name	Type	Example
Timestamp	Datetime (%Y-%m-%d %H:%M:%S)	2020-03-20 03-02-25
Result	String (Yes|No)	Yes
GroupName	String	GroupA
Confidence	Float	0.92

- Topic name: member_result
    - It contains member verification result and timestamp information.
    - MV(Member verification) sends it to DT

5.3 Updates of SensiNact to support DT

The modular architecture of sensiNact (Section 3.1.3, D3.3) allows the easy integration of a Digital Twin (DT) application service. The following updates were performed inside sensiNact in order, not only to support DT, but also to adapt to the DECENTER Fog Platform.

- The data model of the DT, initially discussed in Section 6.2 of D4.1, is updated to reflect the data identified by the UCs. Figure 25 illustrates the Digital Twin's data model properly adapted for the needs of UC4. Similar adaptations will be performed for the rest of the UCs.

- The core of sensiNact is updated to support any data model, which can be given at run time. This allows the UCs to define their own model and share it at run time using a simple XML format. An example of this XML is shown below, which describes the feature of Face_position from the microservice TYPE_FI_DT of UC4 (see Section 5.2.4). The XML can be easily extended to support more features and data types.

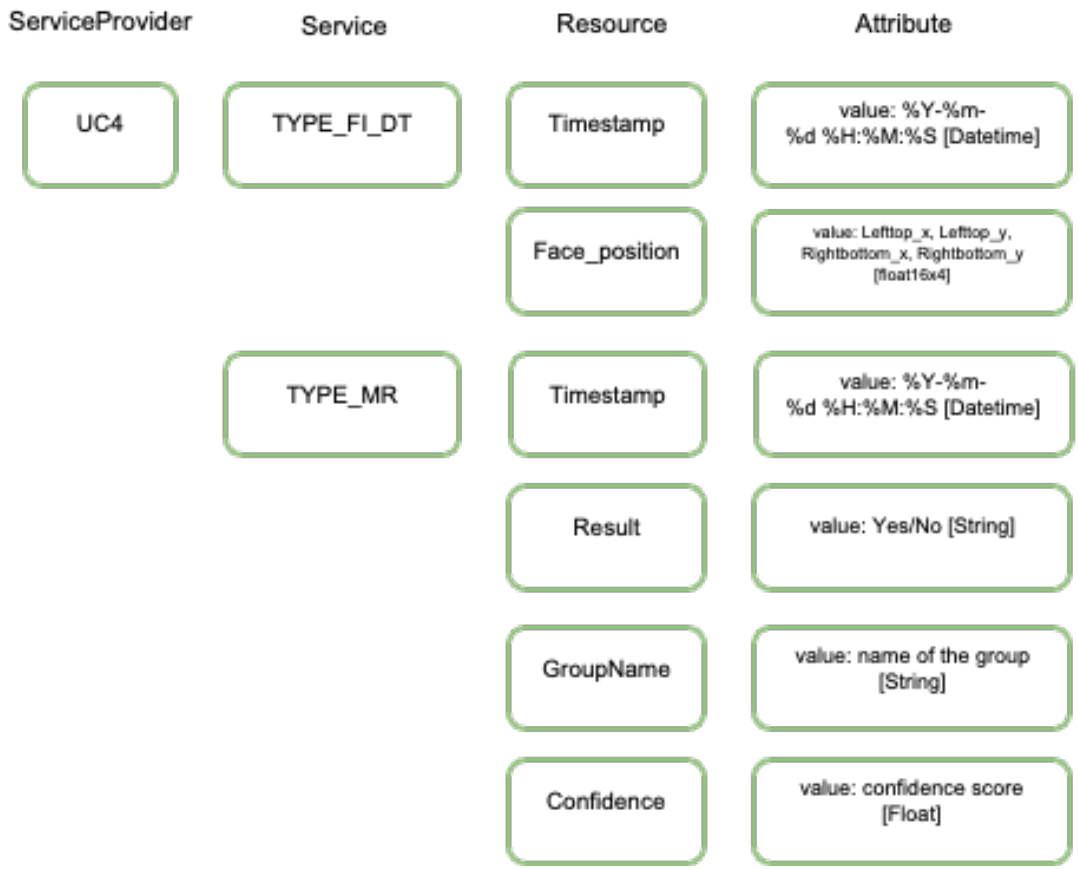


Figure 25 Digital Twin Data Model, UC4

```
<resourceInfo target="TYPE_FI_DT" name="Face_position" xsi:type="resourceInfoSensor">
        <policy name="SENSOR"/>
        <identifier xsi:type="stringContent">Face_position</identifier>
        <attributes>
                <attribute name="value" type="string">
                        <metadata>
                                <meta name="description" type="string">
                                        <value>0, 0, 0, 0</value>
                                </meta>
                        </metadata>
                </attribute>
        </attributes>
</resourceInfo>
```

The XML is particularly useful for offering metadata information, such as the data type (e.g., string) of each feature or the description of its formatted message (e.g., 0, 0, 0, 0). However, the XML is not mandatory for building an appropriate data model. For instance, if the XML is not provided by the UC, sensiNact is able to automatically and dynamically create the DT data model by translating the topic of the MQTT messages. The topic consists of topic levels, each one separated by a forward slash:

sensinact/serviceprovider/UC4/service/TYPE_FI_DT/resource/Face_position

The Digital Twin subscribes to multiple topics simultaneously, by replacing one topic level with a single-level wildcard. The plus symbol represents a single-level wildcard in a topic:

```
sensinact/serviceprovider/+/service/+/resource/+
```

The multiple subscriptions mechanism allows sensiNact to learn the data model dynamically and at run time as data arrive. For example, the arbitrary string placed after the serviceprovider topic level (i.e., UC4) is mapped to the ServiceProvider entity of sensiNact. A similar mapping is performed for the rest of the topic levels and thus the data model is built. Last but not least, regarding the operationalisation of those two services, we have used the containerization tool developed within the eclipse source repository allowing to build dedicated Docker container images. This has allowed us to make sensiNact, along with these Digital Twin components, available as a single application service.

5.4 Remarks

During the second year, we focused on integrating the work of Digital Twin with the UCs. For this purpose, we defined their between interactions in a microservice level, as well as the format and technical details of the exchanged messages. The integration work will be demonstrated using UC4 during the second year's review.

During the next upcoming months, we plan to integrate the data of at least one more UC in the Digital Twin. Moreover, we plan to provide sensiNact Studio (Section 7.2, D3.1) as a microservice for a more intuitive and user-friendly graphical representation of the Digital Twin.

6 Updates on AI solution design

6.1 Machine Learning Workflow

As described in D4.1-Section 2.2 [11], the continuous flow of activities to develop and operate and Machine Learning (ML) model resembles the DevOps approach to software development life cycle. In fact, this flow has started to be known as MLOps, and clearly differentiates between the ML model development phase (in dark blue in Figure 26) and the activities of the Ops phase (light blue).

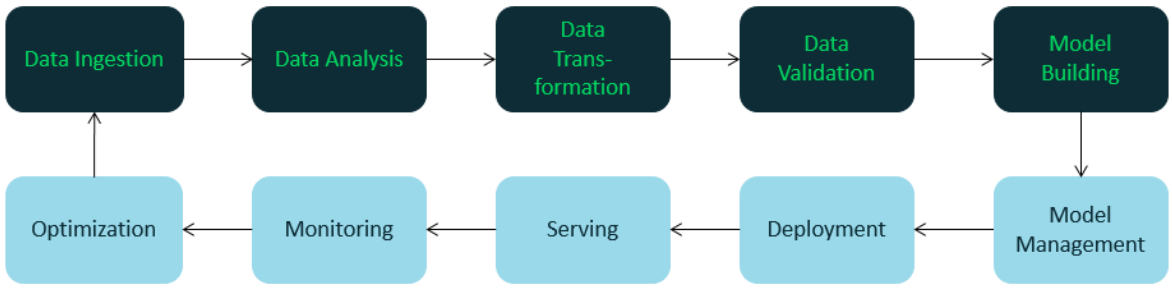


Figure 26 Continuous workflow of activities in the MLOps systems development life cycle. The activities of the ML training phase (in dark blue) are followed by the activities of the ML operations phase (light blue)

Kubeflow

Different ML platforms exists – and continue to appear every month – to give full support to this new MLOps paradigm. One of them is Kubeflow10, a CNCF (Cloud Native Computing foundation)11-backed ML platform for Kubernetes. Actually, Kubeflow offers a series of useful abstractions specifically designed to simplify the use of Kubernetes for MLOps activities; those abstractions are presented as Kubernetes custom resources, so the user just needs to manipulate those rather than the lower-level conventional Kubernetes resources.

In its current **version 1.0, Kubeflow** offers the following three main capabilities:

I. **Method development:** Typically, data scientists or ML method designer create new algorithms, for example new Deep Neural Networks, to address a specific problem. A central part of this process is the preparation of datasets for training, validation and testing, which normally takes the method creator through a series of steps, from data ingestion and exploration (analysis) to data transformation and validation to obtain suitable datasets for the new ML method. Together with the datasets preparation, the method designer provides an interactive development environment based on Jupyter Notebooks to specify (program), validate and test the ML pipeline – i.e. sequence of data processing tasks – which comprise the method. It also provides facilities to execute those notebooks in different runtime configurations.

II. **Model building**: Kubeflow gives support to ML engineers to operationalise model building pipelines in a reliable, cost-efficient and scalable way. To do so, it offers a series of so-called training operators which let the engineer to use different distributed or centralised model training frameworks.

[10] https://www.kubeflow.org/
[11] https://www.cncf.io/

III. **Model serving**: Once the model is built and ready for production, the ML engineers needs to undergo a series of activities to operationalize the execution of ML model or model inference (see MLOps workflow shown in Figure 26): from configuration management of the model (including versioning and storage) and deployment following different roll-out strategies, to the actual model execution (known as model or inference serving) and performance monitoring. In Kubeflow, this capability is officially provided by either KFServing[12] or Seldon Core Serving[13]. **KFServing** and **Seldon Core** are considered *multi-framework model serving systems*[14] because:

A. They accept models to be run on many different ML frameworks: e.g. TensorFlow, PyTorch, Sckit-learn, XGBoost, ONNX, TensorRT, etc. This is radically different from conventional approaches to model serving which usually are framework specific. For example, Tensorflow Serving or TensorRT Inference Server are considered a *standalone model serving systems,* centred on running models for Tensorflow and TensorRT frameworks respectively.

B. They offer already a series of complex features for a whole coverage of the model serving requirements in different categories:

1. ***Frameworks*** *supported to run models on them.*
2. ***Graphs.*** Chain of model executions or inferences to achieve a composite (e.g. ensemble) of inference services.
3. ***Analytics.*** Model performance monitoring and analysis.
4. ***Scaling.*** Elastic autoscaling of the model server to meet SLOs.
5. ***Custom.*** Resources to facilitate the integration and packaging (i.e. containerization) of the model to be deployed.
6. ***Rollout.*** *Support for different strategies for the actual deployment of model servers, given your requirements of availability and performance (i.e. execution time).*
7. ***Management.*** Model versioning and lifecycle management, including retirement (due to obsolescence or underperformance) and deciding when to trigger new optimizations and when and how (i.e. the best rollout strategy) to deploy them.

KFServing

KFServing enables serverless inference serving on Kubernetes and provides high abstraction interfaces for common ML frameworks like TensorFlow, XGBoost, scikit-learn, PyTorch, and ONNX to solve production model serving use cases. Its technology stack is based on the serverless technology stack of Kubernetes, with Istio and KNative on top of Kubernetes core services (see Figure 27). Some highlights:

- Provide a Kubernetes custom Resource Definition for serving ML models on arbitrary frameworks, which add enormous simplicity to the operationalization of ML models.

[12] https://www.kubeflow.org/docs/components/serving/kfserving/
[13] https://www.kubeflow.org/docs/components/serving/seldon/
[14] https://www.kubeflow.org/docs/components/serving/overview

- Self-management of autoscaling, networking, health checking, and server configuration to bring cutting edge serving features like GPU autoscaling, scale to zero, and canary rollouts to your ML deployments.
- Enable a simple, pluggable, and complete pipeline for your ML inference server by providing a framework for **pre-processing, providing prediction, post-processing and explainability out of the box**.

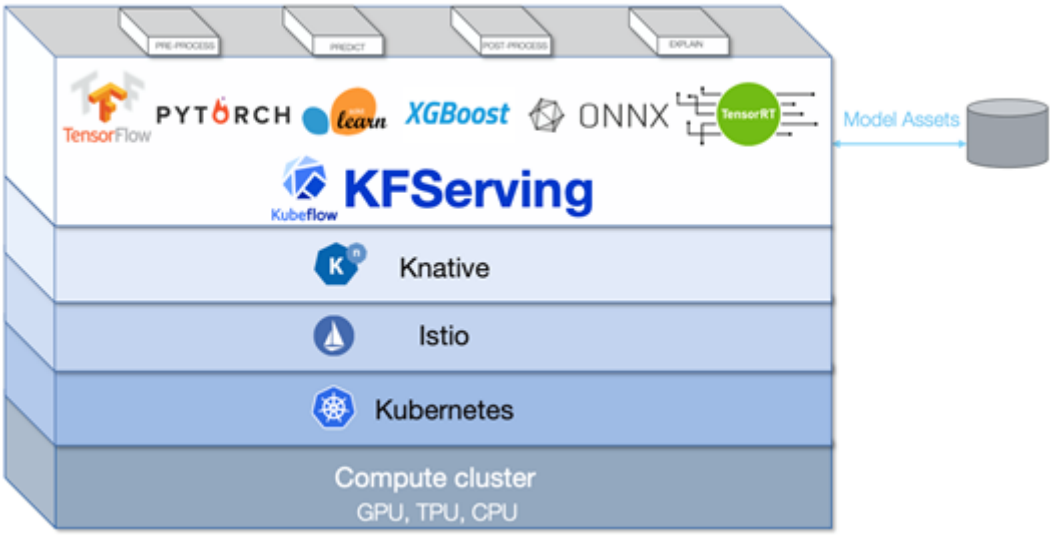


Figure 27 Technology stack of KFServing, the supported multi-framework model serving system of Kubeflow (extracted from KFServing GitHub Repository[15])

KFserving is officially developed with Kubeflow, it has a strong community of contributions that help KFServing to grow in features. Its technical Steering Committee driven by Google, IBM, Microsoft, Seldon, and Bloomberg.

DECENTER multi-framework model serving system

Seldon Core comes installed with Kubeflow 1.0; it is not part of the Kubeflow project ecosystem, but it is supported within Kubeflow.

The containerization of AI methods supported by DECENTER, and specifically the set of software tools and resources to do it that have been developed within WP4, could aim at becoming a **DECENTER multi-framework model serving system** (a.k.a DCServing) similar to KFServing and Seldon Core, but focused on decentralized ML scenarios. In order to assess the viability of the concept of DCServing, we have compared what we have already with the features and sub-features supported by those two systems, and that are used to compare them already[16].

[15] https://github.com/kubeflow/kfserving
[16] https://www.kubeflow.org/docs/components/serving/overview/

Table 8 Comparison between multi-framework model serving systems: the two officially supported by Kubeflow (KFServing and Seldon Core) and the envisioned DECENTER Serving (DCServing).

Feature	Sub-feature	KFServing	Seldon Core	DCServing
Framework	TensorFlow	✓	✓	✓
	XGBoost	✓	✓	
	scikit-learn	✓	✓	
	TensorRT	✓	✓	
	ONNX	✓		
	PyTorch	✓	✓	
Inter-pod Graph (optimization)	Transformers	✓	✓	
	Combiners	Roadmap	✓	
	Routers including test A/B and MAB[17]	Roadmap	✓	
	Splitters			✓
Intra-pod Graph (optimization)[18]	Plugable model server pipeline	✓		✓
Analytics	Explanations	✓	✓	
	Monitoring	✓	✓	
Scaling	Knative (serverless)	✓		
	GPU AutoScaling	✓		
	HPA[19]	✓	✓	
Custom	Container	✓	✓	✓
	Language Wrappers		✓	
	Multi-Container		✓	✓
Rollout	Canary	✓	✓	
	Shadow		✓	
Service mesh[20]	Istio Connect	✓	✓	
	Istio Secure	✓	✓	
	Istio Control	✓	✓	
	Istio Observe	✓	✓	
Management	Versioning			✓

[17] MAB (Multi-armed bandit) https://en.wikipedia.org/wiki/Multi-armed_bandit
[18] Pluggable, and complete pipeline for model serving through a framework for pre-processing, prediction, post-processing and explainability out of the box.
[19] HPA (Horizontal Pod Autoscaler) https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
[20] https://istio.io/docs/concepts/what-is-istio/

	Updater			✓

The result of the evaluation is that the concept of DCServing is far from covering most of the (sub)features already covered by KFServing and Seldon Core. However, it would provide some unique features in comparison with those two:

- Splitters: ability to split a deep neural networks (DNN) into several containerized sets of layers; it means dividing the ML pipeline captured by a single DNN to be deployed in different microservices, which will the invoked sequentially at runtime.
- Model versioning: ability to understand and work with the concept of version of models).
- Model updater: ability of the model server to decide when it is the best moment and/or required to update the ML model, due to reasons such as obsolescence or underperformance.

Next steps

We see value in continue exploring the concept of DCServing during the last year of the project, mainly to align our results in terms of ***optimization and containerization of AI methods*** with Kubeflow, conceptually and ideally also technologically (since DECENTER platform is based on Kubernetes as well). Kubeflow project is expected to become a strong reference among ML researchers and practitioners in the upcoming years.

During the third year, we will identify the minimum set of features it would make DCServing "complete" from the Kubeflow user's point of view but in the context of decentralized AI scenarios and use cases. And we will try to bridge that gap by developing the (sub)features most needed in that context.

6.2 AI Solution Design

6.2.1 AI Application reference architecture

In this section, we describe the DECENTER Reference Architecture for AI applications. An initial version was delivered in D4.1; in that deliverable, we described five types of AI application architecture building blocks (from D4.1):

Table 9 Solution building blocks of AI

Solution building block type	Description
AI Model	A trained model (either a deep neural network or other ML).
AI Method	An entity serving the inference of a trained model (runs the model on the ML engine).
AI Service	An entity serving a certain AI function (model serving layer) based in an AI Method.

AI App	An entity invoking AI Methods to provide AI functionalities to the end user through a GUI on a responsive and/or mobile app.
AI Application Service	Set of AI functionalities closely related comprising a software application or product function and delivered as a service to an end user.

In MS11 (M18), we presented an enhancement of that reference architecture which added a new architecture view: the so-called AI application process. This describes an AI Application as a sequence of AI Application Services which serve functions of five different kinds:

Table 10 Functionalities related to AI service

AI application service kind	Description
Sensing	Data collection, monitoring
Analysis	Also known as perception in AI systems, may incorporate attention, data/sensor, image segmentation, object classification, object localization, object detection, scene classification, scene interpretation, etc. Typically, high volume of unstructured data to be processed through Deep Neural Network. Descriptive advanced analytics with machine learning methods.
Decision	Knowledge representation and reasoning, search / optimization, planning / scheduling, behaviour selection (reactive planning). Prescriptive machine learning, e.g. reinforcement learning
Action	Behaviour (decision) enaction. Control of the course of action or sequence of actuation (e.g. moto) commands.
Effecting	Execute a command (e.g. motor). Also means “actuation,” to activate, or to put into motion; to animate.

Each kind of AI application service belongs to a specific step in the so-called adaptive control loop of an AI application - as defined by DECENTER reference architecture:

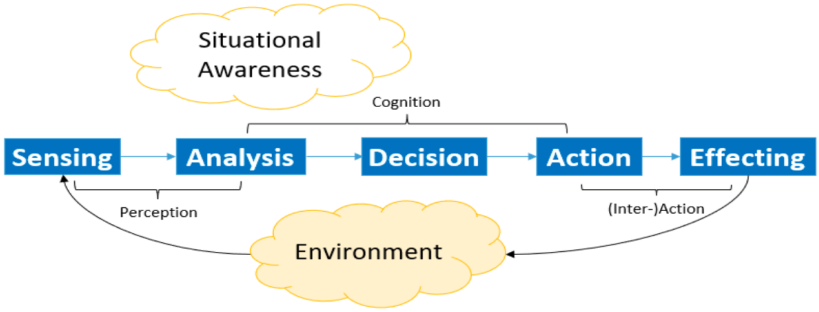


Figure 28 Adaptive behaviour control loop of an intelligence (adaptive) system; is has been taken as the reference AI process model in DECENTER

As explained in D4.1, the application services break down into finer-grained functional blocks which follow the microservices architecture style. Thus, for example, we could have one, two or more microservices collaborating to realise a given Application Service. It is important to notice that those microservices can organize themselves in directed acyclic graphs of computing tasks to serve requests while maintain the required AI Application Service's Quality of Service (QoS) – typically described in the form of Service Level Objectives (SLO) (for a detailed description, see D2.2).

We call AI services to those microservices that carry out an AI/ML task, for example, to run an inference (e.g. forecast, prediction, clarification, etc.). Included in that type of microservice, there would be a model (inference) server exposing API interfaces to call.

When those microservices carry out AI/ML tasks or functions, we call them AI (micro)services; these run a ML model for a given inference (e.g. forecast, prediction, clarification, etc.). To achieve that, those microservices are developed using certain library or frameworks providing already much of the functionality needed for model serving. In DECENTER, we propose the use of our DECENTER AI package, a python library which facilitates the development of decentralized AI applications. But other solutions can go from the conventional model serving libraries provided by the most popular ML frameworks (e.g. Tensorflow Serving or TensorRT Inference Server) or the more advanced *multi-framework model serving systems*, such as KFServing. See Section 6.1 for a complete discussion on them.

Last but not least, the DECENTER Reference Architecture for AI applications fully supports the kind of multi-tier deployment which is typical of decentralized AI applications—as targeted by the DECENTER project (for more details, see D2.2 and D3.3). The DECENTER platform is able to make the decision on where to deploy and run the microservices comprising an AI application service, which would be rolled out on Kubernetes (a container orchestration platform) through a Kubernetes Deployment and exposed to consuming services through a Kubernetes Service. The following section explains how DECENTER solves the problem of containerising AI microservices.

6.2.2 Level of containerization

An AI microservice relies on various software libraries, and AI solution should provide a good reference which to be containerized to the AI microservice container. In Y2, we were able to identify the hardware characteristics and software stacks that AI microservices would require in the context of DECENTER, and also provide methods to help develop and build containerised AI microservices for given AI Methods. The guidelines to design an AI solution are described in this section.

The software stack for AI microservice to be run on computing acceleration hardware is depicted in Figure 29. Though Nvidia case is used for the figure, it can be applied to other acceleration devices without loss of generality. For example, it needs appropriate AMD device drivers and compatible OpenCL (equivalent to CUDA of Nvidia stack) to use AMD GPU for AI application (see Section 4.1 for more details).

Therefore, we need to take that into consideration when building the software stack to containerize an AI microservice. Basically, the AI microservice has dependencies over other software modules, from the deep learning framework (e.g. TensorFlow) to the device driver (e.g. Nvidia device driver), and there is a trade-off between level of containerization. That is, if we put only the AI microservice on the container while leaving all the other software modules on the host, we can build a lightweight container. However, this adds the complexity that dependencies between multiple software modules needs to be managed by the host. Suppose

that there is one AI container that uses CUDA 10.0, and another that uses CUDA 11.0; in this case, the host device needs to manage multiple versions of the same CUDA libraries. By the contrary, if we put (build) all the things into the AI microservice container, the isolation of runtime environments is directly achieved and so no need to manage dependencies to multiples versions of software modules on the host. The downside is that the size of the container can grow significantly, which might cause delays in its roll-out and even the impossibility to deploy in resource-constrained hosts such as edge computing nodes.

DECENTER proposes the approach that containerizes the software stack from the acceleration library up into the AI microservice container. Even though the size of the increases, this approach has a few benefits from the viewpoint of resource orchestration: if we leave the acceleration libraries and deep learning framework on the host, then only containers which have compatible dependencies can be deployed on that host, which can degrade resource usage enormously, Moreover, it requires operations (or platform) teams to manage libraries with different versions and use additional labelling for the compute nodes, which makes resource orchestration considerably more complex. However, by including those software modules (software stack layers) in the container, the resources on the node can be characterised with a minimum number of labels, and many different containerised AI microservices can make use of the node resources in a more flexible way.

The benefits coming from using this approach have been validated within the DECENTER use cases, whose AI microservices use different versions of CUDA and TensorFlow on top of various Nvidia GPUs. The AI microservices can be deployed onto those GPU-enabled nodes with just the consideration of the device driver version. However, it is worth mentioning that this is not a hard constraint on AI microservice's software stack containerization. For example, when not many types of AI methods exist in your application and many of them use the same (latest) version of CUDA, it could be more efficient to containerize the AI microservice including just the AI method logic while installing all its dependencies on the host.

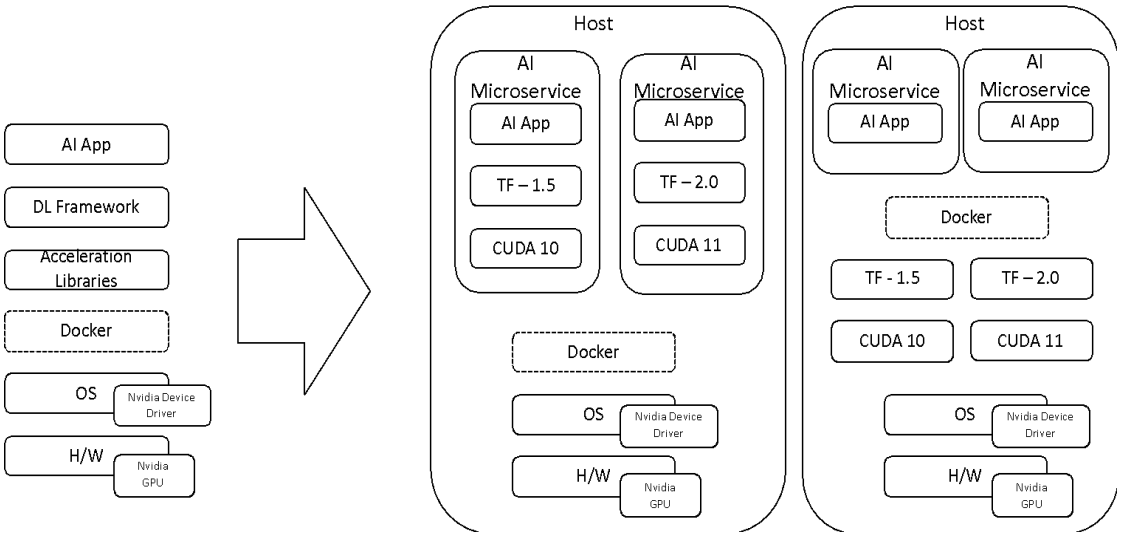


Figure 29 Comparison of containerization of an AI microservice software stack. Containers including all the software stack have benefits on effective virtualization (e.g. environment isolation), with the cons of a larger size

6.2.3 Interaction with Microservices

AI Service has been defined by DECENTER as a microservice serving an AI/ML model (see Section 4.1. As any other microservice, it needs communication means to interact with other (consuming and consumed) microservices. The DECENTER AI package let developers create interfaces for this interaction with less effort by separating network interface logic from the AI method logic to be used as is by the AI developer.

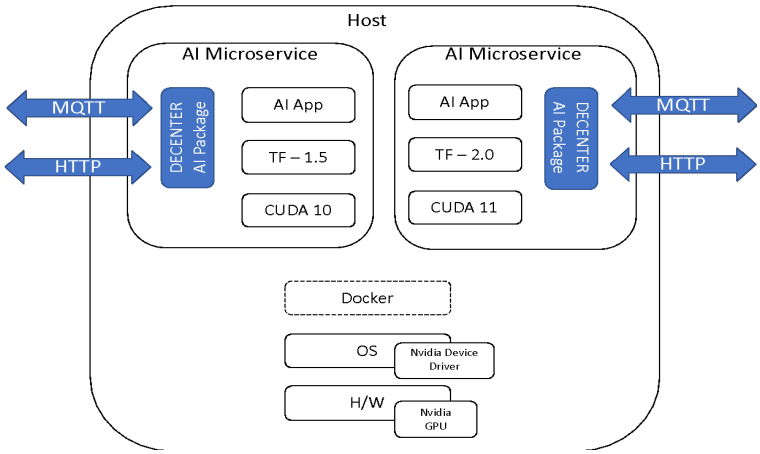


Figure 30 Example deployment of AI microservices with DECENTER

DECENTER AI package provides two kinds of network interfaces, HTTP and MQTT (See Figure 30). These two interfaces enable synchronous and asynchronous interaction with other microservices with respectively a client-server and a publish-subscribe architecture. The HTTP interface lets configure and operate the AI methods inside a container, while the MQTT interface lets operate the AI method in response to data (event data or data stream) generated by IoT devices. For a complete description go to Section 4.2; in particular, the definition of APIs, in Table 4.

6.3 Use case AI application description

This section presents the architecture of each DECENTER use case AI application, which follow the DECENTER AI application reference architecture (presented in Section 6.2.1). These descriptions show the main building blocks of an AI Application, the so-called Application Services, how they chain to run an adaptive feedback control loop, and how they are realised in containerised microservices. Some of these microservices being AI microservices, which means that serve and AI method. Finally, for each use case, a deployment architecture view describes on the possibilities (as an example) of how those microservices can be deployed (scheduled) by the DECENTER platform along the cloud-to-edge (compute) continuum (tiers)

6.3.1 Smart City Crossing Safety (UC1)

- Name: Crossing Safety
- Application domain: Smart City
- Description: This application identifies potentially dangerous situations cyclists or pedestrians crossing the street, e.g. a fast car approaching, noise, reduced view due to rain, fog, snow and obstacles, and generates the right alert. Some constraints are:
    - Recognize that a person is in danger with good accuracy and in the shortest possible time,

- when a dangerous situation is detected it is important to trigger alert in less than 200 milliseconds, and
- privacy of the persons involved must be guaranteed.

Crossing Safety - Application Services

#	AI Application	AI application service	Activity kind	Function
1	Crossing Safety	**Detect Environmental Conditions at Pedestrian Crossing**	Analysis	Detection of detailed environmental conditions (e.g. day/night, visibility, temperature, humidity, frost, etc.) in a pedestrian crossing from nearby IoT sensors
2	Crossing Safety	**Detect Objects from Pedestrian Crossing Video**	Analysis	Detect key objects (e.g. pedestrian, cyclist, vehicle, etc.) approaching the pedestrian crossing from video signal
3	Crossing Safety	**Detect Objects from Pedestrian Crossing Audio**	Analysis	Detect key objects (e.g. pedestrian, cyclist, vehicle, etc.) approaching the pedestrian crossing from audio signal
4	Crossing Safety	**Determine Dangerous Situation at Pedestrian Crossing**	Decision	Rule-based inference to determine dangerous situations given environmental conditions and the presence of certain objects and people.
5	Crossing Safety	**Warn People at Pedestrian Crossing**	Action	In case of a dangerous situation, send of sound and light signals to the people involved
6	Crossing Safety	**Present Control Interface to System Admins**	Action	A control interface (it may be a terminal interface) for the application or system administrator.

Crossing Safety - Architecture Views

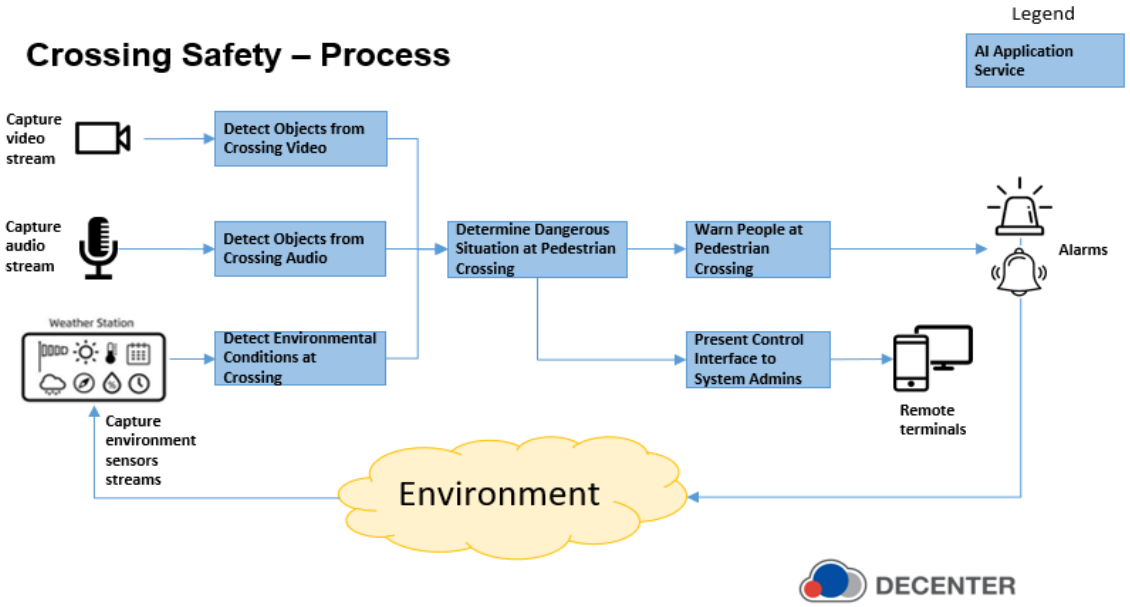


Figure 31 Process view of UC1 AI Application

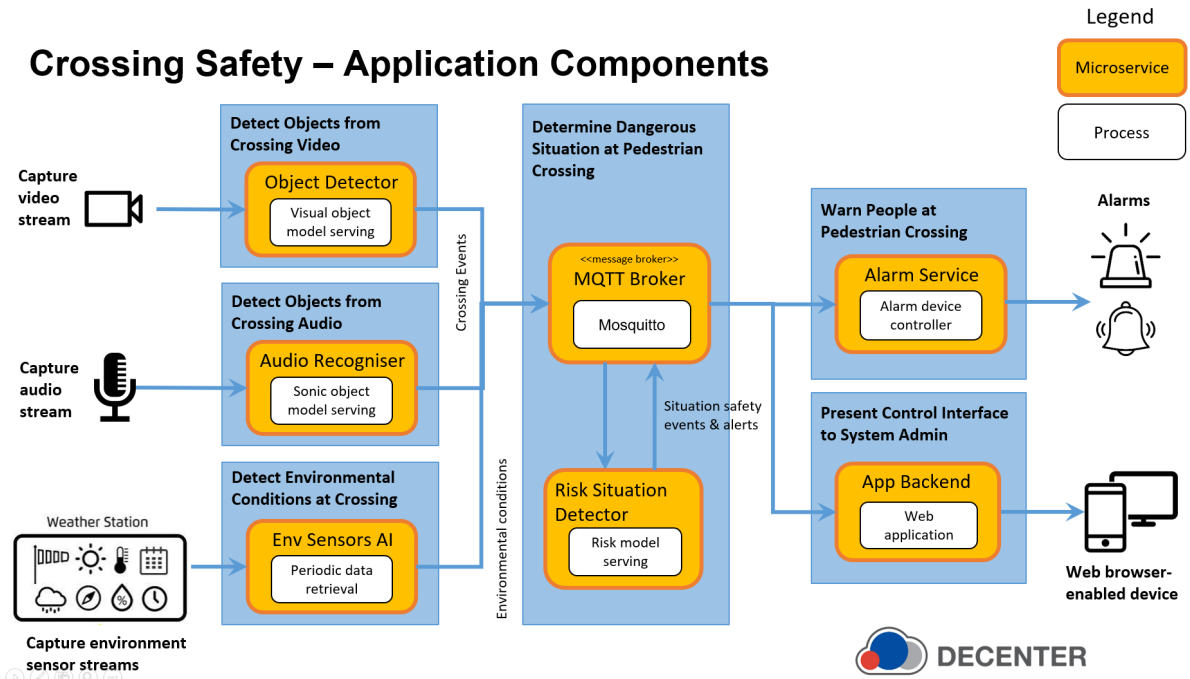


Figure 32 The Application Components view shows the control flow between microservices (enclosed in each Application Service), in UC1

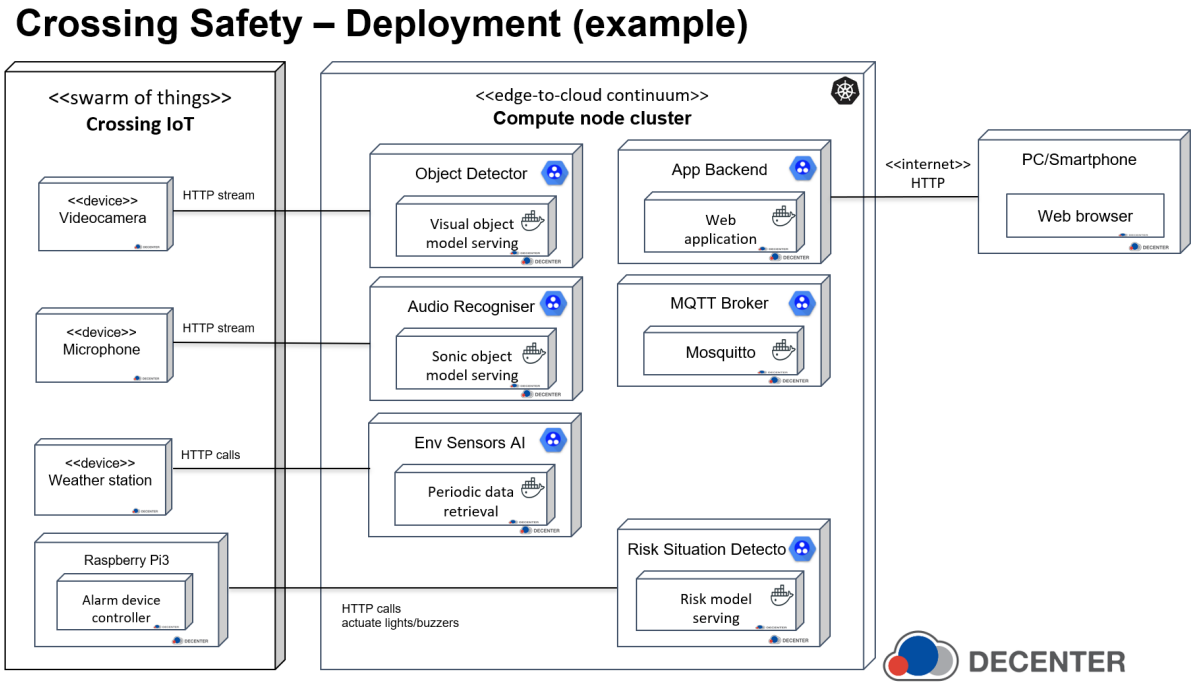


Figure 33 Example of deployment of UC1 containerised microservices on Kubernetes-based Fog Platform

6.3.2 Logistics Robotics Optimization (UC2)

- Name: Logistics Optimization
- Application domain: Logistics Robotics
- Description: This application drives logistics robots on a warehouse floor shared with humans as well as other robots, providing them with an updated "free" path to reach a given location:
  - It maintains an updated map of the warehouse, including static and dynamic objects encountered by the robots in their navigation, typically other robots and humans, and
  - It provides alternative paths to robots to reach their destination when they encounter an obstacle in their path, depending on whether this obstacle is another robot, a human, or anything else.

Logistics Optimization - Application Services

#	AI Application	AI application service	Activity kind	Function
1	Logistics Optimization	**Detect Obstacles on Path**	Analysis	Detection of presence in front of the robot with on-board laser-based distance sensors.
2	Logistics Optimization	**Identify Objects on Path**	Analysis	Location of persons and robots in an image captured with the robot's front camera pointing towards the robot's path. It may make use of the readings from on-board laser-based distance sensors.

3	Logistics Optimization	**Fleet Management System**	Decision	Update the warehouse global map by overlapping static information and dynamic objects information detected and identified by the robots. Plan the navigation path for each robot of the fleet, based on that global map.
4	Logistics Optimization	**Navigate to Location**	Action	The robots use the navigation path generated for it by the Fleet Management System to go to a given location in the warehouse. It implies sending commands to its actuators (e.g. motors).
5	Logistics Optimization	**Pick/Leave a Rack**	Action	Pick or leave a rack (weight) from/at a given location set by the robot task as the starting or ending location of its navigation path. It implies sending commands to its motors.

Logistics Optimization - Architecture Views

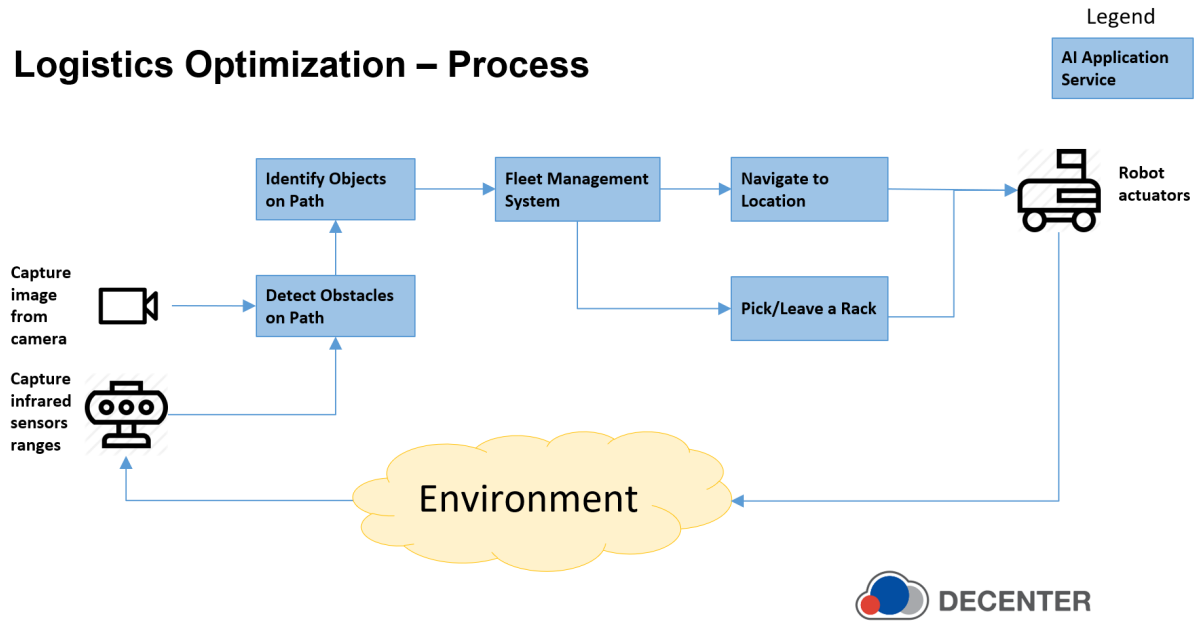


Figure 34 Process view of UC2 AI Application

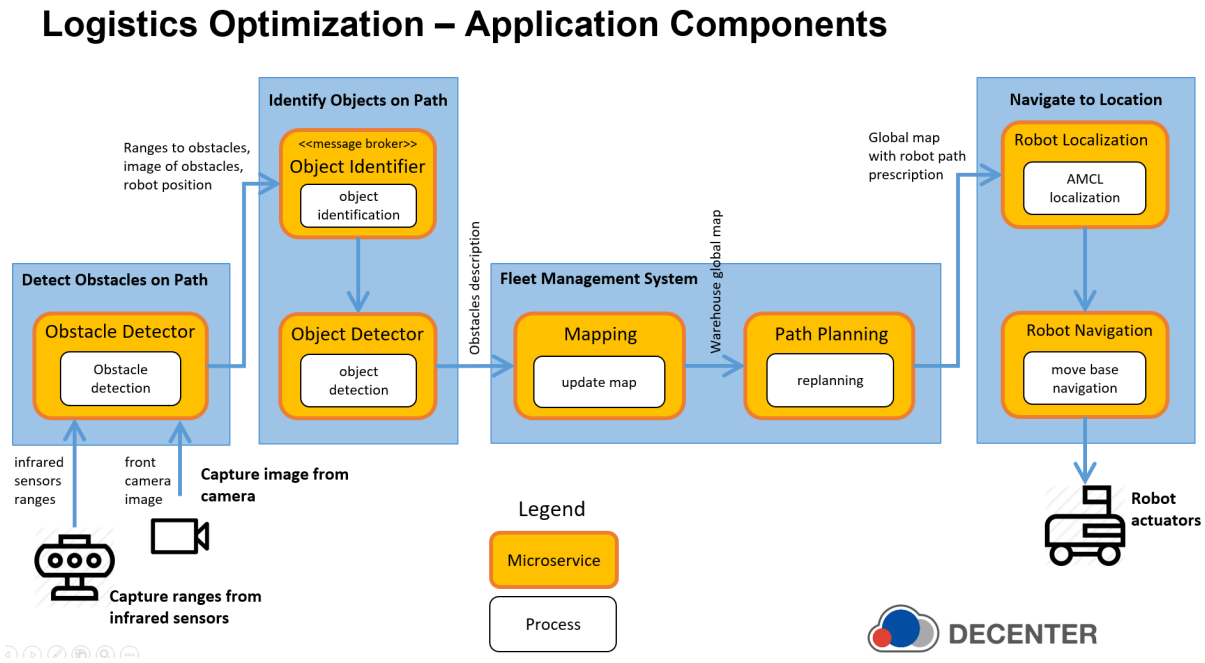


Figure 35 The Application Components view shows the control flow between microservices (enclosed in each Application Service), in UC2

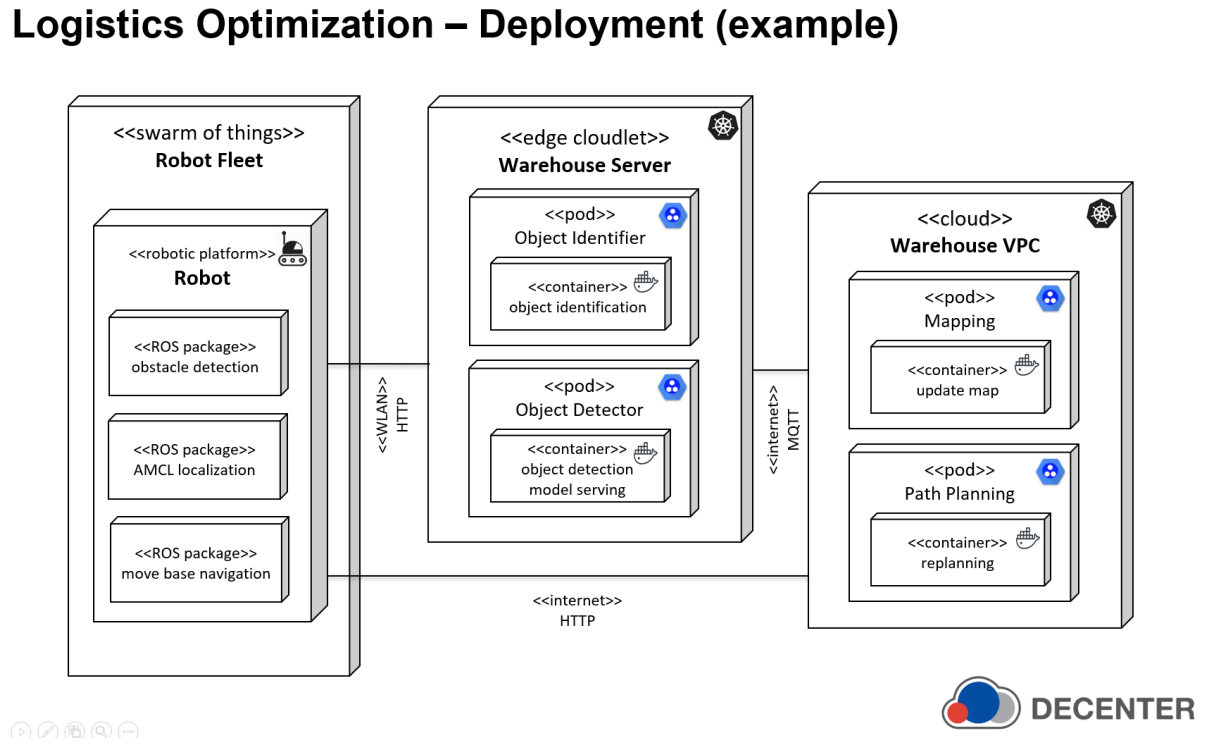


Figure 36 Example of deployment of UC2 containerised microservices on Kubernetes-based Fog Platform

6.3.3 Smart and Safe Construction (UC3)

- Name: Smart and Safe Construction
- Application domain: Smart Construction
- Description: This application improves safety at construction sites by issuing notifications to the construction site manager, when a safety violation is detected. In particular, the application is supposed to (1) object detection and identification (i.e. vehicle detection) and (2) member verification (i.e. person detection and group membership identification). More specifically:
    - Provide object recognition precision of at least 70%.
    - Detect objects within 30 seconds. In particular, the system must detect a person and verify if the person has access to the construction site; in a distance of at least 4.0 m (the width of manipulation intervention road) with a presumption that average walking speed is 1.4m /s.
    - Maintain privacy of information processed on each construction site
    - Maintain high reliability and operate even if a specific Fog Node fails to respond
    - Perform correctly in different temperature/illumination conditions.

Smart and Safe Construction - Application Services

#	AI Application	AI application service	Activity kind	Function
1	Safety at Work	**Object Detector**	Analysis	Detects objects (i.e. vehicles and people) from the camera video stream. When it detects an object, it triggers the deployment of AI model that facilitates face recognition and member verification.
2	Safety at Work	**Face Recognizer**	Analysis	Extracts a feature vector from the detected face image. Each feature vector contains discriminative face representations. Utilizes the face feature vector to recognize if the face belongs to a member of a group or not. The output includes a confidence score.
3	Safety at Work	**Notification Management**	Decision/ Action	If the detected person does not belong to the member group, a notification is sent to the construction site manager. Also, in case of a vehicle detection, a notification is sent to the construction site manager.

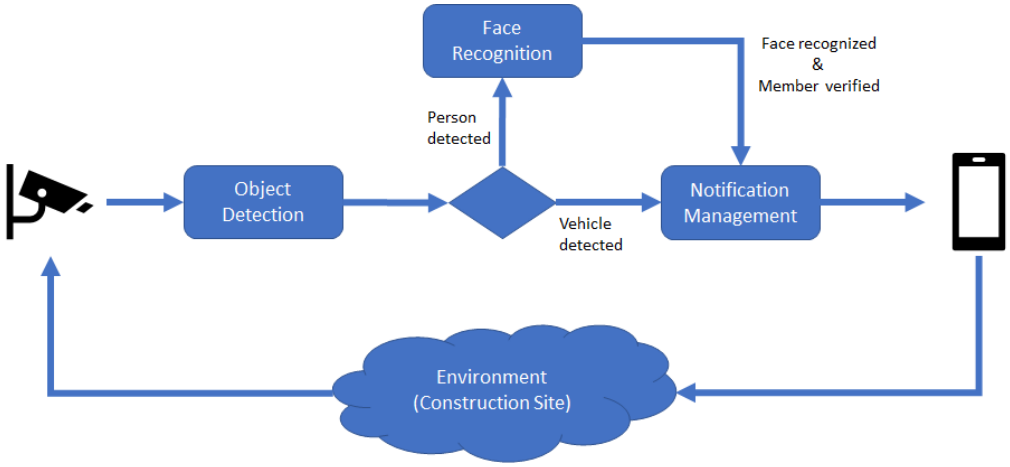


Figure 37 Process and data flow of UC3

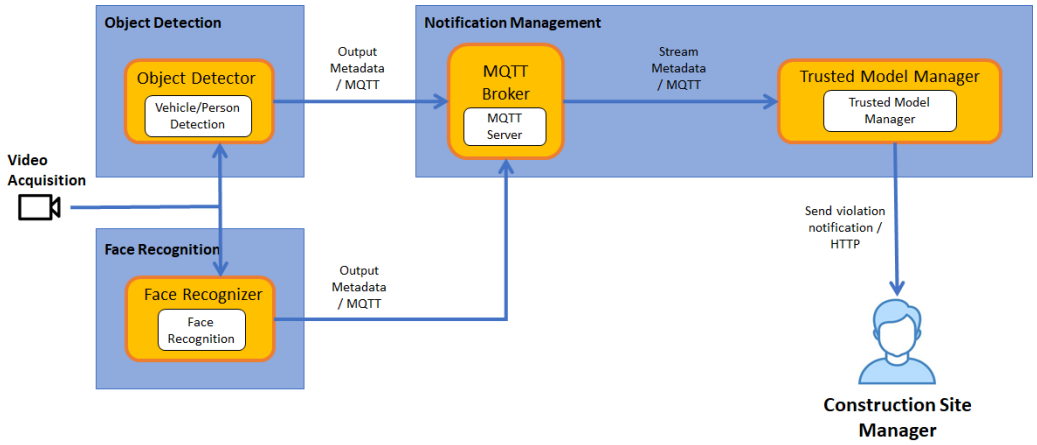


Figure 38 Process view of AI microservices in UC3

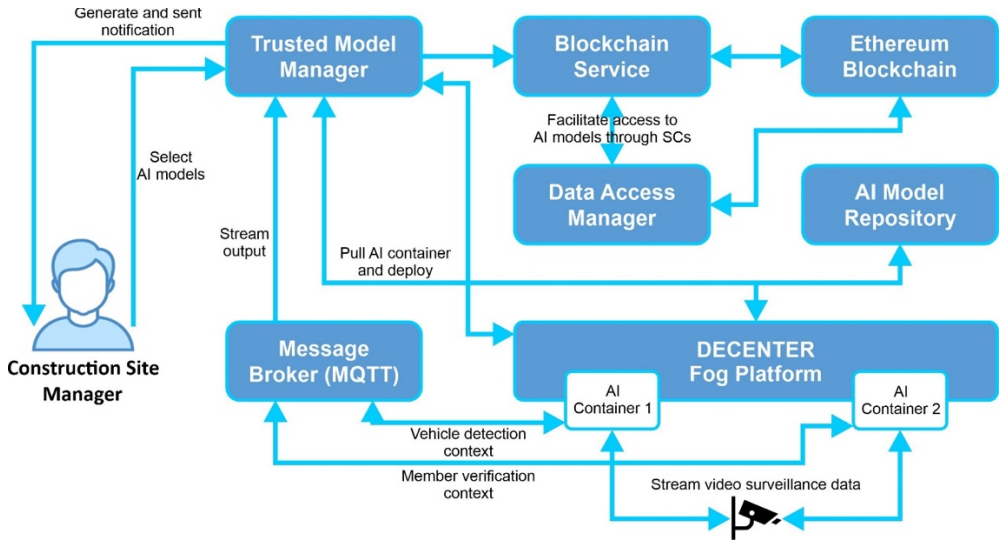


Figure 39 Application components view shows the control flow between microservices in UC3. The consecutive steps of the workflow are described in D2.2

6.3.4 *Ambient Intelligence for Office Environments (UC4)*

- Name: Ambient intelligence
- Application domain: Smart office
- Description: This application displays a proper content according to the member verification result, when a person comes in front of the camera.
    - It integrated two verifiers that can verify two group members: A group verifier, and B group verifier. We assume that only these two groups are targeting to see specific content.
    - The deployment makes proper use of edge and cloud resources to ensure user face image and model privacy. Thus, the processes at the edge can verify whether the visitor of a certain space can consume certain content or not in that space, without sharing personal information with the cloud.

Ambient Intelligence - Application Services

#	AI Application	AI application service	Activity kind	Function
1	Ambient intelligence	Face Detection	**Analysis**	Detects a face in the image from camera (UC4-FD). When it detects the face, it crops face area and transfers it to UC4-FE.
2	Ambient intelligence	Member Verification	**Analysis**	First, extracts a feature vector from the detected face image (Facial Feature Extractor microservice or UC4-FE). Each feature vector contains discriminative face representations (i.e. facial features). Second, the facial feature vector is used to determine whether the face belongs to a certain member group or not (Member Verifier microservice or UC4_MV). The output includes a confidence score.
3	Ambient intelligence	Contents Service Management	**Decision / Action**	Decides what content to provide and serve the appropriate content to the service front end (Content Serving microservice or UC4-CS).

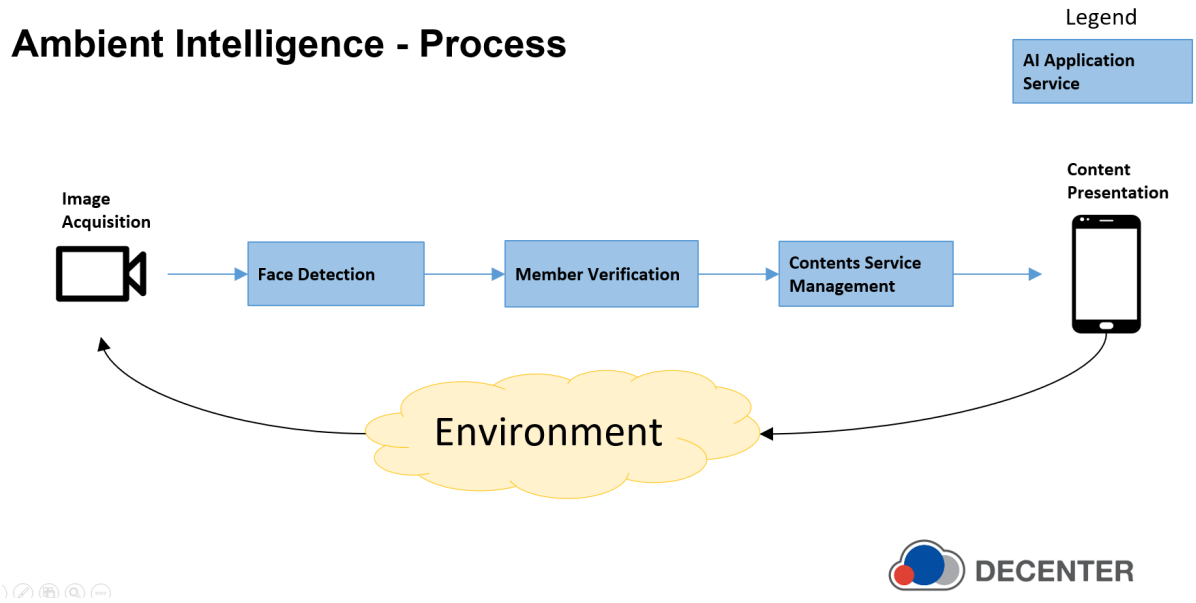


Figure 40 Process view of UC4 AI Application

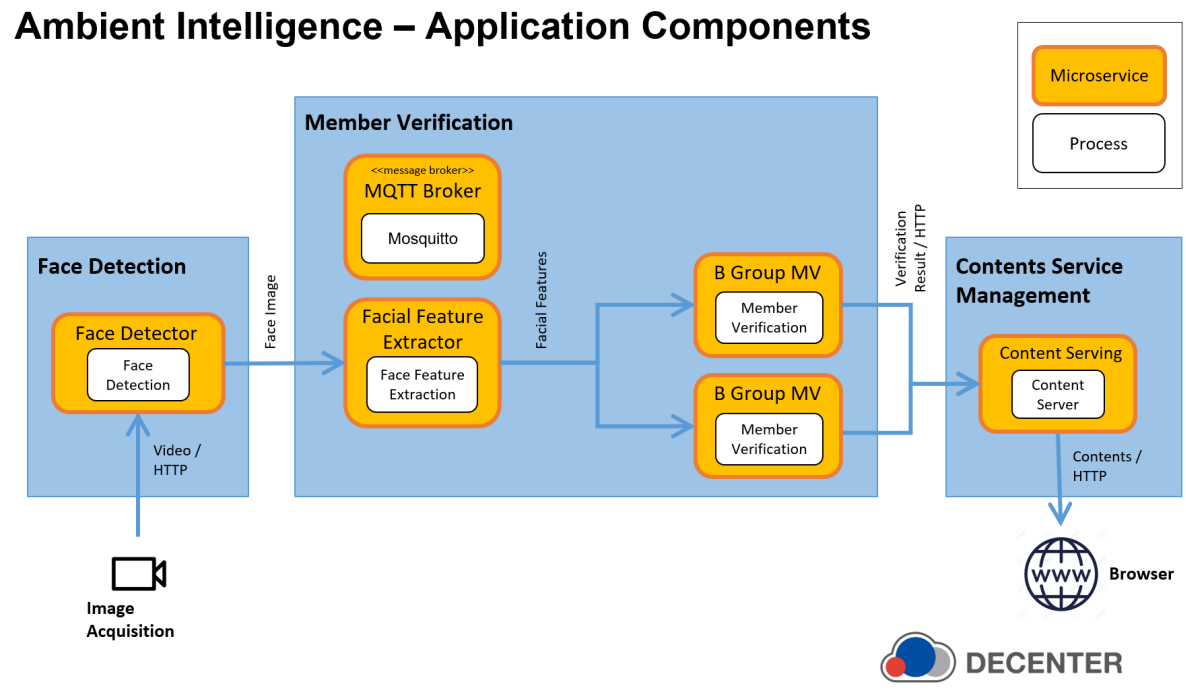


Figure 41 The Application Components view shows the control flow between microservices (enclosed in each Application Service), in UC4

Ambient intelligence – Deployment

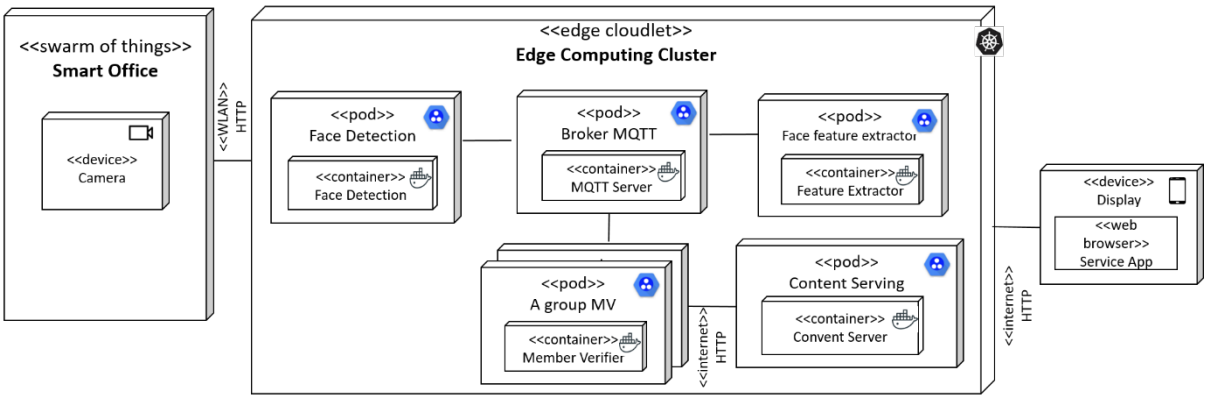


Figure 42 Example of deployment of UC4 containerised microservices on Kubernetes-based Fog Platform

6.4 Interaction with Platform

6.4.1 Adding resources to the infrastructure

Once the AI microservice container image is built, it may need to be deployed on a device with proper resources, such as a computing acceleration hardware like a GPU. For this, it is necessary to describe the computing resources it needs from the infrastructure to be used in the resources request and orchestration of containers on the DECENTER platform.

There are various types of acceleration devices. In DECENTER use cases, two types have been used so far: a GPU (Graphics Processing Unit) and a SoC (System on Chip). GPU device refers to a specific device that can be used for computing acceleration and that is particularly suitable for the kind of computation typical from linear algebra (i.e. operations with matrixes). A GPU device is usually made of a graphics card, which consists of a chip and dedicated memory for faster computation. SoC device refers to a hardware with acceleration-device-integrated system-on-chip. For this type of device, memory is usually shared between the CPU and the acceleration unit. To facilitate the request and orchestration of this types of compute resources, DECENTER defined a model to describe them, which is described in Table 11.

Table 11 Resource definitions for AI

Label	Values	Description
decenter.accelerator_type	GPU SOC	Describes whether the accelerator is SoC or GPU.
decenter.accelerator_name	String	Describes the name of the accelerator. It consists of vendor name and hardware name e.g) nvidia_gtx1080ti
decenter.accelerator_driver_version	int.int	Describes version of device driver
decenter.accelerator_memory	nGB	Describes size of memory dedicated to the accelerator

In Y2, the “accelerator name” label is implemented to identify GPU resources. Thus, a custom label (accelerator) has been added to the description of each node in the cluster to identify the location of GPU resources, and then used for the request and allocation of that specific resource. Custom Resource Definitions (CRDs) are defined for this description, and the details are can be found in Annex A of D3.3.

6.4.2 Request resources for the deployment

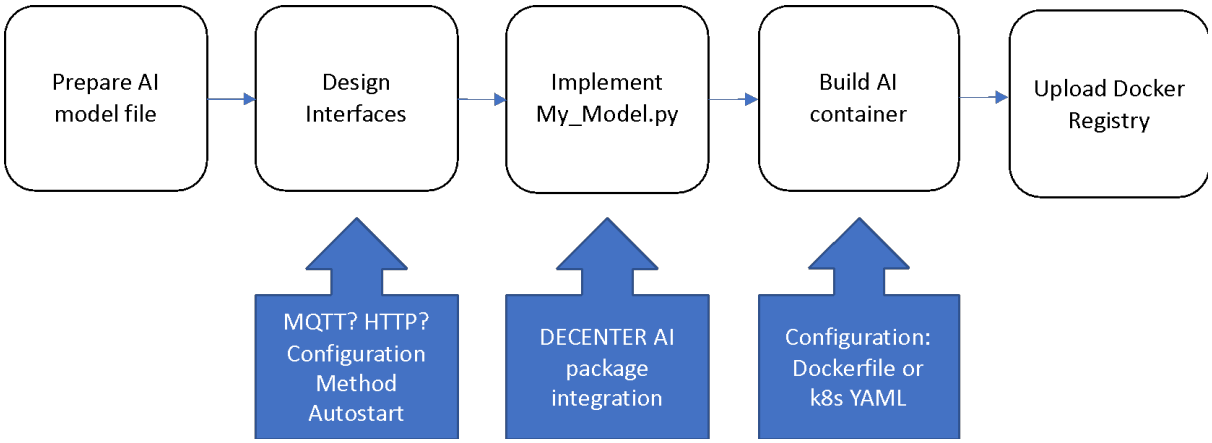


Figure 43 AI-container building process with DECENTER AI package

Figure 43 depicts the development process of an AI microservice from an AI method. When building a microservice which contains AI methods and makes use of DECENTER AI package, the developer needs to decide what the interfaces are going to be such HTTP, MQTT or both. The design patterns proposed in this document can help decide the best combination of protocols with respect to the purpose of AI application. And then developer needs to refactor its AI logic to fit into the templates in My_Model.py. The templates in My_Model.py are designed to interact with various protocols with the help of DECENTER AI package which means that less considerations of network interfaces are required for AI developer. A docker container can be built with My_Model.py, DECENTER AI Package and other libraries and registered to the docker repository.

In DECENTER, AI model file are processed independently from the AI container. The AI model files are stored in an AI model repository, which is a submodule of cross-border data management module. So when registering the AI container on to docker registry, it needs to register the AI model to be used with the AI container on the AI model repository as well.

All the resource requirements for the AI microservice are identified on the design phase – i.e. existence of GPU, type of GPU, memory for GPU, etc. Those AI-related resource requirements are to be described in either the Dockerfile manifest or the Kubernetes Deployment definition according to the deployment environment. The orchestrator of the DECENTER platform will choose appropriate node to deploy the container by comparing the resource request and custom label on the node.

7 Federated Learning with DECENTER

Federated Learning (FL) is a recent learning method which makes use of decentralized data on decentralized resources. The concept of this decentralization is quite a good fit for DECENTER outcomes; therefore the consortium has investigated applying FL to update the AI models that run into AI microservices on the DECENTER platform to verify its utility on decentralised learning scenarios. In our solution to support FL in DECENTER , we make use of AI model repository (T4.3), DECENTER AI package (T4.4) and DECENTER Platform as building blocks to implement a practical AI Service which runs a training (i.e. model building) process—besides the conventional inference (i.e. model serving) process—based on FL principles. This section describes the details of the design and implementation of such a process.

7.1 Introduction to Federated Learning

Usually, training DNN models require an important amount of computational resources; for example, the memory requirement for a training process such as that for handwritten digit recognition problem with MNIST dataset goes up to 1GB, and to speed up it needs many CPUs or even a dedicated acceleration hardware such as GPU (Graphics Processing Unit). Due to this reason, using a cloud machine instance or high-performance cluster of instances for building a Deep Neural Network (DNN) model is a common practice. In an environment like that, the training process is carried out centrally, as data are collected, stored and processed in a single data centre, with centralized resources such as storage and compute.

Federated Learning (FL) is a modern approach to use decentralized data and resources for the model training phase of the ML process. Instead of applying centralized data for the training on a resource-rich environment, FL applies decentralized local data to the training which is carried out locally, with local resources such as compute nodes and storage, to build a local version of the model to later aggregate it with together with other local models built on other locations, into a global model.

There have been several studies about the FL including protocols and model aggregation algorithms on the (central) server side [14]. However, to apply FL techniques to continuously build and optimize a production AI method, we need to design, implement and deploy a FL system able to operate side-by-side with the AI services containing those methods, providing it with accurate and up-to-date ML models on a continuous basis.

DECENTER focuses on orchestration of cloud and edge resources for distributed AI application deployment, so it is a good context to apply and test the benefits of FL. This section describes activities carried out in Y2 on how DNN models are managed so to provide a good performance for the training process, and how the FL services can be operationalised within the DECENTER Platform along with the other DECENTER facilities including Model Repository and AI Package.

7.2 AI Service and Federated Learning

Federated Learning (FL) is a novel model training approach for Deep Neural Networks (DNN), which uses decentralized data and infrastructure resources for model building or training [2]. The data pipeline for the AI model consists of collection of data to be trained, building deep learning model by training, deploying the trained model, and retrieving feedback from the deployed trained model. Typically, the training of a deep learning model requires a vast amount of training data and rich computing resources such as large amount of storage, memory or an acceleration hardware like GPU; due to those constraints, training usually takes place in a centralized cloud or data centre. Instead of training in a centralized way, the FL

process make use of distributed data and resources—like those found in edge computing infrastructure—for the training.

A reference architecture for FL systems [14] has been proposed with the description of the actors and their collaboration protocol (i.e. control and data flows between those actors) to build ML models. The actors identified in the FL process are as follows:

- **FL Client** processes local update of an ML model with local training data. It receives a global model from the FL Server and update it by training with local training data. When the update is complete, FL Client uploads local model updates to the FL Server.
- **FL Population** is made of a series of FL Clients that participate and join in the FL processes.
- **FL Server** is the main actor for the FL process. It distributes the global model to be trained locally at the FL Clients in FL Population, and receives updated local model from them. It updates global model by aggregating the local model updates and re-distribute a new (i.e. updated) version of the global model to the members of the FL Population.

The protocol devised by the authors of such reference architecture focuses on secure and efficient communication between the FL Server and FL Clients. The local ML model update refers to a model trained in an FL Client while the global ML model update means a model generated by aggregating those local model updates. The local model and its updates exist only for the FL process; therefore, they won't be used in the AI application service. The updated global model is the one rolled out back to the AI microservice running the model, to improve its performance.

The FL decentralized model of data collection and processing fits quite well into DECENTER. If the concept of FL can be applied to the cloud-to-edge continuum, FL could be realised on the DECENTER platform. To populate FL, the interaction between the AI application service and the model update with FL needs to be clearly identified. That interaction needs considerations from the viewpoint of model deployment which is also a good fit for edge computing architecture. Motivated by this idea, an architecture which is based on DECENTER platform to provide reliable AI application services based on FL distributed over the cloud-to-edge continuum has been designed and implemented.

7.3 Design of Federated Learning Architecture with DECENTER

7.3.1 Model Management for Inferencing and Learning

The FL process defines protocols for data and control flows between the FL Server and the FL Population (i.e. a group of FL Clients). The global model will be updated by aggregating local updates from FL Client by FL Server, which will be distributed to FL Client again so to make AI application to use the updated global model for a service. Therefore, to provide a practical FL process, the relationship between the FL process and the AI application service needs to be considered as well as the FL Server-Client relationship.

The interaction between AI application and FL process can be defined as follows. First, the AI application should support replacing the ML model it is using for the new (i.e. updated) global model built by the FL Server. When the updated global model is delivered to the FL Client, FL Client needs to store that model to local storage and able to tell AI application service to use updated global model for its service. However, the updated global model does not always guarantee better performance. FL is heavily dependent on locally collected, and sometimes the performance of the global updated model is not as good as the previous one. Yang et al.

[16] have defined the accuracy loss function for the FL algorithm; we use that function as the criterion to decide whether to roll out the global model update to the AI application. Let the accuracy of a model trained with FL be V_{fed} and the accuracy of model trained with centralized data V_{cent} .

$$V_{fed} - V_{cent} = \sigma \text{ . (1)}$$

The global model update will be applied to the application only if σ has non-negative value. When eq. 1 is extended to the iteration process of FL, it becomes

$$V_n - V_{n-1} = \sigma_n \text{ (2)}$$

where V_n refers the accuracy of current round and V_{n-1} refers that of the previous one, and the global model of current round will be applied to the application only when σ_n has non-negative value. Otherwise that global model will be applied to the FL Client only. To manage these different model management policies between FL Client process and AI application, the AI Model Registry of DECENTER is used. the Model Registry handles ML model updates for both processes—i.e. inference or scoring of the AI application service using the model, and rebuilding of the model by the FL Client after the collection or new data.

While designing the proposed application, care has been taken to keep changes of AI application as less as possible. It would be very inefficient for a developer to re-write the AI application completely to work with the Federated Leaning. So, in the proposed application, model management is solely processed by the FL Client with DECENTER AI model repository, only the managed model will be shared with the AI application. The only thing that is needed for an AI application is to substitute the model. When the global model update is received by a FL Client, the FL Client will notify its change to the AI application so that the model of the application can be replaced to a new one. The implementation of the AI application used DECENTER AI package, with an extra RESTful interface to handle this model change request.

7.3.2 Resource orchestration

Among the different architectural styles of cloud computing, edge computing fits nicely into the concept of FL. The edge computing approach makes use of computing resources near data source or service endpoint to provide a service with better QoS with less privacy issues. Both cloud and edge resources are used to provide a service with respect to the characteristics of each one in a distributed manner. FL Clients are deployed in edge nodes while the FL Server is deployed in a cloud node. The FL Client collects local data and update local model from global model by undergoing a local training process—with new local data collected at the edge node where the FL client is running, and no local data will be uploaded to the cloud.

Resource managements are of importance for the FL Process since in FL Process the training is distributed to all over FL Population as well as FL Server. Unlike the centralized training, the training process follows the communication protocols between FL actors. This also means the process entirely relies on the connectivity between the compute nodes running them. Therefore, if one of the actors fails due to some reason the overall training process will be affected. Suppose that an FL Client without enough resources for training joins the FL Population: the FL Client will continuously fail to deliver its local model update to the FL Server and so its local data won't be considered when building the global model update. Another example can be an FL Server with not enough resources such as storage, network or CPU. If the storage is not enough, the local update from FL Population cannot be stored and discarded, thus degrading performance of aggregation algorithm. On the other hand, if the CPU of the edge node is not fast enough, its network bandwidth is not enough, or the

communication link experiences continuous failures, the whole FL process will be delayed and, more importantly, might also become unstable or unreliable. To provide a proper FL process, allocation of appropriate resources to each process/task is essential, and cloud technologies used at the edge can be used in this resource management. The idea would be to use elastic scalability at the edge so the FL Clients can request resources on demand and scale up and down at will, which can be achieved with DECENTER platform or Kubernetes.

7.3.3 Architecture

Figure 44 depicts the architecture of our low-invasive FL system on DECENTER. [14] proposed an efficient communication method to update a deep learning model using FL; the main architecture building blocks—FL Server, FL Population, FL Client, FL Process and structure of our system follow those of that work. Moreover, our architecture proposes two new building blocks: Model Registry and AI Application. Also, in that work, some cloud instances where used to run the distributed FL process; in our approach we propose a more realistic setting with docents of instances (edge nodes), where the FL client and the AI application co-exists.

Model Registry stores the global models produced during the FL Process and works as a gateway for model updates and distribution. All the communications regarding the global model are made through this Model Registry. For that purpose, the Model Registry provides an interface to upload and download any model version. When the global model is updated with non-negative σ_n value, a notification is sent to the AI Application so to make it reload the new (updated) version of the global model. Finally, in the proposed system architecture, the FL Server and Model Repository utilise resources in the cloud while FL Clients and AI Application replicas use resources at the edge.

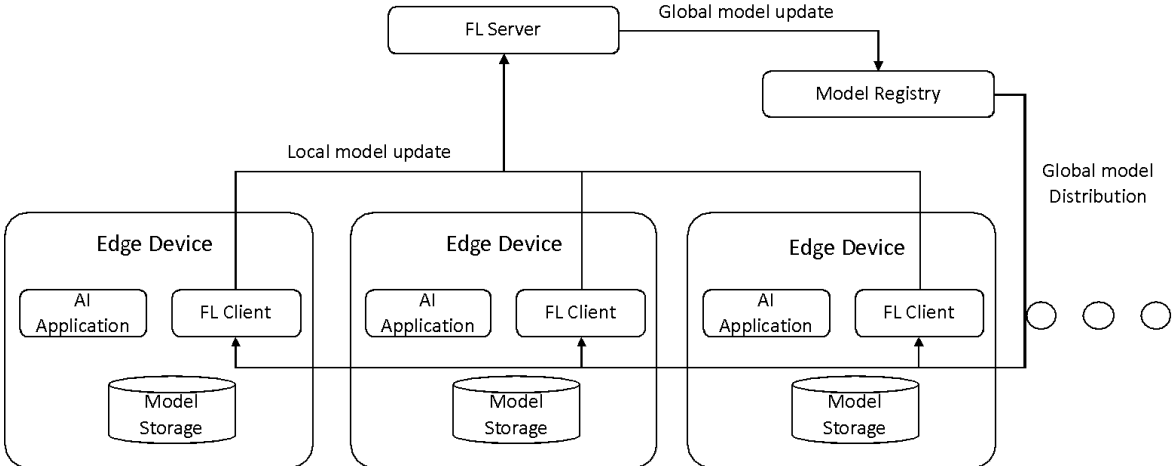


Figure 44 FL service configuration with DECENTER on cloud-edge environment

7.3.4 Data Flow

For the FL Client and AI Application processes to be scheduled on the same edge node or edge node cluster, we need to manage the global model. In the proposed architecture, instead of managing global model in those two processes—running in parallel independently, FL Client manages the life cycle of the global model in that edge node. And for example, this very same model is shared with the AI Application to update its internal model version. This feature is responsible for **one of the main benefits of our FL system: the independence of the AI Application from the FL system**, which continuously updates the ML model locally and

serves them to AI application, with the latter giving computer vision, machine translation or speech recognition services to the users. **In fact, this is why we refer to the low-invasive nature of our novel FL system against its conventional counterpart.**

The flow for FL Process model management protocols are as follows. First the initial model is registered to the Model Registry. When FL Process is initiated, the FL Client registers itself to FL Server to join the FL Population. If there are enough number of FL Clients, FL Server requests update of local model to FL Population. Upon receiving the request, the FL Clients which consist FL Population start local update process. It will retrieve the global model as designated in the request from the Model Registry first, and then it will train the global model to generate local model update by applying local data. When local model update is ready, each FL client transmits it to the FL Server directly. FL Server updates the global model by aggregating local model updates, and then register it to the Model Registry with a new version.

However, the global model update does not guarantee better performance then the previous version. To avoid possible deployment of less accurate model to the AI application, the FL Server register a new version of global model with a special tag 'interim'. And then the FL Server requests validation test to the FL Client with the location of 'interim' model. The FL Client downloads the new model, validate it with the local data and send validation results to the FL Server again. The FL Server check the performance of the new model by comparing validation results received from FL Client, and if it has better performance (non-negative value of eq. 2), it finally removes the 'interim' tag from the model uploaded in Model Registry. When the global model update is complete, the FL Client will let the AI application know that a new version is ready, so to substitutes its model to the new version. These flows for FL Client and FL Server are described in Figure 45 and Figure 46, respectively.

For the AI application and FL process co-exist on a single machine, whether it is a VM or bare metal, it needs storage and memory for the model(s). In the proposed system, rather than managing the model for each process independently, it is designed to share the model among those two processes. The FL process manages the model, since it has roles of updating local model, downloading and validating global model update. The downloaded model will be shared to the AI process. In this way, the storage for the model on a machine can be saved to half, while keeping seamless service running. Also, the network bandwidth is saved accordingly.

Let *M* be the size of global model, B network traffic for FL process, N number of FL Population and R number of rounds for an iteration. Without DECENTER AI model repository, the size network traffic for FL Server is going to be $T_{FLS} = M \times N \times 2$ since it needs send global model and receives local model update to/from FL Client, and the size of storage required on the edge is $S_{FLS} = M \times 2$since it needs to store models for each FL Client and AI application. With the proposed architecture, since the network load is distributed to Model Repository, the network traffic for FL Server is reduced to half. And the size of storage can be reduced to half as well.

```
Register to FL Server;
Download initial version of global model from Model Registry;
while converged != true do
    Wait for FL Server local update request;
    if local update request received then
        update model with locally collected data;
        send local model update to FL Server;
    end
    Wait for FL Server validation request;
    if validation request received then
        download global model to be validated from Model Registry;
        validate global model;
        send validation result to FL Server;
    end
    Wait for converged;
end
Notify AI application to load new global model;
```

Figure 45 Algorithm of FL Server

```
if FL Population ready then
    for each FL Client do
        send local update request
    end
    Wait for local update;
    if round period expired and enough local update received then
        aggregate local update;
        register global model update with 'interim' tag to Model Registry;
    end
    for each FL Client do
        send validation request
    end
    Calculate loss function;
    if loss is less than previous global model then
        register global model update without 'interim' tag to Model Registry;
        converged = true;
    end
    for each FL Client do
        send converged;
    end
end
```

Figure 46 Algorithm of FL Client

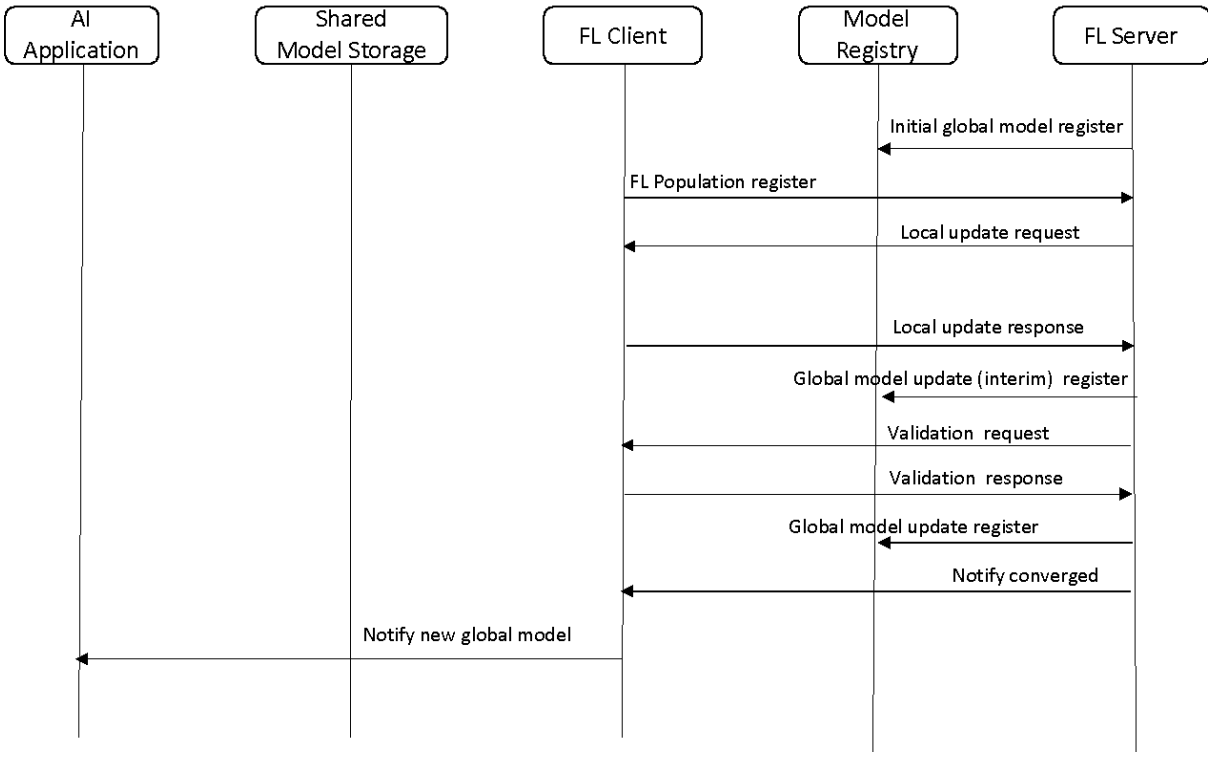


Figure 47 Process between instances of FL process

7.4 Evaluation on MNIST data set

FL consists of various entities to collect local data, train local model and protocols to exchange local model update and global model update between FL Server and FL Client. To improve a deep learning model via FL, it needs to build an FL Population consists of FL Clients. Our DECENTER-based FL system applies a microservices architectural style to ease the management of FL services on the edge and the cloud. Unlike monolithic client-server architecture, microservices architecture combines loosely coupled microservices to build a service, which makes deployment and maintenance easy. The system is decomposed into several microservices with its own roles, which are interconnected through messaging protocols. FL Server microservice implements aggregation algorithm to update global model in the FL. It receives local model updates from FL Population, and build global model update by aggregating them. The global model update is registered to the Model Registry so to make it available to FL Client and AI application. FL Client microservice implements local model updates. It registers to the FL Server to build FL Population. It retrieves cur- rent version of global model from Model Registry and build local model update by training it with locally collected data. The global model of FL Client is shared to the AI Application, which hosts service logic with the global model. Each microservice is independent from each other, which makes it available to update functionalities of whole FL data pipeline. For example, supposed that a developer trying a new aggregation algorithm for his FL Server. In the monolithic service architecture, updating aggregation algorithm may require modification of related software codes, such as collecting local update or registering global model update to the Model Registry. In the proposed model, since all the services are de-coupled with microservice architecture, a developer can update the aggregation algorithm just by updating FL Server microservice and re-deploy it to a node. The microservice structure is depicted in Figure 48.

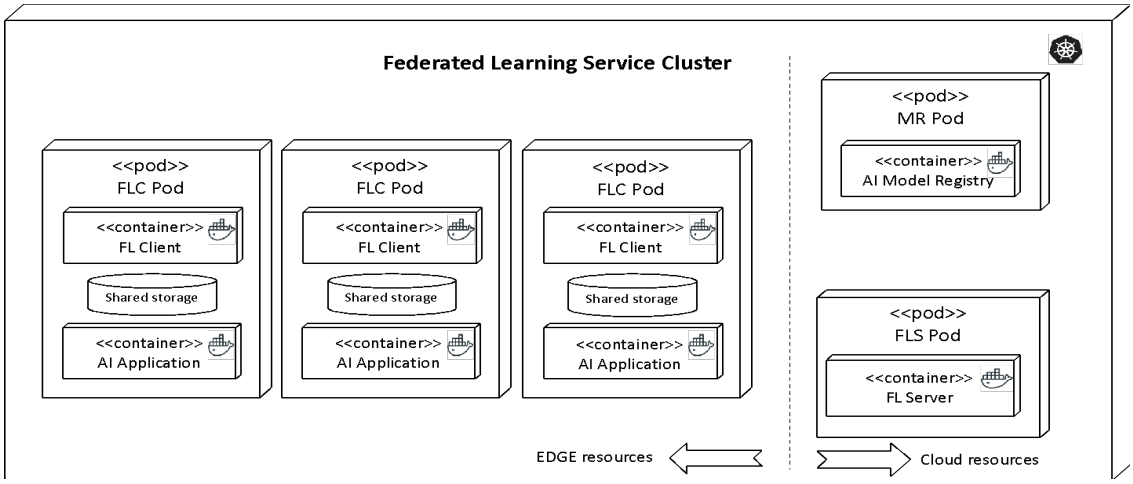


Figure 48 Microservice deployment configuration on Kubernetes cluster

Table 12 RESTful APIs for model management in FL process

Server/Client	REST API Endpoint	Description
FL Server	update_local_ model	transmit local model updates from FL Client to FL Server. Message Body contains local model update
FL Client	request_local_update	request local update from FL Server to FL Client. Message body contains name and version of global model which is registered on Model Registry.
FL Client	request_validation	request validation of global model update. Message body contains name and version of global model which is registered on Model Registry.
Model Registry	register_model	register global model to the Model Registry. Arguments: model name, model version, interim and model weights.
Model Registry	register_model	request download of a global model from the Model Registry. Arguments: model name, model version.

The proposed FL system architecture has been implemented to validate whether it can be applied to FL in a real-world environment. The image classification is chosen to the use case for the implementation, and MNIST dataset has been employed. The machine learning model of the AI application is a deep learning method called Convolutional Neural Network, which is specifically suitable for object recognition and handwritten digit recognition.

It provides a web service for the image classification. User can access MNIST AI Application via a web-browser. FL Client will choose random images from MNIST dataset to simulate local data collection. FL Client uses those data for local model update, which will be transferred to the FL Server, and FL Server will deliver global update model via Model Registry. Docker container is used to implement the microservice architecture in the proposed system. Each microservice which consists the system is implemented as a docker container images. Those docker container images are deployed to the node which has enough computing resources, and DECENTER platform is used for the deployment and resource orchestration. For example, to apply training with MNIST image, it needs at least 2GB of memory and 4 CPUs.

The DECENTER resource orchestration plays an important role in the proposed system. As seen in the previous section, if an FL Client with not enough resources exists in the FL

Population, that FL Client resources and local data are not going to be involved in FL Process and wasted. With the resource orchestration only FL Clients with suitable resources will join FL Population which will bring optimized resource utilization and increase performance of FL Process.

Deployment of MNIST FL System on a DECENTER cluster

The proposed FL system was designed and build as a cloud-native solution, making use of Docker containers and Kubernetes (together with for example Istio for service mesh). It the case of this research, the system was implemented on a Linux. The host PC has a quad-core CPU, 16GB of Memory and 1TB of storage. Theoretically it can host at least 3 Pods to run the FL Client. The microservice architecture for MNIST application and FL Process are shown in Figure 48.

The microservices are deployed in three kinds of Pods according to its purpose. The AI application and FL Client are deployed to the same pod (FLC Pod) so to share a common volume where global models are stored, and there are two more Pods where Model Repository and FL Server microservices to be deployed (MR Pod and FLS Pod, respectively). Total four Docker containers have been built for each microservice implementation, which are FLC container, FLS container, MR container and AI container. As of now, FLC Pod is configured to host both FL Client and the AI Application for the simplicity of implementation, however it is also possible to deploy FL Client and AI application separately. After deploying AI application to provide services to the client, and use DECENTER platform to find the Pod where the AI Application is deployed and deploy corresponding FL Client to the same Pod. In this way the resources in the client device can be saved - instead of deploying both AI Application and FL Client, the system can be used to choose an FL Client which has proper resources to join FL Population. It needs multiple FLC Pods to build the FL Population. To deploy multiple FLC Pods, REPLICA set of Kubernetes is employed. The other two Pods are the single ones - only one instance of each exist in the same DECENTER cluster.

Analysis

The training parameters of each FL Client has been set as batch_size = 20, epoch = 5, and three FLC Pods are deployed to build FL Population along with FLS and MR Pods. The communications between each microservices are implemented as described in Section 4. The proposed system works as designed, providing both AI Application service while updating model with FL. When the global model is updated, it is shared between the AI Application and FL Client on FLC Pod, so to AI application uses updated model. With no management between FL Process and AI Application, FL Server processes both global model update (aggregation) and serving the model to FL Population. If the number of FL Population grows, the computational loads of serving global model will be increased accordingly. The proposed system separates the role of global model update and serving with the concept of DECENTER AI Model Repository, also with scalability by leveraging cloud technologies. The AI Model Repository can scale up or down with respect to the bandwidth requirements of FL Population.

In the experiment, size of an MNIST model was about 48MB. Without the proposed architecture, which means the AI application and FL Process exist independently on different Pods, each Pod will use 4.8MB of storage for storing MNIST model for training and inference. In the proposed architecture with DECENTER platform, FL Client and AI Application exists in the same Pod sharing a volume, and there exist two model files at maximum - one with interim tag and the other one without interim tag. There will be only one model file shared between two processes only when it converges. In the experiment FL Process converges on the third round, and the total amount of storage saved by the proposed architecture is 1.2 MB, which

is about 25% of original model size. It is proportional to the rounds taken until it converges. The network load of FL server is offloaded to the Model Registry in the proposed architecture as well. In this experiment there are three FL Clients in FL Population, which means that three transmissions of models are required for both global model update and local model update. In the proposed architecture distribution of global model is offloaded from FL Server to Model Registry, and the network requirements of FL Server is reduced to slightly more than half. In the experiment the average bytes sent on each round from the FLS Pod was estimated as 5.0 MB, and average bytes received on the same Pod was estimated as 19.2MB. The experiment results show that the distribution of global model update has now offloaded from the FLS to MR effectively. It shows that total received bytes of FLS is directly proportional to the number of FL Population, however the sent byte remains the same.

The proposed system also ensures the return of local model updates by resource orchestration. When building the FL Population it uses resource orchestration methods of Kubernetes, so to make sure that only FL Clients with suitable resources to join the FL Population. The experiments show that at least 2GB of memory and 2 CPUs are required to finish the local model update in a Pod within a round, so the configuration with those values has been used in the implementation of MNIST FL system to make sure that all the results are returned within the each round period. The proposed system shows a practical method to apply global model update on the running AI application. It makes use of shared repository between the containers to reduce overhead to manage deep learning models on two processes independently. This can save more resources on the FL Client, from the viewpoint of storage and network bandwidth.

The proposed system is built with DECENTER which makes management easier. All the microservices are loosely coupled. If update of new aggregation algorithm is required, it can be done by substituting FL Server microservice to a new one with new aggregation algorithm, without intervening any other microservices.

7.5 Remarks

In this section, we have presented the DECENTER activities—and facilities created—aiming to realising a Federate Learning (FL) system on the DECENTER platform. The main benefits of this DECENTER-based FL system come from decoupling the AI application service from the ML model training system running on the cloud-to-edge continuum. **In fact, this is why we refer to the low-invasive nature of our novel FL system versus its conventional counterpart.** The proposed architecture applies edge computing for a service to participate in the FL system, to make itself available to update a model being used on the service, using the data generated locally. The modules which consist the architecture are designed with microservice architecture, which provides more flexible configuration and maintenance. The RESTful interfaces of each microservice are also defined. To optimize resource usage, a model management protocol is proposed to reduce requirements of memory and storage. It has been implemented and validated on a DECENTER cluster for an image classification application and proved using edge resources for the real-world FL application. The results show that a service with FL model update can be decomposed into microservices and deployed on edge resources. Furthermore, it provides efficient resource management by exploitation of existing cloud technologies. The model update protocol helps reducing the storage requirements by sharing a model between two processes on FL Client, and distribution of global model has been offloaded from FL Server so to make the system more scalable. In the future, more complex model will be tested with the proposed architecture and deployment on a cluster with embedded edge devices will be investigated.

8 Conclusion and future works

Table 13 shows the requirements for decentralised AI that were identified in Y1, and how they are addressed in DECENTER until M24. The rightmost column on the table shows where the requirement has been addressed (TBA means To Be Addressed in next year). Some of the requirements needed actions from other work packages besides WP4 so they are resolved with collaborative work between work packages (also specified in the last column).

Table 13 Requirements of AI and how they are addressed.

ID	**Requirements for AI application to Fog Platform**	**Target**	**Addressed**
Containerization of AI specific resources and orchestration			
RAI-01	*AI Application shall be able to be containerized.* **Description**: An AI application is supported by an AI platform, such as TensorFlow, Caffe, or MXNET, and the application logic should be written in the corresponding APIs and languages.	AI method and solutions	**T4.4** **DECENTER AI Package**
RAI-02	*Containerized AI application shall be able to access H/W acceleration devices such as GPU and its memory.* **Description**: To perform H/W accelerated computation for AI, H/W acceleration device shall be supported in the container. A H/W accelerator can be a GPU from Nvidia or AMD, or an FPGA from Intel.	AI method and solutions	**T4.4, WP3** **Custom Node Label on the Platform**
RAI-03	*Containerized AI shall be able to be accessed using uniform interfaces.* **Description**: Containerized AI, either method or application, shall be able to expose its interfaces to other microservices so that an AI service can be built with it.	AI method and solutions	**D4.4** **DECENTER AI Package**
RAI-04	*Resources for AI application (or Service) shall be assigned properly and be managed throughout the lifetime of AI application.* **Description**: Suitable resources shall be assigned with respect to the AI application's request. For example, the application may request for GPU-enabled edge with TensorFlow (CUDA-8.0), and IoT control of IP camera and edge for image processing. Also, the request may include access to Model for image classification, compatible with TensorFlow and suitable to run on embedded GPU.	AI method and solutions, Fog Computing	**T4.4** **Design Pattern** **WP3** **Platform**

RAI-05	*Node with H/W acceleration shall expose its H/W acceleration resources for resource orchestration.* **Description**: H/W acceleration (e.g., GPU) and its related resources (e.g., memory) shall be exposed to the application, so that the application can make a request to access those specific resources. (e.g., Type of GPU, Name of GPU, Associated Memory Size)	AI methods and solutions	**WP3** **Custom Node Label**
RAI-06	*For the resource orchestration, occupancy of H/W acceleration resources shall be exposed.* **Description**: To make efficient use of H/W acceleration resources, their occupancy shall be exposed. If it is occupied by a container, no other container can use it until it is released.	AI methods and solutions	T4.4 AI Solution (TBA)
Orchestration of AI specific resources and other resources			
RAI-07	*Location of a Resource (cloud or edge) shall be exposed.* **Description**: Location (Edge or Cloud) of a specific resource shall be exposed, so that container can be installed/started on the desired Node.	Fog Computing	**WP3** **Node Label**
RAI-08	*The nearest edge to the service endpoint shall be able to be identified.* **Description**: To guarantee prompt response to the service endpoint, the nearest edge to it shall be chosen for the AI application deployment.	Fog Computing	WP3 Node Label (TBA)
RAI-09	*Devices for AI Service shall be able to be selected with proper user interface.* **Description**: When an AI service needs input from a specific device, such as a camera, the application should provide an appropriate user interface to allow the selection of the proper device.	Fog Computing	WP3, WP4 App Composer, AI Solution (TBA)
RAI-10	*AI applications shall be able to provide inference results in a uniform way.* **Description**: If a particular micro-service needs results of the inference engine at an edge, this inference engine should be able to deliver the results to that micro service.	AI methods and solutions	**T4.4** **DECENTER AI Package and Design Patterns**
RAI-11	*Micro services, composing a single AI, service shall be able to identify each other and communicate to each other.* **Description**: For example, if an AI application needs to send an event to an alert service	Fog Computing	**T4.4** **DECENTER AI Package**

	application, each application shall be able to send and receive information from each other.		**and Design Patterns**
Management of AI and Model Container			
RAI-12	*AI Application shall be able to be updated.* **Description**: If an AI application has been modified, the deployed application on the Node shall be updated accordingly.	AI methods and solutions	**WP3** **Platform**
RAI-14	*AI Model shall be able to be updated.* **Description**: When the AI model at the cloud changes, the AI on the edge shall be able to be updated accordingly.	AI methods and solutions	**T4.3** **AI Model Repository**

In summary, through the activities of WP4 the following results have been achieved in the second year.

- Facility to help building an AI microservice from an AI method
  - DECENTER AI package provides intuitive ways to containerize AI methods as a microservice
  - It provides two network protocols to configure and control AI method, along with a few design patterns for AI service design
- AI optimization methods
  - Novel pruning methods to make AI model faster and lighter
  - More suitable AI model to run on an Edge node
- Utilization of acceleration device within a cluster
  - Specific acceleration device request for AI microservice to be deployed
  - Custom node label to identify acceleration device
- AI solution design with the above facilities
  - AI solution reference architecture and guidelines for the use cases
  - AI service design principles and patterns to use DECENTER facilities such as APIs of AI microservice, custom resource labels, AI model repository, etc.
- Evaluation of FL on DECENTER
  - Practical example of using DECENTER on FL
  - Model management, AI microservice has been used for AI application with FL implementation

WP4 will continue investigating and implementing those facilities in Y3 of the project, focusing on the implementation of testbed pilots for each use cases. The activities for the next year includes:

- Evaluation of AI optimization methods on the testbed
- Update of DECENTER AI package
- Update of AI solution reference architecture with respect to the testbed pilots
- Incorporate model training processes besides inference ones within the AI solutions
- Cross-border data management implementation

9 References

[1] Jia, Yangqing. "Learning semantic image representations at a large scale." PhD diss., UC Berkeley, 2014.

[2] Nvidia Accelerating AI with GPUs: A New Computing Model 2016

[3] Li, Jian, Yabiao Wang, Changan Wang, Ying Tai, Jianjun Qian, Jian Yang, Chengjie Wang, Jilin Li, and Feiyue Huang. "DSFD: dual shot face detector." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5060-5069. 2019.

[4] He, Yihui, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks." In Proceedings of the IEEE International Conference on Computer Vision, pp. 1389-1397. 2017.

[5] Han, Song, Jeff Pool, John Tran, and William Dally. "Learning both weights and connections for efficient neural network." In *Advances in neural information processing systems*, pp. 1135-1143. 2015.

[6] V. Vanhoucke, A. Senior, and M. Z. Mao, “Improving the speed of neural networks on cpus,” in Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011, 2011

[7] M. Courbariaux and Y. Bengio, “Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1,” CoRR, vol. abs/1602.02830, 2016.

[8] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. "Speeding-up convolutional neural networks using fine-tuned cp-decomposition." arXiv preprint arXiv:1412.6553 (2014).

[9] Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

[10] Wang, Junpeng, Liang Gou, Wei Zhang, Hao Yang, and Han-Wei Shen. "Deepvid: Deep visual interpretation and diagnosis for image classifiers via knowledge distillation." IEEE transactions on visualization and computer graphics 25, no. 6 (2019): 2168-2180.

[11] DECENTER, “D4.1 First release of applications’ artificial intelligence methods and soutions,” 11.07.2019

[12] DECENTER, “D3.3 Second release of the fog computing platform”.

[13] DECENTER, “D4.2 Initial report on cross-border application data management,” 14.01.2020

[14] Bonawitz et al. “Towards Federated Learning at Scale,” System Design, Feb. 2019 , http://arxiv.org/abs/1902.01046.

[15] Mcmahan HB et al., “Federated Learning of Deep Networks using Model Averaging,” 2012, https: //pdfs.semanticscholar.org/8b41/9080cd37bdc30872b76f405ef6a93eae3304.pdf.

[16] Yang Q, Liu Y, Chen T, Tong Y, “Federated machine learning: Concept and applications,” ACM Transactions on Intelligent Systems and Technology, 2019 jan;10(2):1–19.

[17] DECENTER, “D5.1 First release of the AI-integrated fog computing platform, demonstration KPIs and first setup of pilots”, expected in July, 2020.

[18] Erin Liong, Venice, et al. "Deep hashing for compact binary codes learning." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[19] Jang, Young Kyun, and Nam Ik Cho. "Generalized Product Quantization Network for Semi-Supervised Image Retrieval." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

[20] Yuan, Xiaoyong, et al. "Adversarial examples: Attacks and defenses for deep learning." *IEEE transactions on neural networks and learning systems* 30.9 (2019): 2805-2824.

[21] Madry, Aleksander, et al. "Towards deep learning models resistant to adversarial attacks." *arXiv preprint arXiv:1706.06083* (2017).

[22] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." *arXiv preprint arXiv:1503.02531* (2015).

[23] Yang, Shuo, Ping Luo, Chen-Change Loy, and Xiaoou Tang. "Wider face: A face detection benchmark." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5525-5533. 2016.

[24] Georgiadis, Georgios. "Accelerating Convolutional Neural Networks via Activation Map Compression." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7085-7095. 2019.

10 Abbreviations

ML	Machine Learning
AI	Artificial Intelligence
UC	Ucase case
FL	Federated Learning
DPO	Data Protection Officer
EC	European Commission
ECRF	Effort and Cost Reporting Form
FS	Financial Statement
EU-GA	European Grant Agreement
GAS	General Assembly
IITP	Institute for Information & communications Technology Promotion
IM	Impact Manager
IPR	Intellectual Property Right
MSIT/IITP	Korean Ministry of Science and ICT
PM	Person Month
PMT	Project Management Team
PO	Project Officer
PR	Periodic Report
RP	Reporting Period
ToC	Table of Contents
WP	Work Package