D4.1

First release of application’s Artificial Intelligence methods and solutions

02/12/2019
The DECENTER Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

<table>
<thead>
<tr>
<th>Date</th>
<th>Edited by</th>
<th>Status</th>
<th>Changes made</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.05.2019</td>
<td>Seungwoo Kum</td>
<td>Plan</td>
<td></td>
</tr>
<tr>
<td>03.06.2019</td>
<td>Seungwoo Kum</td>
<td>Draft</td>
<td>Chapter 1, 4</td>
</tr>
<tr>
<td>05.06.2019</td>
<td>Raffaele Giaffreda</td>
<td>Draft</td>
<td>Chapter 5</td>
</tr>
<tr>
<td>11.06.2019</td>
<td>Sofia Kleisarchaki</td>
<td>Draft</td>
<td>Chapter 5</td>
</tr>
<tr>
<td>12.06.2019</td>
<td>Soekhee Lee</td>
<td>Draft</td>
<td>Chapter 3</td>
</tr>
<tr>
<td>13.06.2019</td>
<td>Orlando Avila Garcia</td>
<td>Draft</td>
<td>Chapter 4</td>
</tr>
<tr>
<td>20.06.2019</td>
<td>Raffaele Giaffreda</td>
<td>Draft</td>
<td>Chapter 5</td>
</tr>
<tr>
<td>21.06.2019</td>
<td>Seungwoo Kum</td>
<td>Draft</td>
<td>Modification of Chapter 3, 4</td>
</tr>
<tr>
<td>25.06.2019</td>
<td>Raffaele Giaffreda</td>
<td>Draft</td>
<td>Modification of Chapter 1, 2, 5, Chapter 6</td>
</tr>
<tr>
<td>27.06.2019</td>
<td>Seungwoo Kum</td>
<td>Draft</td>
<td>First Draft</td>
</tr>
<tr>
<td>02.07.2019</td>
<td>Juhyung Park</td>
<td>Draft</td>
<td>Typos corrected</td>
</tr>
<tr>
<td>05.07.2019</td>
<td>Sofia Kleisarchaki</td>
<td>Draft</td>
<td>Review and modification of Chapter 6</td>
</tr>
<tr>
<td>08.07.2019</td>
<td>Raffaele Giaffreda</td>
<td>Draft</td>
<td>Modification of Executive Summary</td>
</tr>
<tr>
<td>09.07.2019</td>
<td>Orlando Avila Garcia</td>
<td>Draft</td>
<td>Minor changes</td>
</tr>
<tr>
<td>09.07.2019</td>
<td>Seungwoo Kum</td>
<td>Final (v1)</td>
<td>Finalization of the document</td>
</tr>
<tr>
<td>06.11.2019</td>
<td>Seungwoo Kum</td>
<td>Draft</td>
<td>Section 2.2.1 added</td>
</tr>
<tr>
<td>14.11.2019</td>
<td>Sofia Kleisarchaki</td>
<td>Draft</td>
<td>Section 5.1 modified</td>
</tr>
<tr>
<td>15.11.2019</td>
<td>Marco Savi, Orlando Avila Garcia</td>
<td>Draft</td>
<td>Minor changes</td>
</tr>
<tr>
<td>29.11.2019</td>
<td>Raffaele Giaffreda, Sofia Kleisarchaki</td>
<td>Draft</td>
<td>Modification of Section 5.1</td>
</tr>
<tr>
<td>02.12.2019</td>
<td>Seungwoo Kum</td>
<td>Final (v1.1)</td>
<td>Finalization of the document</td>
</tr>
</tbody>
</table>

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/](https://www.decenter-project.eu/).
Copyright

This report is © DECENTER Consortium 2018. Its duplication is restricted to the personal use within the Consortium, funding agency and project reviewers.

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 ..................................................................................................................................... 2
MORE INFORMATION ...................................................................................................................................... 2
EXECUTIVE SUMMARY ................................................................................................................................... 5
1 INTRODUCTION ............................................................................................................................................... 6
2 ANALYSIS ON AI DEPLOYMENT FRAMEWORK ............................................................................................ 8
  2.1 CHALLENGES IN AI ON THE EDGE .......................................................................................................... 8
    2.1.1 Virtualization of the AI environment .................................................................................................... 9
    2.1.2 Containerization of AI .......................................................................................................................... 11
    2.1.3 Interfaces of AI containers as microservice .......................................................................................... 12
    2.1.4 Providing inference results ..................................................................................................................... 12
    2.1.5 Computing Resource Utilization .......................................................................................................... 12
    2.1.6 Management of Deployed AI methods .................................................................................................. 13
  2.2 COMPARISON OF AI DEPLOYMENT FRAMEWORK .................................................................................... 13
SOFTWARE SYSTEM AND APPLICATION DEPLOYMENT ..................................................................................... 13
AI SYSTEM AND MODEL DEPLOYMENT ............................................................................................................. 14
DESIRABLE FEATURES FOR AI SYSTEMS OPERATIONS .................................................................................. 15
AI DEPLOYMENT FRAMEWORKS COMPARISON ............................................................................................... 17
  2.3 IDENTIFIED REQUIREMENTS FROM USE CASES ...................................................................................... 19
3 AI ON THE EDGE ............................................................................................................................................. 22
  3.1 SEQUENTIAL PARTITIONING OF NEURAL NETWORK .................................................................................... 22
    3.1.1 Related Works.......................................................................................................................................... 23
    3.1.2 Partitioning of a Deep Neural Network .................................................................................................. 25
  3.2 IMPLEMENTATION AND EXPERIMENTS OF SEQUENTIAL PARTITIONING ON YOLO ................................ 25
  3.3 SURVEY ON SEMI-SUPERVISED LEARNING .............................................................................................. 30
  3.4 REMARKS .................................................................................................................................................... 33
4 CONTAINERIZATION OF AI METHODS ............................................................................................................. 34
  4.1 DESIGN OF THE AI CONTAINER: MODULES AND INTERFACES ............................................................. 34
    4.1.1 Level of containerization ........................................................................................................................ 34
    4.1.2 Structure of AI container ......................................................................................................................... 37
    4.1.3 Package for microservice: decenter.ai ........................................................................................................ 38
  4.2 BUILDING AND DEPLOYING AI SERVICES WITH MICROSERVICES ..................................................... 41
    4.2.1 Operational environment for containers within Kubernetes ..................................................................... 41
GENERAL GOOD PRACTICES FOR DEPLOYING MICROSERVICES ON KUBERNETES ........................................... 42
SPECIFIC GOOD PRACTICES FOR DEPLOYING MICROSERVICES ON DECENTERC ........................................ 43
    4.2.2 Flexible AI service configuration with Kubernetes ................................................................................. 44
  4.3 IMPLEMENTATION ACTIVITIES ..................................................................................................................... 47
    4.3.1 Implementation of DECENTER Package and base container .................................................................. 47
    4.3.2 Implementation of VGG16 container ........................................................................................................ 47
    4.3.3 Implementation of YOLO v3 container ..................................................................................................... 48
  4.4 REMARKS .................................................................................................................................................... 50
5 DIGITAL TWIN REPRESENTATION ..................................................................................................................... 51
  5.1 Digital Twin as a Technological Asset .......................................................................................................... 51
Executive Summary

This document describes the Artificial Intelligence Methods of DECENTER which have been designed and implemented between M4 and M12. To identify issues with deploying AI methods onto the cloud and the edge, existing AI methods and deployment platforms are investigated as preliminary research. Requirements of AI methods on the cloud computing infrastructure have been identified and fed to WP2 during these activities. To fulfill both requirements and objectives derived by DECENTER project, several work items have been chosen and researched.

DECENTER is going to leverage cloud technologies to deploy AI service along with IoT technologies, and to this end, requirements and issues from running AI in the cloud shall be identified first. In Chapter 2 illustrates and defines those requirements have been defined through the identification of issues on AI running on the edge, and describes a comparison of existing AI platforms.

One of DECENTER’s objectives is to place intelligence (i.e. run purposefully designed modules) near the user or the end device, allowing the AI services to have more prompt reaction cycles. Furthermore, placing intelligence at the edge will help to preserve privacy compared to processing data in the cloud, which often exposes upload of all data breaching personal data privacy rules. This delivery of intelligence at the edge is described in Chapter 3 (T4.1).

To leverage the benefits of the Fog Platform for the AI service deployment, a well-defined method is needed to containerize and deploy an AI application, following a modularized design approach where an AI service is made of separated building blocks and appropriate interfaces enabling interactions between those building blocks. The identification of building blocks and containerization method for AI are described in Chapter 4, where uniform interfaces to access AI method functionalities within containerized AI services are presented as well (T4.4).

Chapter 5 presents how Digital Twin representations can exploit AI models derived from interpretation of different data sources and how these models (once containerized) can be mapped-to and deployed-over the underlying orchestrated DECENTER fog platform for improved performance. For achieving these purposes we introduce a framework where concepts, events and situations of real life can be digitally represented. The usefulness of the Digital Twin concept is studied in the context of four different Use Cases derived by WP2 (T4.2).

Chapter 6 analyzes how the Digital Twin can benefit from the reciprocal exchange of data with AI Models. Furthermore, it discusses the role of an IoT platform in building the Digital Twin and illustrates, by an example, the data flow among the Digital Twin, the AI Model and the IoT platform. Chapter 6 gives guidelines on how to equip DECENTER with a standard set of APIs for interfacing as many IoT devices and virtual AI entities as possible. Last but not least, it describes a proposed data model for the Digital Twin and a technical guide of the management tool needed to manipulate IoT data. (T4.2).

Activities on WP4 during M4 and M12 include preliminary implementation of several methods, as well as identification and design of each module. Preliminary implementations and results are given at the end of each chapter, and conclusion of first year activities is given on chapter 7.
1 Introduction

Artificial Intelligence (AI) or Machine Learning (ML) has gone through a lot of improvements in the last few decades. There are now many AI algorithms to be used in various areas, and the accuracy of each model improves every other day. For example, the highest accuracy for image classification based on ImageNet images has already reached an accuracy higher than 90 percent. Those AI algorithms are based on deep neural network, which consists of multiple layers of neurons. That deep neural network is trained with a vast amount of training data set to find an optimal weight of parameters on each neuron, and then validated by applying test set to the trained model. When training is completed, those AI models can be deployed to an AI service. In this life cycle of AI algorithm, computation of deep neural network requires a lot of computing resources, both in training and inference phase. Usually, cloud or hardware-accelerated environment are used to provide such powerful computation resources.

However, from the viewpoint of application or service, this far distance between data source and processing and service endpoint is not so desirable. First, all the data shall be uploaded to the cloud or to other hardware-accelerated devices. This requires a certain level of transmission bandwidth as well as storage, increasing hardware requirements to build a service. Second, this transmission of data degrades QoS. All the data has to be uploaded to the cloud to get the scoring result. Further, the upload of data to cloud arise security and privacy issues. For the AI service with low latency and privacy preservation are required, the use of the edge is essential.

The objectives of DECENTERVER include applying fog computing platform to this AI service environments, in order to make the best use of federated computing resources on IoT-edge-cloud continuum, while guaranteeing QoS on AI services. To this end, WP4 is going to develop a federated AI technology suitable for decentralized infrastructure and make solutions suitable for use cases described in WP2 and demonstrated in WP5. This document describes WP4 activity achievements from M4 to M12 with respect to the design and implementation of the AI methods of DECENTERVER. The structure of this document is as follows: in Chap. 2, analysis of existing AI frameworks is given. The requirements which are derived from the analysis and also from the use cases are described in the same chapter. Each task’s activities are described in dedicated chapters. The descriptions of these activities include their design and implementation, followed by remarks at the end of every chapter.

![Figure 1 Hierarchy of AI-related software entities.](image_url)
To make a clear distinction of the AI-related terms, this document defines the AI-related S/W as follows:

- **AI model**: A computational model which is trained from training data. The AI model is a set of numerical methods and parameters with respect to its architecture. The AI model can be saved to file(s) with serialization.

- **AI method**: AI method refers to a software module which does AI computation for the inference. The input as well as the output of this AI method is a set of numerical values. For example, VGG16 AI method receives an array, which is a result of pre-processing of JPG image and outputs an array which is emitted from the output layer of VGG16.

- **AI service**: An AI service refers to an application which uses AI model with real-world data. It consists of various functionalities including AI method. For example, consider an AI application for image classification with VGG16. This AI service receives an image as input. The input will be resized, and then saved to an array. This array will be fed to the AI method which uses VGG16 model. The output of this AI method will be interpreted given class definition files.

- **AI Application Service**: It refers to an end-to-end service including AI service and other related services such as cloud or IoT. Consider a service, consisting of an IoT, an AI and a GUI: the IoT camera captures a frame for every 10 seconds, the AI service receives those frames, and the GUI service presents the original frames along with all the detected objects in that frame. The communication between each module shall be provided, along with the GUI.

Figure 1 schematically shows the interaction between the aforementioned components.
2 Analysis on AI Deployment Framework

In this chapter, the preliminary analysis of the challenges regarding AI to be deployed at the edge, and the requirements of the AI on DECENTER are given. One of the DECENTER’s novelties include the use of the fog computing platform, which guarantees enhanced QoS by placing application intelligence near data source or service endpoint. This fog computing platform is being designed and implemented in WP3, and it is going to leverage existing cloud technologies such as container and/or Kubernetes to build fog computing platform. The AI methods and solutions, defined in WP4, shall comply with those technologies making the application intelligence suitable to be deployed at any node of DECENTER fog computing platform.

Several considerations exist regarding the use of the fog platform for AI. The procedure of building ML which is usually referred to as AI/ML life cycle consists of data analysis, feature engineering, model training, deployment and maintenance/improvement. DECENTER’s fog platform is going to provide a de-centralized fog platform for AI. With respect to the AI/ML life cycle, DECENTER will provide insights about to where and how the fog platform can be applied to improve the overall performance of the AI services. Further, some light will be shed on which AI platforms, among the numerous already existing, are supported by DECENTER’s orchestrated resources.

2.1 Challenges in AI on the Edge

AI services have become popular due to the advances in machine learning, especially in deep neural networks. Recently, deep neural networks have recently shown performance very close to human experts in selected areas, such as image recognition [1]-[3] and speech recognition [4]. A large cluster of computing nodes or hardware accelerator have become available, and large amounts of data for the training can be processed to train large scale neurons in less time. To increase the accuracy of AI service, neural network size becomes bigger and bigger, and it is quite common to find a neural network model with several hundred megabytes [5]. However, for services where security, privacy and latency are critical, the use of edge computing platform and related hardware devices is essential. Edge computing platform places an edge device, which is capable of processing data by itself, near the data source or service endpoint. These near-data processing eliminates the needs of uploading raw data to the cloud, thus reducing latency, and also better preserving of privacy.

There have been quite a few researches on delivering intelligence from cloud to edge with different approaches reflecting various aspects of technologies when delivering an AI service. Usually, the edge has limited computing resources when compared to cloud resources, and it might not be able to load big-sized models on it. There are several approaches to resolve this issue, including reducing the size of a model to be distributed by pruning of neural network [6] or quantization of neural network parameters [7], and partitioning of a model into several partial models [8], [9]. These approaches help to reduce the size of a model to be loaded at the edge device’s memory, and achieve small footprints on the edge device. Further, they succeed in minimizing energy consumption of it. On the other hand, there are more architectural oriented approaches that deliver intelligence at the edge. ALOHA [10] proposed a framework to deploy a neural network model to an edge. Docker is getting more focus on deploying applications on fog or edge architecture. Deployment of docker on fog platform [11], and edge intelligence framework based on docker [12] have been proposed, which are using docker technology for distribution. Based on these propositions, the issues for intelligence delivery can be categorized into two main streams: how to utilize edge resources, and how to deploy and manage intelligence efficiently.
2.1.1 Virtualization of the AI environment

One of the purposes of DECENTER project is to bring the intelligence near data source, so to resolve issues such as latency, bandwidth and/or security. Methods to deliver intelligence from the cloud to the edge device or fog nodes are requested to acquire this goal. An investigation of existing AI runtime environments has been investigated to find out underlying problems more clearly.

AI application or Deep Learning application runs on a software stack which is composed of various software modules. First it needs a device driver for hardware accelerator such as GPU. This will provide low-level hardware acceleration. And there are libraries to help build application with hardware accelerator, for example, CUDA, cuDNN or OpenCL. Those libraries provide uniform interfaces to process faster computation with hardware accelerator. And there come the deep neural network platforms such as TensorFlow or Keras, and deep learning models and/or applications are built on top of these deep neural network platforms. To move an intelligence from one place to another, these software stacks for the AI application running environment needs to be considered as well. For example, you can move just the application and the model if there is the same running environment on the target or move all the stacks along with those.

The intelligence is going to be delivered from the cloud to the edge device or fog nodes in this project, and this software stack makes it more complex. First, this software stack can be built with various software. Second, the software used in the stack are usually dependent on each other. AI techniques are continuously created and updated, and their related software stacks are updated very frequently with new or deprecated functions. Due to this reason, it is quite common to have requests of specific version of the software stack for an AI application runtime environment. In short, it can be said that an AI application needs a runtime environment with a specific set of software with specific versions for each. In our use cases, it can hardly be assumed that all the nodes have the same runtime environments. Moreover, if a node is going to host more than one AI application with different runtime environments, then the node needs to manage multiple instances of software stack to resolve version conflicts. Managing this kind of issues on device- or node-level is quite inefficient, and isolation of each runtime environment are requested for efficient management of those applications.

This isolation of runtime environments is important not only for delivering AI but also for security and privacy reasons. Without isolation, each application on the same node can access the data being used on other applications. If those data and services are based on private data, such as image or video of a person, there can be a security leak, and this isolation technologies help preventing security leak.

Virtualization technologies can provide this kind of isolated environment for each application. Also, those technologies are very popular in DevOps and being used on the deployment of cloud applications. In here, we investigated two of the major virtualization technologies, Virtual Machine (VM) and Containers, from the AI application’s perspective.

Virtual Machine

Virtual Machine (VM) is defined as “an efficient, isolated duplicate of a real computer machine”\(^1\). It is placed on a host operating system (OS) by a so-called hypervisor, and contains a guest OS in it where applications can run independently from the host OS. It shares computing resources of host device such as CPU, memory and storage. To share a GPU attached to the host OS among guest VMs, a GPU-supporting hypervisor is placed in the host

\(^1\) https://en.wikipedia.org/wiki/Virtual_machine
OS, and each guest OS is provided with a virtual GPU (vGPU). Therefore, applications running on the guest OS can make use of the GPU by means of such a vGPU. GPU(s) in a host can be made available to a VM in pass-through, shared or live migration modes.

**Container**

OS-level virtualization is an operating system paradigm in which the kernel allows the existence of multiple isolated user-space instances, and container refers to those user-space instances. Unlike VMs, Docker containers does not include OS in itself and make use of host OS\(^2\); this is why Containers are a considered a light-weight form of machine virtualization. Also, containers can have access to the GPUs in the host without the need for the vGPU element. Finally, Docker containers are extensible and composable, which means that a new Docker container image can be build by just adding new layers of software configuration to an existing image. This makes this technology specially suitable for packaging software components with their software stack, that is, including in the same deployment unit not only the software component source code and configuration files but also runtime frameworks, libraries, drivers and OS configurations needed. Therefore, this layering mechanism for container image building facilitates developers the reuse and configuration management of the software stack of their application.

**VM vs. Container for AI Application**

![Figure 2 GPU access methods between VM and Container](image)

**Table 1 Comparison of VM vs. Container from AI’s perspective**

<table>
<thead>
<tr>
<th></th>
<th>Isolated</th>
<th>GPU Access</th>
<th>Guest OS</th>
<th>Device Driver</th>
<th>Layering</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>Yes</td>
<td>Yes, with vGPU</td>
<td>Yes</td>
<td>Yes, for vGPU</td>
<td>No</td>
</tr>
<tr>
<td>Container</td>
<td>Yes</td>
<td>Yes, with a device option</td>
<td>No</td>
<td>No.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

---

The comparison of VM and Containers (Docker) is given on table NNN. Even though there are significant differences between using and not using vGPU between them, we were not able to find notable performance degradation using GPU on both of them by literature. Both can provide isolated runtime environment and access of GPU with little degradation of performance. However, Docker Container layered organization can help AI application developers to take advantage of the common software stack structures and software lifecycles (retrieve data - apply logic - interpret result) shared by the AI applications. Moreover, containers are more lightweight, more portable and faster to start with respect to Virtual Machine: all these characteristics are of paramount importance in the case of heterogeneous and highly distributed computing infrastructures like the ones foreseen by in decentralized AI over a fog computing paradigm.

**Microservices Architecture**

Microservices architecture is an architectural style that arranges a software application as a collection of loosely coupled services\(^3\). Microservices architecture is a very effective and efficient way of consuming compute, storage and networking resources and managing the software lifecycle of the applications making use of them. Let's take as an example a hypothetical face detection application service; suppose this service retrieves a video stream from an IP camera and analyzes it by applying a deep learning model (trained to detect faces in an image) to each video frame. When built as a single application in a monolithic fashion, all the functions or components are deployed together, in a single application package-i.e. to receive video stream, to process each frame, to run the deep learning model, etc. If you want to change the application to receive images from a source different than a video stream, you need to re-build and re-deploy the whole application. With a microservices architecture, the application can be split into different microservices (one per function or feature) with independent software lifecycles. For the previous example, we can imagine two microservices: a microservice for retrieving a video stream from an IP camera, and another one for serving the deep learning model (i.e. running the model inference). This application can be changed to receive any input by simply replacing the first microservice with another that is able to be fed with a different data source. Moreover, the partition of the application into these small building blocks with independent integration, deployment, monitoring and provisioning cycles make it simpler and more efficient to manage the application's reliability (e.g. fault tolerance and availability) and scalability; this is because the different microservices can be independently healed (restarted), upgraded (rolled-out) and scaled horizontally (replicated) as well as vertically. More importantly, the application deployment can be shrunk at any moment by scaling in and/or down idle or underused microservices. This performance attribute is called elastic escalability and it is really difficult (or inefficient, at most) to achieve in case of a monolithic application.

2.1.2 Containerization of AI

To bring intelligence to the edge on DECENTER fog computing platform, the AI needs to be containerized since DECENTER's fog computing platform leverages docker and Kubernetes for resource management. Before describing the containerization of AI, the components of an AI need to be checked first. Current AI algorithm development environments consist of:

- Machine Learning Library: This library provides various functionalities to build an ML algorithm. Examples of this library are TensorFlow, Keras, Caffe and Theano.
- AI models: Neural network models which have been trained with training data set.

\(^3\) [https://en.wikipedia.org/wiki/Microservices](https://en.wikipedia.org/wiki/Microservices)
If hardware accelerator such as GPU is involved, the following components shall be added to the environment.

- **Computing Resources:** AI is based on ML algorithms, which requires lots of computing resources. There are processors which are suitable for ML such as GPU or NPU. Those processors support high-speed parallel processing of data, resulting in a faster computation of ML.

- **Device Driver:** When GPU or NPU is used for ML, those processors are recognized as devices, and the proper device drivers are required.

- **Parallel computing platform:** This is a platform which supports parallel processing of data using hardware accelerator such as GPU. CUDA or OpenCL is two of the most popular parallel computing.

The first two parts can be achieved easily. There are already many AI/ML platforms which support docker. For example, TensorFlow and Caffe provide official docker image which can be used for building AI models. However, there are several issues to be considered when it comes to the other part which is related to a hardware accelerator. From the viewpoint of docker, a hardware accelerator is a device in the host, and need to be mapped to the container properly so that the containerized application can access that specific host device of the host. Among the latter three parts, only the last one (parallel computing platform) can be containerized, with an appropriate mapping of GPU device in the host.

### 2.1.3 Interfaces of AI containers as microservice

Even though there is a docker already, using them as a microservice is not sufficient. These official containers provided by AI/ML platforms, mainly support a development environment for building an AI model, and not a service development environment. To implement an AI model as a service, the developer needs to implement appropriate interfaces for the microservice. However, there is no formal methodology for implementing those interfaces yet. This means that the interfaces to access an AI model as a microservice will vary from one implementation to another, and this will reduce the productivity of building a service capable of utilizing an AI service. To make the best use of an AI container, uniform interfaces to access containerized AI methods needs to be defined, based on the common data flow of neural network computation.

### 2.1.4 Providing inference results

The output of applying source data to the neural network model is an array which is emitted from the output layer of the model. This array needs to be interpreted in order to get meaningful data from it. As applications for the AI/ML applications become diverse, it needs a lot of effort for the interpretation of output data. For example, top-1 or top-5 classes are needed for image classification, while more additional data, including bounding box position, are needed for object detection. Moreover, if someone wants to develop an object detection service based on containerized AI and needs to show the detected object with its bounding box on the input video stream, s/he shall be able to draw the overlaying bounding box exactly on the frame which has been previously analysed. These applications put more restrictions and additional requirements on containerization of AI/ML methods.

### 2.1.5 Computing Resource Utilization

In DECENTER, AI models will be containerized and then deployed to the cloud or to the edge device. In this case, special care should be taken to match computing resource requirements to those of the actual device. There are already terms to request a computational resource,
such as CPU work time or memory size for an application using Docker or Kubernetes. However, those terms are not sufficient when it comes to hardware accelerator for AI/ML computation. Hardware accelerator for AI/ML computation exists as a specific device, and to meet the QoS and more resource utilization, it needs methods are required to express the computational resource for hardware accelerator, the measures for the utilization, and the methods to actually measure them.

2.1.6 Management of Deployed AI methods

After the AI/ML models are successfully built and deployed, the deployed model on the fog platform shall be managed properly. From the viewpoint of the cloud technology, this management includes monitoring of the AI methods, and load balancing of resources. From the viewpoint of AI, this management includes updating of AI methods which have already been deployed. For example, if one model is updated to provide a better performance, all the containers using that model shall be able to receive the updated model. This requires a semantic definition of metadata for a model, serialization methods for a model, and methods to updated model remotely in a secure way.

2.2 Comparison of AI deployment framework

Software system and application deployment

DevOps (development and operations) is an enterprise software development paradigm which proposes a close and agile relationship between software development and IT operations. Its main feature is a continuous and smooth flow of software changes from development into production as well as continuous feedback from operations to development. This paradigm aims to avoid the long, error-prone, large-scale deployments which characterised the era when developers and IT system administrators belonged to different organizations within the same company.

DevOps is characterised by the streamlining and automation of the software delivery process, that is, the set of activities required to make a software system or application available for use. Within this paradigm, developers are provided with a toolset to automate the provisioning (reservation, configuration) of the IT environment or infrastructure (including compute nodes, storage and networking resources), and the integration, testing, deployment, validation, monitoring and optimization of the application running in it. The full automation of this process is known as Continuous Deployment (CD) and has become one of the star capabilities for software development organizations in the digital age.

Figure 3 shows the continuous flow of activities in a DevOps software system or application life cycle. Activities such as planning (requirements, architecture, design), coding (programming), building (integration) and testing are considered part of the Dev phase; they are executed for every change or feature the software product needs to incorporate. The Ops phase is comprised of those activities which bring the integrated and tested piece of software into production (i.e. available to the final user). This starts with the release (configuration management) of the software component; it is followed by the deployment (i.e. installation and configuration) and operation (run) on the IT environment; and finalized with the monitoring of its performance. The DevOps loop closes with application performance metrics and alerts (e.g. QoS) feeding back into the Dev phase to trigger a new cycle.
Figure 3 Continuous flow of activities in a DevOps software life cycle. The activities of the Dev phase (in dark blue) are followed by the activities of the Ops phase (light blue).

**AI system and model deployment**

Like in standard software system or application life cycle, AI system or application life cycle makes a clear distinction between development (Dev) and operations (Ops) phases. AI development comprises a set of activities to go from raw data to models for solving complex functions, such as perception, prediction, decision-making or actuation in complex, dynamic and uncertain environments. Depending on the nature of the AI method used, we will elucidate about machine learning (ML)’s statistical and probabilistic (e.g. Bayesian) aspect, along with its models.

The AI system Ops phase includes activities such as model versioning (configuration management), model deployment on a suitable infrastructure or IT environment, model operation or run (serving model inferences, e.g. predictions or prescriptions), and model performance monitoring to drive changes (optimizations) to the AI system. Notice the enormous similarities between this cycle and the DevOps cycle shown in Figure 3.

Figure 4 shows the details of the continuous flow of activities within the DevOps cycle of an AI system or application. Data ingestion (collection), analysis, transformation and validation aim to prepare the data to be fed into the AI method, for example, a given certain ML technique. The enactment of an AI method in the so-called Model Building activity (called model training in the case of ML techniques) results in a new model. This activity closes the Dev phase. The Ops phase starts with model management tasks such as model validation, auditing, versioning, cataloging and storage. Then, the activity to deploy the model in a given IT environment follows. Once the model is deployed, the actual operation (serving), as well as its performance monitoring, will be carried out. Finally, depending on the performance results, an optimization activity may also be carried out to correct (enhance) the AI system performance.
Kubeflow is an ML toolkit for Kubernetes\(^4\) aiming to facilitate the DevOps (development and operation) of simple, portable and scalable ML workflows or pipelines. It integrates nicely with Kubernetes both conceptually and technically. This integration lets Kubeflow to be independent of the underlying infrastructure up to some extent; this lets developers deploy the same workflow to different IT infrastructure models, from public Infrastructure-as-a-Service (IaaS) and public managed Kubernetes services (such as EKS or GKE) to a virtualized data center, including the hybrid-cloud and multi-cloud scenarios.

Kubeflow has been conceived to specifically support the kind of workflows that arise in the DevOps of AI systems and applications. To do so, it aggregates many best-of-breed open-source frameworks, libraries and tools to support the different activities and tasks in the life cycle. To support the Ops phase, Kubeflow integrates several ML deployment frameworks, which provide the developers with an increasing number of features for Model Management, Deployment, Serving, Monitoring and Optimization.

Desirable features for AI systems operations

We have identified a series of desirable features for the five activities which comprise the AI system operation phase: Model Management, Deployment, Serving, Monitoring and Optimization. We will use them as the set of criteria to evaluate AI deployment frameworks.

1. **Model Management**
   a. Validation.
   b. Auditing: assurance and certification for its use on safety-critical scenarios.
   c. Versioning.
   d. Storage: offering of a model repository.
   e. Data provenance: certifying and validating the data used to build the model.
   f. Quality of Service (QoS): support for quality assurance, Service-Level Agreements (SLA) and Service-Level Objectives (SLO), etc.
   g. Security: ensuring model integrity, model access control (authorization), etc.

2. **Deployment**
   a. Containerization.
   b. Deployment: installation and configuration of a piece of software on a compute node.
   c. Scaling: provisioning of infrastructure resources (including compute, storage and networking) in the first deployment as well as during operations. Includes scaling up and down as well as out and in (elastic scalability).

\(^4\) https://www.kubeflow.org/docs/about/kubeflow/
d. Updates: supporting different deployment strategies for changes on the model, for example rolling, canary, blue/green and shadow deployments.

e. Launch or run the model serving process/es.

f. Health checking.

g. Recovery: implies the automated healing (typically, restart) of the unhealthy serving process/es.

3. Serving

a. Cross-ML/AI: support to different ML/AI engines or runtimes for example, TensorFlow\(^5\), TensorFlow Lite\(^6\), TensorRT\(^7\), PyTorch\(^8\), Scikit-learn\(^9\), H2O\(^10\) or Spark\(^11\).

b. Acceleration: support for GPUs, FPGAs, TPUs.

c. API: support for REST, gRPC, etc. protocols.

d. Connectivity: offering of a reverse proxy implementing an API Gateway and/or load balancer.

e. Messaging: support to synchronous vs asynchronous inference requests, and streaming vs batch inference requests.

f. Security: ensuring inference access control, including both authentication and authorization (e.g. OAuth).

4. Monitoring

a. Logging: support for tracing good practices and/or standards.

b. Metrics: support to a standard set of performance efficiency indicators, such as latency, throughput or memory footprint, as well as functional adequacy indicators, such accuracy, precision, recall, F1, etc.

c. Visualization

d. Alerting

e. Probes for different model Serving features.

5. Optimization

a. Parallelism: supporting simultaneous deployment and run of several models.

b. Inference orchestration: implies the support for multiple processes within the AI system which collaborate to deliver the model inferences.

i. Request routing: to optimize (comparing) between models at inference time; support to standard traffic optimization techniques such as A/B testing or Multi-Armed Bandits.

ii. Ensemble response: to combine different models to produce a single inference (ensemble models).

iii. Request transformation: to preprocessing incoming (request) data for data cleaning, feature normalization, security, etc.

iv. Response transformation: to post-processing the outgoing (result or inference) data for outlier detection, concept drift detection, etc.

\(^5\) https://www.tensorflow.org/
\(^6\) https://www.tensorflow.org/lite
\(^7\) https://developer.nvidia.com/tensorrt
\(^8\) https://pytorch.org/
\(^9\) https://scikit-learn.org
\(^10\) https://www.h2o.ai/
\(^11\) https://spark.apache.org/
c. Deployment adaptation: implies changes in the deployment configuration of the AI system to enhance its performance efficiency as well as functional adequacy indicators.

**AI deployment frameworks comparison**

We consider in this comparison the three main deployment frameworks supported by and integrated with Kubeflow: TensorFlow Serving\(^{12}\), Seldon-core\(^{13}\) and TensorRT Inference Server\(^{14}\).

TensorFlow Serving\(^{15}\) is one of the most popular toolkits to deploy and run models based on TensorFlow\(^{16}\), one of the most popular Deep Learning frameworks. However, it can be easily extended to serve other types of ML/AI runtimes. The central abstraction is the *servable*, which are underlying objects that clients use to perform computation (for example, a lookup or inference). They represent processes with flexible size and granularity running within the same serving facility.

Seldon-core\(^{17}\) provides deployment for any ML runtime that can be packaged in a Docker container. Its main abstraction is the *inference graph*, which chains a group of processes together within the model serving facility. These processes can be of different types: Models, Routers (e.g. for A-B testing Multi-Armed Bandits), Combiners (e.g. for model Ensembles) and Transformers (e.g. for feature normalisation or outlier detection). Seldon-core provides out-of-the-box implementations of those building blocks; but more importantly, users can create their own components and place them in the runtime inference graphs.

Figure 5 shows an example of complex runtime graphs for model inference to be deployed through Seldom-core. In it, several standard components, a transformer for outlier detection and a multi-armed bandit optimization for optimization (router), are deployed together with user-specific components in the same inference graph. In this example, the whole graph is deployed as a Kubernetes pod with each component (graph node) in an individual container.

---

\(^{12}\) [https://www.kubeflow.org/docs/components/tfserving_new/](https://www.kubeflow.org/docs/components/tfserving_new/)

\(^{13}\) [https://www.kubeflow.org/docs/components/seldon/](https://www.kubeflow.org/docs/components/seldon/)

\(^{14}\) [https://www.kubeflow.org/docs/components/trtinferencesserver/](https://www.kubeflow.org/docs/components/trtinferencesserver/)

\(^{15}\) [https://www.tensorflow.org/tfx/guide/serving](https://www.tensorflow.org/tfx/guide/serving)

\(^{16}\) [https://www.tensorflow.org/](https://www.tensorflow.org/)

\(^{17}\) [https://www.kubeflow.org/docs/components/seldon/](https://www.kubeflow.org/docs/components/seldon/)

\(^{18}\) Extracted from [https://es.slideshare.net/seldon_io/seldon-deploying-models-at-scale](https://es.slideshare.net/seldon_io/seldon-deploying-models-at-scale), slide 29.
NVIDIA TensorRT Inference Server\(^\text{19}\) is a serving facility for popular Deep Learning runtimes: TensorRT, TensorFlow and Caffe2. The server is optimized to deploy models on both GPUs and CPUs at scale. The actual inference server is packaged within the TensorRT Inference Server container\(^\text{20}\).

Table 2 shows the results of the comparison of these three deployment frameworks supported by Kubeflow, with respect to their relative support to desirable features for AI systems operations.

**Table 2 Comparison of the three main deployment frameworks supported by Kubeflow.**

<table>
<thead>
<tr>
<th>Management</th>
<th>TensorFlow Serving</th>
<th>TensorRT Inference Server</th>
<th>Seldon-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Auditing</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Versioning</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Storage</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Data provenance</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QoS</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Security</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deployment</th>
<th>TensorFlow Serving</th>
<th>TensorRT Inference Server</th>
<th>Seldon-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containerization</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Deployment</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Scaling</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Updates</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Launch</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Health checking</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Recovery</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Serving</th>
<th>TensorFlow Serving</th>
<th>TensorRT Inference Server</th>
<th>Seldon-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiples ML/AI</td>
<td>5</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Acceleration</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>API</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Connectivity</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Messaging</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Security</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monitoring</th>
<th>TensorFlow Serving</th>
<th>TensorRT Inference Server</th>
<th>Seldon-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Metrics</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Visualization</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Alerting</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Integrations</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Probes</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelism</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Inference orchestration</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Deployment adaptation</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\(^\text{19}\) [https://www.kubeflow.org/docs/components/trtinferenceserver/](https://www.kubeflow.org/docs/components/trtinferenceserver/)
2.3 Identified requirements from use cases

This section describes the identified requirements of AI methods and solution. With respect to use case, the identified issues in the previous sections and the relationship with DECENTER Fog Platform (WP3), those requirements are grouped in three categories: requirements related to containerization of AI method, requirements related to resources for orchestrated with fog platform, and requirements related to deployment and management. Table 3 presents description of the main AI methods under consideration from each use case, and Table 4 presents the requirements categorized in those three categories. Bold text within the ID column of the table indicates that the activities took place between months M4 and M12.

Table 3 Main AI Methods on DECENTER Use cases

<table>
<thead>
<tr>
<th>UC</th>
<th>Short Description</th>
<th>Input Type</th>
<th>Roles of AI Methods</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Event detection in crossing</td>
<td>Video stream</td>
<td>Object detection from video</td>
<td>Pedestrians presence, Vehicle (type, direction and speed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Audio stream</td>
<td>Event detection from sound</td>
<td>Object type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IoT sensors</td>
<td>IoT data analysis</td>
<td>Environmental conditions (temperature, humidity, light, rain)</td>
</tr>
<tr>
<td></td>
<td>Analysis results</td>
<td>ensemble</td>
<td></td>
<td>Prediction of road event</td>
</tr>
<tr>
<td>2</td>
<td>Smart Logistics - optimal path update</td>
<td>Video stream or sensor data</td>
<td>Obstacle identification</td>
<td>Type of obstacle: person, robot, anything else</td>
</tr>
<tr>
<td>3</td>
<td>Smart Construction site</td>
<td>Video stream</td>
<td>Detection of Vehicle type, colour and/or license plate number</td>
<td>Vehicle information</td>
</tr>
<tr>
<td></td>
<td>Video stream</td>
<td>Detection of worker's safety gear</td>
<td>Worker's safety gear status</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Video stream</td>
<td>Object detection from video</td>
<td>Quantity of asses, tools and waste</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Member verification</td>
<td>Video Stream</td>
<td>Face Detection</td>
<td>Image region that contains face</td>
</tr>
<tr>
<td></td>
<td>Image</td>
<td>Member verification</td>
<td>Binary result - whether the detected faces are member(s) of pre-registered group</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>ID</td>
<td>Requirements for AI application to Fog Platform</td>
<td>Target</td>
<td>Related UCs</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>Containerization of AI specific resources and orchestration</td>
<td>RAI-01</td>
<td><em>AI Application shall be able to be containerized.</em> <strong>Description:</strong> 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.</td>
<td>AI method and solutions</td>
<td>UC1, UC2, UC3, UC4</td>
</tr>
<tr>
<td></td>
<td>RAI-02</td>
<td><em>Containerized AI application shall be able to access H/W acceleration devices such as GPU and its memory.</em> <strong>Description:</strong> 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.</td>
<td>AI method and solutions</td>
<td>For all use cases which uses GPU (UC1, UC3, UC4)</td>
</tr>
<tr>
<td></td>
<td>RAI-03</td>
<td><em>Containerized AI shall be able to be accessed using uniform interfaces.</em> <strong>Description:</strong> 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.</td>
<td>AI method and solutions</td>
<td>UC1, UC2, UC3, UC4</td>
</tr>
<tr>
<td></td>
<td>RAI-04</td>
<td><em>Resources for AI application (or Service) shall be assigned properly and be managed throughout the lifetime of AI application.</em> <strong>Description:</strong> 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.</td>
<td>AI method and solutions, Fog Computing</td>
<td>UC1, UC2, UC3, UC4</td>
</tr>
<tr>
<td></td>
<td>RAI-05</td>
<td><em>Node with H/W acceleration shall expose its H/W acceleration resources for resource orchestration.</em> <strong>Description:</strong> 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)</td>
<td>AI methods and solutions</td>
<td>For all use cases which uses GPU (UC1, UC3, UC4)</td>
</tr>
</tbody>
</table>
| 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 | For all use cases which uses GPU (UC1, UC3, UC4) |
| 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 | UC2, UC3, UC4 |
| 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 | UC2, UC3, UC4 |
| 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 | UC1, UC3, UC4 |
| 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 | UC4 |
| 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 application, each application shall be able to send and receive information from each other. | Fog Computing | UC1, UC2, UC3, UC4 |
| RAI-12 |  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 | UC2, UC4 |
3 AI on the Edge

With the advent of CPU / GPU technologies and ML platforms, AI has gained attention over the last few decades. Starting from CNN [13], which emerged on 2012, several ML algorithms based on neural networks have been proposed in a wide area; from IoT data analysis to more complex tasks such as audio or video analysis. Advancement in platforms such as Google’s TensorFlow, Keras or Caffe helped developers to implement their own neural network in a simple and quick way. Nowadays, AI becomes very popular with the help of both hardware and software. The last decade these technologies can be seen, not only in exhibitions, but also in real-life deployments. AI speakers finally understand the natural language, the photo management application knows where the picture is taken, who and what are inside the picture. These kind of AI services are mostly dependent on a neural network, which is defined as a network of neurons. These neural networks can be trained with vast amount of data, and their accuracy is getting higher and higher over time. However, usually lots of computational resources are required for training and inferencing from these neural networks. Moreover, the implementation of an AI service usually takes place at the cloud or at a device with GPU where developers can take benefits from rich resources. This rich-resource requirement has imposed a restriction to the AI service architecture: it requires to be implemented at the cloud or at a specific device. This restriction results in increasing the bandwidth to transmit data and also induces delay to the AI service. As described in the previous chapter, there are many researches to overcome this restriction by applying edge-device concept such as network pruning, quantization of network parameters and partitioning of AI models.

DECENTER’s goal on AI includes a) bringing the intelligence near to the device or to the service endpoint, in order to ensure prompt response from the AI service, and b) provide suitable methods for scalability of AI methods deployed at multiple edges. During the first year of the project, WP4’s activities were focused on providing scalable AI methods at the edges and analyzing the effectiveness of the proposed method. These activities include the design and the implementation of AI partitioning to make it suitable to be deployed at the Fog, and survey of utilizing edge for semi-supervised learning. This chapter is organized as follows: neural network partitioning is described in section 3.1 and description of its implementation is given in section 3.2. Section 3.3 presents a survey on semi-supervised learning and remarks about these activities are followed.

3.1 Sequential Partitioning of neural network

The AI methods in DECENTER can take benefit from the de-centralized nature of DECENTER’s Fog Platform, developed in WP3. In DECENTER, computational resources are managed by Fog Platform, regardless of their node type or resource owner. Thus, more flexible configurations of AI can be provided. Comparing to the conventional AI which usually resides on the cloud environment, DECENTER’s AI can be deployed on any node (cloud or edge). Moreover, this de-centralized nature enables placing a single AI service across multiple nodes. For example, if individual computational resources are not powerful enough to load an AI service, the AI model can be partitioned into multiple parts so that these partial-networks can be placed at sequentially connected nodes to provide a single AI service. This sequential partitioning of a neural network has several benefits comparing to conventional monolithic neural network model. Below we discuss some of those benefits:

- Make best use of computational resources;

If there is not a single node within a cluster which can provide enough computational resources for a specific neural network, there is no way to provide AI service with monolithic neural network. A lightweight model, such as a model with pruned network or quantized parameter,
can be used on devices with limited-resources. However, this modification of neural network will affect the accuracy of the AI service. In DECENTER, the neural network will be partitioned sequentially, so that each model can be deployed to the nodes of less computational resources without the AI Service losing any accuracy. Last but not least, the effect of model distribution will be investigated.

- Flexible configuration of neural network computation between nodes;

With the de-centralized nature of DECENTER’s Fog Platform, the neural network computation can be assigned to computational resources, regardless of their characteristics such as location or vendor. By applying sequential partitioning of a neural network to the Fog Platform, a very flexible configuration of the neural network computation (or AI service based on it) can be achieved. The configuration will allow, not only the deployment of the partitioned neural network models at the edges, but also their deployment across edge and cloud nodes. For example, a part of the neural network, which processes input data, can be placed at an edge near data source, and other parts of neural network can be placed at the cloud where more powerful resources can be utilized.

- Can help privacy issues;

With monolithic neural network model at the cloud, all the data to be analysed shall be uploaded to the cloud and be processed there. This upload of raw data might threaten privacy, since these data might contain private information, such as device log, audio and video recording of a user. There are many security measures developed for the privacy preservation on cloud networking, such as authorization and encryption. The sequential partitioning of neural network might help privacy preservation as well. Under sequentially partitioned neural network, raw data will be fed to the input layer of the first partitioned model, and the intermediate data will be fed from the output of the first partitioned model to the input layer of the second partitioned model. With this distributed AI model, only processed data which contain less (or no) private information will be transferred.

- Efficient deployment of transfer-learned networks;

Transfer learning is one of the deep learning methods where a pre-trained model is reused as the starting point of the second model development (instead of training the whole network from the scratch). Transfer learning has several implementation methods, and one of them is to re-train only the last layer (output layer) of the network. In this case, the original network and transfer-learned network has all the layers in common except for the last layer. In DECENTER, instead of placing the entire two neural networks in each node, the common part will be placed in one node and will be connected to each other node using different output layer. Thus, only one resource with powerful computing is needed.

3.1.1 Related Works

In this section, a survey of related work is presenting, focusing on balancing the trade-offs especially by compressing the feature maps are presented in this section. There are few works related with feature map compression. The brief and basic explanation of feature compression is explained in this section based on [15]. The recent works have well organized the basic pipeline of compressing the feature map and they propose advanced ways of compressing it by combining machine learning knowledge. The work in [15] suggests breaking down the feature map compression into three steps. The proposed pipeline sparsifies, quantizes and encodes (entropy encoding) feature map, and it roughly complies with general algorithms of feature compression.
Before we get into the first step, the feature map generation needs to be clarified. The recent deep learning network consists of a convolutional layer, non-linear activation function, pooling. The non-linear activation function is used in deep learning to allow convolutional neural networks, which are linear computations, to grasp some non-linearity. One of the non-linear activation functions used in the majority of modern CNN architectures is ReLU (Rectified Linear Unit) shown in the equation below. The function induces non-linearity and achieves fast execution time for both training and testing since it does not require heavy computations. It also allows the network to achieve the sparsity of feature maps by turning into zero any value below zero.

\[
f(x) = \max(0, x)
\]

In sparsification step, the feature map is sparsified by increasing the number of zeros in the feature map using an activation function. While ReLU is a simple strategy to induce sparsity, the authors in [15] proposed an advanced idea of adding a loss function to generate sparser feature maps based on ReLU. The proposed loss functions, \( E(w) \), is shown below.

\[
E(w) = E_0(w) + \frac{1}{N} \sum_{n=1}^{N} \sum_{l=0}^{L} \alpha_l \| x_{l,n} \|_1
\]

Where \( E_0(w) \) is some loss function used for the task and for regularization, \( n \) is the index of the training sample, \( N \) is the mini-batch size, \( l \) is the index of the layer, \( x_{l,n} \in \mathbb{R}^{H \times W \times C_l} \) denotes the feature map of the training sample \( n \) at layer \( l \) after the activation function. The equation uses the L1 norm so that the network learns to generate the sparser feature map at every layer. It is proven that the norm is the relaxation of L0 norm, which decreases the number of non-zero values [16].

The second step is quantization of the floating point of the feature map, and it is a simple but effective technique for reducing the size of the feature map. The equation to quantize the feature map down to \( q \) bits is the following.

\[
x_{i}^{\text{quant}} = \frac{x_i - x_{i}^{\text{min}}}{x_{i}^{\max} - x_{i}^{\text{min}}} \times (2^q - 1)
\]

Where \( x_i \) is a value in the feature map of layer \( l \), \( x_{i}^{\max} \) and \( x_{i}^{\text{min}} \) are the maximum and the minimum value of \( x_i \) in the feature map inside the training set. \( x_{i}^{\text{min}} \) is 0 for this case since the network uses ReLU, which clips negative values to 0. But in the testing environment, \( x_i \) can pass \( x_{i}^{\max} \), and for this case, the value is clipped to \( x_{i}^{\max} \). [17] showed that retraining the model after quantization could improve performance.

The last step is entropy coding, which is a lossless data compression. The typical and simple representations are a) compressed sparse row (CSR) and b) compressed sparse column (CSC), which balances between efficiency and compression. The paper [15] proposed sparse-exponential-Golomb (SEG), which is based on effective-Golomb (EG)[18]. The algorithm allows to compress the feature map in on-line fashion, whereas CSR and CSC require the feature map or the matrix prior to the execution.

The techniques described above are used to perform experiments over the initial version of sequential partitioning network and to implement simple but promising feature map compression. Further details of the experiments and the implementation are written in the next section.
3.1.2 Partitioning of a Deep Neural Network

The neural network consists of layers, which consist of neurons. The idea of sequential partitioning of the neural network is to split a neural network into multiple parts, place them sequentially, and transfer intermediate data from output layer of each partitioned model to the input layer of the next partitioned model.

Figure 6 shows the concept of partitioning simple DNN into two parts. A DNN model has been partitioned into two parts, and intermediate data between each partitioned model are transferred from the output layer to the input layer of the next partitioned model. In partitioning a model into multiple parts, selecting where to partition becomes important, since it affects the overall performance of the neural network model. Since intermediate feature maps need to be transmitted from one node to the other, the size of feature maps is a key factor to consider. The size of the feature map will increase if the point of the partition is poorly chosen, and the entire process of executing an AI algorithm would take a long time as the size of the feature map increases. The impact of where to split a DNN has been investigated in [2], and it shows that the selection of a layer affects the size of the feature map and the overall performance of DNN.

A smaller feature map size is not always preferred. If not enough size is used to encode the given input information, the feature map will not be able to encode the information or to represent in the limited feature space to process in the latter nodes. This would result in decreasing the performance of the AI. For instance, it can result in a low accuracy of the classification task. For this reason, balancing the trade-off between the size of the feature map and the performance is required.

This section described a way of how sequential partitioning is implemented for DECENTER and tested based on the knowledge of feature map compression. For the experiment, two models, namely YOLO v1 [19] and YOLO v3 [20] for object detection are chosen and tested on the effect of partitioning. Each model is partitioned into two parts, at the layer where small size intermediate data are expected and tested with original model to see the effect of partitioning.

3.2 Implementation and experiments of sequential partitioning on YOLO

We have implemented the sequential partitioning AI algorithm and performed experiments especially on You Only Look Once model, which is well known as YOLO version 3. However,
the size of the intermediate feature maps, which need to be transmitted between nodes, were larger than our expectation. We implemented the simplified initial version of the feature map compression algorithms described in the previous section in order to reduce the size of the feature map. The further details of the implementation and the experiments are described below.

In the beginning, we used YOLO since it runs fast due to its simple structure, and it satisfies the exact characteristics of our project needs. YOLO is a popular object detection algorithm. It is a unified object detection network, which does not require region proposals in prior, whereas the former works such as Fast R-CNN [21] and Faster R-CNN [22] require region proposals with the use of external object proposal algorithm or internal proposal algorithm. This simple structure allows the network to run fast. The authors of YOLO algorithm dealt with the object detection problem as a regression problem. A convolutional neural network structure can directly predict the bounding box and the class probability from the input image.

![Figure 7 Overall architecture of YOLO and partitioning point.](image)

The simplified architecture of YOLO is shown in Figure 7, and the red dotted line corresponds to the point where the network is split into two networks. The network is split by taking into account of balancing between the size of the feature map and the computational loads. We did not apply any feature map compression techniques in this stage.

The original intermediate feature map is a 7x7x1024 matrix represented in 32 bits and float type, and the size is about 200KB. The size of the feature map is reasonably small, but the result of the detection is not satisfiable and below our expectation as shown in Figure 8. The algorithm fails to distinguish two closely located small objects. It is one of the drawbacks of YOLO version 1 due to its lightweight and simple architecture, and it is not suitable for detecting small objects.
For this reason, we have switched the network to YOLO version 3, which has more powerful performance than its formers. YOLO version 3 has hourglass-like structure that generates the feature map in the middle to have the most contextual information at cost of losing spatial resolutions. As the feature map gets larger, it retains spatial resolution while keeping the rich contextual information by combining the feature maps in the middle of the network and the feature maps generated in earlier stages. These earlier stages have rich spatial information, but lack of contextual information. The architecture is shown in Figure 9, and also the red dotted line indicates the splitting point.

We have split the network in the middle since it only needs to transmit three small feature maps and it balances well the computational loads. Three feature maps are created; a 52x52x256 matrix, a 26x26x512 matrix, and a 13x13x1024 matrix. The size of the unmodified feature maps is 4,377KB (about 4MB), which is extremely large to transmit between nodes.
The feature compression techniques from the previous section are used to unravel the problem. We performed three steps to reduce the size of the intermediate features: modification in activation function, quantization, and entropy coding. The overview of the entire process is depicted in Figure 10.

**Figure 10 Overview of feature map compression in sequential partitioning of neural network.**

The first step is to modify the last activation function at the transmitter node. Originally, YOLO version 3 uses leaky ReLU in all layers, and it is one of the well-known methods to avoid “dead ReLU” problem where the majority of values, after applying ReLU, becomes zero. It may benefit the performance of the network, however, it generates dense matrix, which has mostly non-zero values. Replacing leaky ReLU to ReLU, only at the end of the first part of the splitted network, induces the sparsity of the intermediate feature map at the cost of the performance since it is simply discarding the negative values.

The second step is quantization of the floating point in the intermediate feature map, and this is a simple but effective technique for reducing the size of the feature map. The equation for feature map quantization is shown below. For the experiments, the transmitted feature map is quantized into unsigned integer of 8 bits. The maximum threshold value and the minimum threshold value of the feature map is empirically selected and fixed in both networks. The values above the maximum threshold are clipped before passing to the second network. In the second network, the received feature map is dequantized based on the maximum and the minimum threshold used in the first network. Then, the retrieved feature map proceeds through the rest of the network computation.

\[ x_i^{\text{quant}} = \frac{x_i - x_i^{\text{min}}}{x_i^{\text{max}} - x_i^{\text{min}}} \times (2^q - 1) \]

The last step is entropy coding, which is compressing the actual feature map using entropy coding. For our experiments, the transmitted feature map is compressed using the deflate algorithm provided by NumPy library [23].

The result of the experiment is shown in Table 5. The original result column refers to the compression result when using entropy coding. The ‘Modification in the activation function’ column shows the result after the modification of ReLU with entropy coding. Last but not least, the ‘Final compression result’ column is the result using all the aforementioned techniques. The size of the feature map from the final result reduced by x7 compared to the original result. The modification in the activation function sparsified 37 percent of the feature map.
Table 5 Result of original YOLO v3, modification and final compression.

<table>
<thead>
<tr>
<th></th>
<th>Original result</th>
<th>Modification in activation function</th>
<th>Final compression result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature size</td>
<td>4,377 KB</td>
<td>2,799 KB</td>
<td>598 KB</td>
</tr>
<tr>
<td>Data type</td>
<td>Float 32</td>
<td>Float 32</td>
<td>UINT8</td>
</tr>
</tbody>
</table>

The result of reduction in the size was effective, but the performance of the splitted network compared to the original one needs to be considered. The detection result of the original network and the result from the final compression are shown in Figure 11. The result misses tiny objects in the given input image, but it is still able to retrieve most of the object detected in the original network. Further, the processed result showed far better performance compared with the result from YOLO v1 in Figure 8.

Figure 11 Comparison of results between before (left) and after (right) feature map compression

These are results from the initial experiments, and there is still room for improvement. One of the possible future work would be changing the entire activation function of the network into ReLU from the training stage. In this way, modification would not be required, and it would sparsify the intermediate feature map at no cost. The second method would be adding the sparsifying loss term, as mentioned in the previous section, and it would create much sparser feature map. Also, there are some ideas in the direction of changing the network architecture to reduce the number of parameters. It would not directly impact the size of the feature map, but it would reduce the computation loads at the cost of performance.
3.3 Survey on semi-supervised learning

Over the past few years, there have been unprecedented advances in various areas such as image recognition and self-driving vehicles using deep learning networks. These networks are trained from two major learning paradigms; the one is supervised learning and the other is reinforcement learning. Both paradigms require specific goals designed by humans. In the case of supervised learning, these are the “labels” (assigned classification value of each image), and in the case of reinforcement learning, they are the “rewards” for successful behaviour (such as getting a high score in computer games). Having such designed goals has limitations in network learning. First of all, assigning labels to all training images is too expensive and the reliability of the assigned label is also difficult to measure accurately. Learning for the fixed goals allows the neural network to follow humans, however, it eliminates the potential of autonomous intelligence. Semi-supervised learning is proposed to alleviate above stated problems, which learns about the data they observe with only the certain amounts of labelled datasets without any other reward mechanisms.

The concept of semi-supervised learning is appropriate for DECENTER since large amounts of images without labels captured from edges can be utilized to improve the AI performance. To demonstrate the capability of semi-supervised learning on DECENTER, we surveyed several methods for image retrieval with semi-supervised learning schemes as follows. The first deep hashing method that can perform hash code learning and feature learning simultaneously in a semi-supervised fashion was proposed in ‘SSDH: Semi-supervised Deep Hashing for Large Scale Image Retrieval’ [24]. The semi-supervised deep hashing (SSDH) approach is designed to perform more effective hash function learning by simultaneously preserving semantic similarity and underlying data structures. The main contributions are as follows.

(1) Existing deep hashing methods design loss functions to capture semantic information of neighbours while maintaining similarity for effective hashing. In this procedure, the underlying data structures are ignored and as a result, the meaningful nearest neighbours are returned. To exploit the underlying structures of unlabelled data for more effective hashing, semi-supervised loss is proposed. It jointly minimizes the empirical error on labelled data as well as the embedding error on both labelled and unlabelled data. This strategy does not only preserve semantic similarity, but also can capture meaningful neighbours in the underlying data structures. This means that the SSDH method can benefit greatly from both labelled and unlabelled data in semi-supervised fashion.

(2) New deep network structure is designed to learn hash functions and image representations. It consists of three parts: a) representation learning layers extract discriminative deep features from the input images, b) hash code learning layer maps the image features into binary hash codes and c) classification layer predicts the pseudo labels of unlabelled data. In addition, semi-supervised loss function gives the semantic similarity constraints to the network, meanwhile preserves the underlying data structures of image features. The whole network is trained in an end-to-end manner to perform hashing and feature learning simultaneously.

(3) Traditional offline graph construction methods are time consuming due to $O(n^2)$ complexity and are difficult to handle for large scale data. Online graph construction strategy is proposed. Rather than constructing neighbourhood graph over all the data in advance, constructing the neighbourhood graph in a mini-batch during the training procedure is proposed. In the middle of online graph construction, it only needs to take into account the much smaller mini-batch of data, which is efficient and suitable for the batch-wise training.
Consequently, online graph construction can benefit from the evolving feature representations extracted from deep networks.

![Overview of the SSDH framework.](image)

**Figure 12** Overview of the SSDH framework.

![Image retrieval results of NUS-WIDE and CIFAR10 datasets using Hamming ranking on 48bit hash codes. The blue rectangles denote the query images and the red rectangles indicate wrong retrieval results. SSDH achieves the best results [25].](image)

**Figure 13** Image retrieval results of NUS-WIDE and CIFAR10 datasets using Hamming ranking on 48bit hash codes. The blue rectangles denote the query images and the red rectangles indicate wrong retrieval results. SSDH achieves the best results [25].

Following this approach, a new method was proposed in ‘Semi-Supervised Deep Hashing with a Bipartite Graph’ [26]. Another semi-supervised hashing method was proposed, named Deep Hashing with a Bipartite Graph (BGDH), which performs graph embedding, feature learning and hash code learning in a unified framework. The main contributions are as follows.

1. A bipartite graph is constructed to capture the information hidden in the labelled and unlabelled data. This graph could be an anchor graph that describes relationships between images and concepts, similarities between images and landmarks, or a traditional nearest neighbour graph.
(2) For each instance, to predict the neighbourhood context in the graph and feed both raw pixels and embeddings to a deep neural network, it needs to learn and concatenate the corresponding hidden layers while producing binary codes. BGDH can be a general learning framework in which any loss function of hashing and any type of graph can be incorporated. SSDH employs graphs to exploit the unlabelled data, while in contrast BGDH makes use of bipartite graphs, which can be constructed more efficiently since building an anchor graph only costs $O(n)$ time.

![Figure 14 Overview of the BGDH framework.](image)

The most recent research was proposed in 'Semi-Supervised Generative Adversarial Hashing for Image Retrieval' [27]. Constructing the graph model of large-scale data is extremely expensive in terms of time and space and using batch data instead may lead to a suboptimal result. The semi-supervised generative adversarial hashing (SSGAH) utilizes a generative model to model unlabelled data and uses triplet-wise labels as supervised information. The main contributions are as follows.

1. Semi-supervised generative adversarial hashing (SSGAH) approach is proposed to make full use of triplet-wise information and unlabelled data. It unifies generative, discriminative and deep hashing models in an adversarial framework, where the generative and discriminative models are carefully designed to capture the distribution of triplet-wise information in the semi-supervised fashion, all of which contribute to build semantic preserving binary codes.

2. Semi-supervised ranking loss and adversary ranking loss are proposed to learn better binary codes to capture semantic information of both labelled and unlabelled data. Semi-supervised ranking loss preserves relative similarity of real and synthetic samples. Adversary ranking loss makes the deep hashing model and generative model to improve each other in a two-player minimax game.
3.4 Remarks
In this chapter, activities related to delivering AI onto fog have been described. Techniques to deploy an AI onto devices with limited resources have been investigated, and among several of them, partitioning of AI has been chosen. This selection is done, because this approach is well-suited for the decentralized nature of DECENTER fog platform. To see the effect of model partitioning, two models are selected for object detection, which are closely related to the use cases of DECENTER project. The implementation of partitioning AI model includes a novel method to compress intermediate data to be transferred between partitioned AI models. That method showed that the intermediate data size can be reduced, while preserving accuracy in good levels. Also, techniques for using edge for training have been investigated. From those activities, we have identified issues related to model distribution on Fog Platform such as trade-off between intermediate data size and service accuracy, effect of model partitioning on AI service.
4 Containerization of AI methods

This chapter describes the WP4 activities of DECENTER, which took place during months M4 to M12 and are focused on putting AI method or application into a container. The container is suitable to be deployed as a microservice, so that any developer can build their own AI service from it. AI is a very complex entity which uses various technologies, including deep neural network model, audio/video processing and many other software modules to build a service frontend. The hierarchy of the AI service is already described in Chapter 2, and care should be taken of the level in the hierarchy where the containerization occurs to provide efficiency, flexibility and re-usability in the building of the AI service. DECENTER is going to provide methods to containerize AI into a microservice and investigate the effects on containerization.

This chapter is organized as follows: Section 4.1 describes the defined structure of AI container by identifying data flow within an AI service and an AI application. Section 4.2 describes the decomposition of the AI service into microservices and defines the building blocks in the AI service flow. Section 4.3 describes the details of the containerization of the AI application, and presents the implementation of two AI models on the following sections.

4.1 Design of the AI container: modules and interfaces

Docker provides an environment for developing and delivering of software in a package, called container. Recently docker has attracted attention from AI model development. Famous AI/ML platforms, such as TensorFlow and Caffe, provide their own development environment packaged in a Docker container. Instead of installing everything from scratch, developers can just pull the image which contains a complete development environment for AI/ML from the container repository, and start their own model building right away. However, those docker images are for the development purpose, and not ideal for service deployment. First, the container includes all the libraries and packages for the AI model development, thus making the image bigger and heavier. Development images, based on TensorFlow for video processing, can reach up to several GB storage, even without an AI model in them. As identified in [14], up to several hundred megabytes will be added with respect to the selected model. Second, those containers do not expose AI functionalities to other microservices. Service interfaces shall be implemented on that container from the scratch, and those interfaces may vary from one implementation to another. The main purpose of those containers is to provide a development environment, and not functionalities for deployment. Even if two AI applications use the same model, the interfaces for accessing the AI application can be different.

As the preliminary research on methods and solutions for AI in DECENTER, containerization of AI has been investigated with aspects described above:

- Level of containerization: On where the AI will be containerized
- What to be containerized: which and what software will be packaged in a container
- How to make container suitable for microservice

4.1.1 Level of containerization

Figure 1 depicts the hierarchy of the AI service which uses an AI model. After the training, a model (whether deep neural network or other deep learning methods) will be created, and there are many things to be added to run a model in the real environment. Consider use case #4 (UC4) of DECENTER for example. UC4 includes a scenario for detecting unregistered face from a video stream, and when it happens, the application server will send a notification to the user. As described in this use case, an AI Application Service consists of various
D4.1: First release of application’s Artificial Intelligence methods and solutions

microservices. It might need interaction with IoT platforms, storage service, service frontend and so on. In this use case, the role of each component which consists the AI service can be described as follows:

- **AI model**: A model file that is trained for face detection and face verification.
- **AI service**: A face detection service, which gets the image as an input and outputs true/false for the registered face.
- **IoT Platform**: software which provides IoT functionalities such as discovering IoT camera and controlling the video stream.
- **AI application**: This is where the application logic resides. In this example, application logic can be described as 'find IP camera and configure AI application to receive a video stream from it, and to send a notification when an unregistered face is detected'.
- **AI Application Service**: A service that consists of an application server, IoT platform and AI application.

DECENTER is going to leverage cloud technology for resource and services orchestration on DECENTER fog platform consisting of microservices. It means that the AI service or applications shall be able to be containerized and built as microservices. From the previous example of the AI service, it is obvious that the AI service can be built from microservices. The AI functionality can be containerized into a single microservice, and it can interact with other microservices such as IoT platform, database or web service frontend to provide end-to-end service.

However, there might be a few approaches for the containerization of AI service. The obvious method is to containerize an AI service as it is. This containerized AI service will receive input from input source, and the output of it will be the analysis result. For example, image classification AI service container will receive an image file as an input, and the output will be a text with the identified class name. This is what we call macroscopic containerization. AI service itself can be decomposed into several smaller microservices. In AI service, the input will be interpreted to an array, and applied to the AI model’s input layer. Output value emitted from the AI model is basically an array, and this value will be interpreted to more readable form. If those unit functions can be identified and containerized individually, more utilization of computing resources can be achieved. This approach is called microscopic containerization.

Usually, the computation of AI needs lot of computing resources since the model is composed of complex layers of neurons which involve numerical computations. Also, pre-processing and postprocessing of data might need many computing resources, especially for data-intensive applications such as audio and video applications. This microscopic containerization approach can help the utilization of resources, and also the scalability of an AI service. Moreover, since all pre-processing and post-processing are detached from models, reusability of those containers can be higher than of macroscopic containerization. Figure 16 and Figure 17 depict examples of microscopic and macroscopic containerization of AI service, respectively.

Microscopic containerization has advantage in resource utilization, however, since all the microservices will talk to each other through a network interface, it will increase requirements of network bandwidth between them. Those intermediate data can be larger compared to the original input size. The size of intermediate data (feature map) for an object detection model has been identified in section 3.2. Also, this inter-container communication will induce more delay from data source to service endpoint. DECENTER will investigate these issues in containerization of AI service, both in microscopic and macroscopic way. For the 1st year, macroscopic containerization has been designed and implemented.
For the first year of DECENTER project, the activities regarding AI containerization (M4 to M12) focused on the second approach. This approach containerizes AI application which consists of AI model, method, and directly related functions into a single container. Figure 17 depicts this configuration. Rationales for choosing second approaches are described here.

1. Model is closely related to the pre-processing and post-processing. Deep learning models are a complex set of numerical methods, and their output is usually a numeric value. Thus, a post-processing of those values to get a meaningful value is required. However, those models and post-processing are closely related. For example, VGG16 gets 244x244 colour image as an input, and outputs a score for each one of 1000 classes, while MobileNet model gets different sized image as an input. For now, detaching the model from its related functions does not have any benefits.
2. Microscopic containerization needs data transfer between each container which composes a single service, and it requires additional bandwidth for communication between containers and also storage on each container. Moreover, the size might be bigger than the original source. If a JPG image is to be applied to the input layer of AI model, it has to be transferred to an array, and this array is not compressed. Comparing to the original image, which is compressed in JPG, this inter-container data communication needs a lot more bandwidth. For the post-processing, the result of the classification is a string, but the communication between the AI Method and the Postprocessor needs 1000 scores for each class, thus increasing bandwidth and consuming more processing power.

4.1.2 Structure of AI container

The structure of DECENTER AI container, along with other AI containers, is depicted in Figure 18. The left part of the figure shows the structure of docker image as it is officially distributed by TensorFlow. Based on Ubuntu O/S, it consists of various libraries for computation, and TensorFlow for ML training. User can easily start developing an AI model with this container, since it eliminates all subtle issues for installing required libraries, version and dependencies between them. However, this container does not provide any methods or utilities to build external interfaces, which are essential for providing AI as a microservice. With the existing TensorFlow container, the developer needs to design these interfaces by him/herself. Orange-coloured boxes in the figure show implementations by a developer. To increase reusability of AI container and also to make it easy for refactoring, the AI functionalities and its interfaces need to be clearly defined. One of the purposes of DECENTER’s AI container is to provide those functionalities so that the developer can easily migrate its own application into a microservice. On the right part of Figure 18, the structure of DECENTER AI is depicted. Compared with the left side of the figure, the number of orange boxes, which need to be implemented by a developer, have been decreased, and green boxes are added. Those green boxes are going to be provided by DECENTER as a software package.

To provide a basis for AI container, which is suitable to package an AI application into microservice, DECENTER’s AI methods and solutions provide a base image for AI container. Basically, DECENTER AI base container consists of basic building blocks for AI application, including container images for operating systems, required libraries to provide application development environment and AI model. On top of that, DECENTER AI Base container has a DECENTER package in it. This DECENTER package is a software package to provide
intuitive methods to transform AI application into a microservice. Details of DECENTER package will be given in the next section.

DECENTER AI base container will leverage existing AI platforms and cloud technologies to package AI application into a container. However, there are many platforms which can be used for AI application development, along with hardware dependency such as GPU. And there is one more option - the embedded system. If the application does not require powerful computational resources at the edge device, an ARM platform can be used for an edge. Also, there might be needs of different libraries with respect to the purpose of AI application - for example, libraries for audio AI may be different from those for video AI.

DECENTER AI Base container has been designed to fulfill those restrictions for AI containerization. The configurations of Base containers are shown in Table 6.

<table>
<thead>
<tr>
<th></th>
<th>CPU (x86 or X64)</th>
<th>GPU (Nvidia)</th>
<th>CPU (ARMv7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Image</td>
<td>Ubuntu 16.04</td>
<td>Nvidia Ubuntu 16.04</td>
<td>Arm32v7/ubuntu</td>
</tr>
<tr>
<td>Parallel Processing Library</td>
<td>None</td>
<td>CUDA 9.0</td>
<td>None</td>
</tr>
<tr>
<td>Language (primary)</td>
<td>Python</td>
<td>Python</td>
<td>Python</td>
</tr>
<tr>
<td>AI Platform</td>
<td>TensorFlow</td>
<td>TensorFlow</td>
<td>TensorFlow</td>
</tr>
<tr>
<td>GPU support</td>
<td>No</td>
<td>Nvidia 1080ti</td>
<td>None</td>
</tr>
</tbody>
</table>

4.1.3 Package for microservice: decenter.ai

DECENTER base container provides a base image which is capable of running AI application on a container, and also provides libraries to help users to build AI as a microservice, meaning open interfaces to communicate with other microservices. The purpose of this DECENTER package is summarized in two objectives: 1) provide uniform interface to control AI containerized method, 2) provide base class for AI application implementation. Design of those interfaces and base class needs high level of abstraction of AI application flow.

4.1.3.1 Identifying common flow of AI application

The functionality to be opened to other microservices shall be identified before the design of the interface. From the viewpoint of AI service, raw data which need to be analyzed are the input of the AI service, and the output (analysis result). From the viewpoint of AI model, input and output are just arrays which are generated by parsing input data, and to be interpreted with some references, respectively. Without loss of generality, these AI application flow can be defined as follows:

1. Get raw data.
2. Preprocess raw data to make it suitable to be fed into the input layer of the neural network model.
3. Compute AI, and then retrieve values emitted from the output layer.
4. Interpret output values with reference.
5. Retrieve interpreted value.

From the viewpoint of client of an AI service, only 3 of the above flows are of interest. Setting data to be analyzed (step 1), control of AI computation (step 3), and retrieving analysis result (step 5). So, as a first step to design the interface, those three steps are considered in the design of the AI application as a microservice.
Existing AI deployment platforms such as Google’s TensorFlow Serving, encapsulate AI model (step 3 in above application flow) in it. The input for the TensorFlow Serving is an array which will be fed to the input layer of the neural network model, and the output is the value emitted from the output layer of the same neural network model. It means that data shall be preprocessed before sending them to TensorFlow Serving, and output of TensorFlow Serving shall be interpreted by the client. However, this encapsulation of model might degrade performances of real-time AI service since it needs the transmission of data between TensorFlow Serving and client repeatedly.

4.1.3.2 DECENTER package for AI container
This package (library) is designed to reflect the common flow of AI application which is identified in the previous section. The characteristics of DECENTER package include:

- **Uniform interfaces to configure/control AI method packaged in a container**: Set of interfaces are defined and implemented with Flask library. These interfaces support configuration of AI method such as input and output set up, and control of AI method itself. AI developers can add custom message handler onto Flask.
- **Base class for AI application implementation**: this class defines several methods for AI application implementation. AI developers can port its application by defining a new class inherited from this one, and then override several methods.
- **AppConfig class for AI application configuration values**: AppConfig is a placeholder to store and retrieve variables related to AI application.

With DECENTER package, AI application functionalities can be accessed from other microservices via these interfaces. Interfaces are designed with RESTful APIs at this stage, and extension to other protocols such as MQTT or gRPC are planned.
### Table 7 DECENTER RESTful Interfaces.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/setinput</td>
<td>Set input source to be fed to AI application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameter: source URL</td>
</tr>
<tr>
<td>GET</td>
<td>/getinput</td>
<td>Return current input</td>
</tr>
<tr>
<td>GET</td>
<td>/compute</td>
<td>Request computation of AI method with the neural network model. Output can be sent to the response message or delivered to the destination designated with <code>setdestination</code> method.</td>
</tr>
<tr>
<td>GET</td>
<td>/setdestination</td>
<td>Set destination to where output is to be delivered</td>
</tr>
</tbody>
</table>

Also, based on this Base class, AI developers can easily import their application into microservices. From the inherited class from Base class, three main methods (pre-processing AI computation and post-processing) shall be overridden. Message Handler in DECENTER package will provide a link between the RESTful interfaces and those methods. Developers can add more methods with respect to the model and add custom messages by extending the message handler.

- **Base class UML diagram and method explanation**

![Base class UML diagram](image)

*Figure 20 Classes of decenter.ai package.*

Figure 20 shows the UML diagram of DECENTER package. BaseClass provides a template for the AI application. Developers can build an AI application by inheriting and overriding the methods in the diagram. AppConfig class holds variables for AI applications and provides
setter/ getter methods for them. Class flask provides external HTTP (RESTful) interfaces for the AI application.

4.2 Building and Deploying AI services with Microservices

Docker containers, along with other cloud technology, can provide many benefits on resource and service management. DECENTER platform which is being investigated on WP3 is going to leverage technologies related to cloud and edge. Some consideration is needed in order for the AI Application Service (as microservice) to benefit from those technologies. In this Section, considerations on designing AI microservice to make most of cloud technologies are described.

4.2.1 Operational environment for containers within Kubernetes

Docker containers have a unique attribute - they run one, and only one, process at a time. In Kubernetes container orchestration platform, a Pod is the unit of container deployment and management. A pod aggregates one or more containers, which run closely related processes, which are collocated (scheduled) on the same node of a Kubernetes cluster (physical or virtual machine) and share the same resources of the node, such as network, memory and storage.

Each pod gets a unique IP address which is shared by all the containers (and all the replicas) belonging to it. That is not all - each container running within the same pod gets the same hostname, so that they can be addressed as a unit.

Pods scale vertically by requesting more resources to the containers they aggregate. Likewise, they scale horizontally by increasing the number of instances (or replicas) of the deployed unit. Replica sets are the number of pod instances running and represent the way to scale in/out deployments. Kubernetes Service is an object which acts as a single-entry point and load balancer for all the replicas of a Pod.

Within a pod, containerized processes:

i. **Communicate** through a shared file system (volumes) or via loopback network interface (127.0.0.1/8) which can help with critical performance use cases. On the downside, they need to export (be accessible through) different TCP/IP port numbers as they share the same Kubernetes cluster node (virtual machine, physical machine or cloud instance).

ii. **Vertically scale** all at once, as all the processes in a pod are deployed in the same Kubernetes cluster node (virtual machine, physical machine or cloud instance). In other words, if we need to provide them with more powerful machines, we need to provision a new machine and schedule all processes (containers) in that new machine. We cannot distribute/schedule the execution of some of the processes/containers in machines with low resources.

iii. **Horizontally scale** all at once, we increase/decrease the number of replicas (instances) of the pod; we cannot increase the number of instances of individual containers. This makes scaling a little bit inefficient as we need to duplicate all processes (the whole pod) at once even when just one of them needs it.

iv. **Expose a function** (service) to the outside world at an specific path of the Kubernetes Service hostname related to the Pod Deployment; the traffic to the http://host+path will be load-balanced (routed) toward one of the Pod replicas, physically deployed/scheduled in one of the nodes of the Kubernetes cluster and, specifically, to the TCP/IT port of the node that is exported by the process/container.
General good practices for deploying microservices on Kubernetes

Microservices architecture has a direct impact in the structure of the IT organization, preferring a structure of teams which account responsible for a single piece of functionality (a microservice) to a single team. A microservices architectural approach allows teams develop, deploy, update (evolve) and scale their services independently from the rest of the services and teams.

Therefore, when thinking about the best way to deploy and manage microservices, the decisions we need to make are mainly two:

A. Shall I package and run all the microservice code in a single container?
B. Shall I aggregate multipole containers (containing either whole microservices or parts of it) in a single pod?

Table 8 Evaluation of the benefits of the microservices approach with different correspondences between containers, pods and (micro-)service.

<table>
<thead>
<tr>
<th>Microservices Benefits</th>
<th>Single-container microservices</th>
<th>Multi-container microservices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multi-container pods</td>
<td>Single-container pods</td>
</tr>
<tr>
<td>Developed independently</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Deployed independently</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Updated independently</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scaled independently</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Cohesive life cycle</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 8 compares different options when answering those questions. This table shows that delivering more than one microservice in the same pod prevents us from two of the benefits of microservices architectures: deploy and scale each microservice independently. Moreover, it shows that the strategy of deploying different pieces (components) of a microservice in different pods may harm the cohesion of the administrative unit that the microservice must be (as the different pieces of the microservice have different life cycles).

We take the following conclusions from the previous evaluation results:

- To keep multiple containers, each of them comprising one microservice, in the same pod (unit of deployment and management in K8S) is discouraged.
- You may still want to encapsulate a microservice in a single container; however, no more microservice-enabling containers should be deployed in the same pod.
- Multi-container pods should only be used when they are highly coupled (processes) or you need a “helper process” to assist a “primary process,” needed for support of the main container such as a data loader.
  - Each container is considered as a schedulable task or process of a single microservice.
It is generally recommended to keep different microservices in different pods. In the cases where a microservice is implemented by a single container (schedulable task or process), there will be a correspondence 1-1 between container, pod and microservice.

As an example, let us take the typical LAMP application (e.g. dynamic web sites or web applications); while it is common to see a virtual machine (VM) running the full solution stack, when using containers and container orchestration platforms such as Kubernetes, the application must be split into at least two containers—one running the web server and the web application (e.g. Apache with PHP) and the other running the database (e.g. MySQL). If you include a cache into the stack (e.g. Memcached or Redis) to improve access to the database, it needs to be deployed and run on a separate container.

### Specific good practices for deploying microservices on DECENTER

From the general good practices outlined in the previous section, and according to the reference architecture for AI services which DECENTER plans to give support to, we have elaborated the following list of recommendations to deploy the microservices which

- Containerize the AI Method (most commonly the inference/prediction/scoring of a machine learning model, for example, a deep neural network) in one single container.
- Package the pre-processing and the post-processing tasks in one container each.
- Deploy the AI Method container together with the pre-process and the post-process containers as a Kubernetes Pod. Consider this unit a "microservice" (as it follows the microservices pattern) and call it AI Service.
- Expose all the ports of the containers aggregated in the AI Service pod through the same Kubernetes Service; this will act as a load balancer and single point of access for the replicas/instances of the pod when scaling horizontally. We can call this Kubernetes Service the entry point or gateway of our AI Service.
- An AI Application Service is made of multiple microservices which work together to deliver functionality to the end user, typically:
  - One or more AI Services
  - One or more App components (business logic, presentation, etc)
  - IoT platform services (e.g. data management)
  - Data stores service
  - Communication middleware service

---


4.2.2 Flexible AI service configuration with Kubernetes

There can be various configuration on using AI over cloud and edge. Zhou et al. [14] has proposed 6-level rating for intelligence, and defined four categories on edge-centric inference modes between device-edge-cloud continuum, which are: edge-based mode that inference is processed on edge, device-based mode that process inference on the device, edge-device mode that partial models are distributed over device and edge, and edge-cloud mode that partial models are distributed over edge and cloud. However, this approach does not quite match to DECENTER since in DECENTER, IoT device is a data source or data sink in IoT-edge-cloud continuum, and an edge is a device on where an application is deployed. Basic configuration models for cloud-edge collaboration in DECENTER are defined as follows. By combining those basic models, any combination of a distributed model can be deployed on nodes of fog platform.

4.2.2.1 AI distributed on an edge
This is the basic configuration of using the edge for inference. After the model is built on the cloud (or on some resource with rich computing power), the model and its application are going to be packaged into a single container and deployed onto edge resource. The deployed application uses data near edge resource for analysis and provides output to the service endpoint.

4.2.2.2 AI distributed on edges
This configuration focuses on the utilization of edge resources. After the model is built on the cloud, the model will be partitioned into several sub-models to make it suitable to be deployed on edge resources with smaller footprint. For example, the size of an AI model for VGG16 is about 550MB, while those of sub-models containing convolution layers and fully-connected layers are about 50MB and 450MB, respectively (file size is estimated with checkpoint file format in TensorFlow). It means that fewer computation resources are required for partitioned models, thus makes it able to utilize edge resources more efficiently. After the model is partitioned, independent containers for each sub-model and its application are going to be built and deployed onto edges.

However, this configuration requires additional communication between each container which contains the partitioned model. Particular care should be taken for choosing where to partition the model. If poorly chosen, the size of the intermediate data is going to be large, which might degrade performance of the whole AI service.

4.2.2.3 AI distributed between edge and cloud
Basically, this is the same as the previous configuration, except that sub-model can reside on cloud resources as well. On previous configuration, model partitioning for edge resources with smaller footprint is important since all the sub-models are going to be deployed onto edge resources. The purpose of this configuration is to help privacy preservation with AI on the edge, along with resource utilization. With this configuration, no raw data are going to be uploaded to the cloud resources. On the contrary, only intermediate data, which are results of being applied to partitioned model on edge resources, will be uploaded. This privacy-related aspect of using edge will help the privacy preservation on AI service, combined with other security measures of Fog Platform.

Again, since communication between partitioned AI models are required for this configuration, care shall be taken on where to split the AI model.
4.2.2.4 Practical Examples of AI distribution on Fog Platform.

This section describes practical examples of AI distribution on Fog Platform using proposed configurations of AI method and applications. The benefit of applying intelligence on the fog platform is not only placing intelligence near data source or service endpoint, but also very useful on applying recent AI technologies such as transfer learning or decision fusion.

- Partitioned AI on multiple edges

Consider an application which uses VGG16 model for image classification. VGG16 neural network consists of several layers which consist of convolutional layers which extract feature map from input image and fully-connected layers which computes scores for each class from that feature map. In this example, VGG16 is partitioned into two parts, one with convolutional layers and the other one with fully-connected layers. From the base container, containers with each model are built and deployed on edge resources with fog computing platform. The output of first sub-model is transmitted to the input of the second sub-model, and this configuration is set via the resource management part of the fog computing platform. Figure 21 depicts this configuration.

- Partitioned AI for re-trained network on edges

Transfer learning is a famous technique for re-training of the neural network. Instead of building and training a neural network from the scratch, transfer learning trains only the last part of a neural network which is trained for a similar purpose. For example, a neural network trained for human detection can be re-trained for car detection with transfer learning, and moreover, a common set of layers can be used for multiple re-trained neural networks. Figure 22 depicts this configuration. Based on the base container of DECENTER, three models are containerized: one with common layers with convolutional layers, and the other two with using re-trained parts of neural network. Instead of building two whole re-trained neural networks, this configuration enables efficient use of resources by providing a shared model for common layers of re-trained neural network.
Combination of AIs on edges - Ensemble or Decision Fusion.

It is a quite common approach to make a decision based on different input data. By combining different input types (i.e. text, video, audio), the accuracy of a decision can be improved. This kind of combining multiple AI methods are called ensemble or decision fusion. With fog platform and AI methods of DECENTER, this decision fusion can be implemented in a more intuitive way. Figure 23 depicts an example of AI service which uses two kinds of data (audio and video). Two AI applications, which are dedicated to audio and video analysis that are built, and another one for decision fusion which gets output results from those two AI applications as input is placed at the end of service configuration.
4.3 Implementation Activities

Implementation activities for containerization of AI have taken place along with the design phase, following the identification of the AI application flow. Among many platforms for AI model development and deployment, TensorFlow has been chosen for Y1. Environments for implementation and testing are as follows:

- Hardware: CPU (Intel) and GPU (Nvidia)
- AI Platform: TensorFlow
- Containerization: Docker container
- Language: Python (version 3)

Based on these S/W development environment, DECENTER AI package has been implemented and containerized in a docker container with AI platform. Then, two AI containers are implemented: VGG16 for image classification and Yolo v3 for object detection. Details on implementation are described in the following sections.

4.3.1 Implementation of DECENTER Package and base container

DECENTER package and base container have been implemented as described in section 4.1.3.2. For DECENTER package, a package named ‘decenter’ has been implemented, with ‘decenter.ai’ as a sub-package. ‘decenter.ai’ sub-packages contain a base class - ‘decenter.ai.BaseClass’ which is a template for the AI application implementation and an AppConfig class, named ‘decenter.ai.AppConfig’ which holds the variables for the AI application. A Flask server implementation is developed, named ‘decenter.ai.flask’ to provide RESTful interfaces for other microservices.

4.3.2 Implementation of VGG16 container

To validate the containerization of an AI method, an AI service which is able to classify an image is implemented. The neural network model used in this implementation is VGG16. VGG16 model gets 224x224 sized 3-channel (RGB) image as an input and generates scores for 1000 classes from the input image. A VGG16App class is defined from BaseClass, and several methods are overridden.

- preprocess_input(): reads source media, resize it to 224x224 image and save to an np array.
- compute_ai(): load neural network model onto memory and applies pre-processed input to that.
- postprocess_output(): read values from the output layer of VGG16 model, and interprets it to find top-1 and top-5 classes for the input image.

This container exposes three RESTful interfaces for configuration and control of the AI method inside the container, as defined in Table 7. The classification result will be sent inside the body of the response packet of the GET request sent to ‘http://container ip/compute’. For now, the class number with the highest score is returned.

Figure 24 depicts the UML diagram of VGG16 class and related classes. VGG16 application container is implemented on a Linux box with no hardware accelerator support. The Implementation is verified by accessing the RESTful APIs directly with a web browser.
4.3.3 Implementation of YOLO v3 container

After validating the containerization of the AI method with the previous implementation, we moved to implementing the scenarios defined in DECENTER use cases. There are four use cases in DECENTER, and several of them utilize object detection and classification from a live video stream. YOLO v3 has been chosen as a first implementation of the containerized object detection application. YOLO v3 gets images as an input and generates a bounding box coordinate for detected objects as well as its the corresponding class name.

Purpose of YOLO v3 implementation differs from the previous one over two aspects. First, it handles a continuous stream of data, so this implementation will verify how the stream can be processed in a container and what kind of resources are required. Since real-time video processing requires more computing resources, the implemented platform contains a hardware accelerator. A PC with Nvidia 1080ti and an embedded system with Nvidia Tegra GPU (Jetson TX2) has been chosen for implementation, and corresponding containers are built for each one. Second, it needs to generate a continuous output of analysis result in real-time and overlaying it with the original image. To this end, a very simple MJPEG streamer has been added at the end of the post-processor and flask of DECENTER package. Figure 25 depicts them UML diagram of YOLO v3 container implementation. Comparing to the previous one, there are two newly added methods in YOLOv3App class. One is compute_ai_stream() methods, which generates continuous stream of outputs, and compute_stream() interface for RESTful API which accesses compute_ai_stream() method.

Environments for YOLOv3 implementations consist of: an IP Camera which is able to generate H.264 RTSP stream, a GPU box with YOLO v3 container running on it, and a client device with a web browser. YOLO v3 container on GPU box will retrieve H.264 RTSP stream from IP camera and analyse it. After analysis, the YOLO v3 container will draw a bounding box with a name for each detected object from the input stream. This output image will be provided with MJPEG stream to the client device’s web browser.
Figure 25 YOLO v3 application on DECENTER container.

Figure 26 shows the Implementation result. It depicts results from GPU container for Yolov3App. On the left side, the original streaming video from the IP Camera is shown, and on the right the resulting video stream of Yolov3App is presented. Three containers are built for different hardware platform, PC (Intel i5), PC with Nvidia 1080ti GPU and Jetson TX2 which is an embedded platform with Nvidia Tegra GPU. Test result on each platform is given in Table 9.

Table 9 Containerized YOLOv3App performance.

<table>
<thead>
<tr>
<th>Container</th>
<th>Intel i5</th>
<th>Nvidia 1080ti</th>
<th>Nvidia Jetson TX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing time for a frame</td>
<td>Up to 1.2 sec</td>
<td>Up to 48ms</td>
<td>Up to 4.3 sec</td>
</tr>
</tbody>
</table>

Figure 26 Result of YOLOv3App. Left: original stream received from IP Camera, Right: Object Detection Result streamed from an YOLO v3 Microservice.
4.4 Remarks

This chapter described the DECENTER activities regarding the AI containerization during M4 and M12. With those activities, data flow and issues on AI containerization were identified. Based on the data flow, a basic structure of the AI container has been defined - what will be containerized in an AI container, on which level the AI will be containerized, and how it will be containerized. To provide intuitive methods to merge an AI application into a container, a DECENTER package has been designed and implemented. This DECENTER package provides uniform interfaces to access AI methods and a base class for the AI application implementation. Together with AI containerized AI platform, this DECENTER package composes a base container for AI application. Compared to the other AI platform containers, which are focused on providing development environment, DECENTER base container for AI application is more suitable for service deployment since it has uniform interfaces to communicate with other interfaces regardless of the AI model it is based on. Three base containers, for PC, GPU and embedded GPU, have been implemented with DECENTER package, and two AI applications have been containerized with them. Implementation of YOLO v3 container on a GPU device shows a reasonable performance, with real-time streaming output generation encoded in MJPEG format over HTTP.

Based on this result, the following work items will be investigated in the forthcoming year.

- Containerization of partitioned models.
- Partitioned model deployment on Fog platform with proper configuration.
- Monitoring of AI resources.
- Building AI service with microservices.
5 Digital Twin representation

5.1 Digital Twin as a Technological Asset

The concept of Digital Twin is not new. While it is commonly considered to be developed in 2002, digital twin technology itself has actually been a concept practiced since the 1960s. The first inventor of this concept is NASA which created physical duplicated systems at ground level to match, simulate and assess the systems in space. Nowadays, these systems are no longer physical, but mainly virtual and are used to perform fully digital simulations to mirror and diagnose problems in orbit.

Even though the term of digital twin was initially related with complex entity representation and heavy, time-critical simulations, the term took off and widely used after Gartner included the digital twin in the top-10 strategic technology trends for 2017. The proliferation of Internet of Things (IoT) devices enabled the digital twin to become cost-effective and easy-to-use. In Industry 4.0 the concept of Digital Twin is already part of the strategy of companies and product design, while G2 Digital Trends\(^23\) have included digital twins as a key factor on business modernization. Nowadays, several digital-twin business applications can be found in a number of sectors, such as manufacturing, automotive, healthcare. These applications can represent complicated (e.g., autonomous vehicles) and simple (e.g., light lamp) entities and the amount of data used to build and update these entities determines how precise the simulations are.

In DECENTER, the Digital Twin is not used in the most common and well-known context of manufacturing; for monitoring, running simulations and making decisions at each stage of a product’s manufacturing process. Given the diverse nature of supported use-cases therefore, within the project the wording is rather used to refer to a virtual representation realised at software level of a real situation that is being observed through sensors and influenced through actuators. In this respect we are looking at a twin image at software level of a real scenario, hence the Digital Twin wording, used in DECENTER after it being stripped of its recent “industry-related” connotations.

This peculiar Digital Twin concept is used in the context of smart environments, such as safe street crossing, safety at home, etc. Even though the context is different, there are several common objectives and needs when compared to the manufacturing domain. For instance, the need for data aggregation, data analysis and inference, maintenance and control methods is paramount. Experimenting and validating algorithms on a digital representation of the real world clearly makes things easier to manage from a research and assessment point of view.

In this chapter, we elaborate on how the concept of Digital Twin is being used in DECENTER and how it will exploit the underlying orchestrated DECENTER fog platform.

**Digital Twin Definition**

Several definitions have been proposed for Digital Twin over the years\(^24\). Although each one of them emphasizes on different aspects (technological, business, etc) of the Digital Twin, we can conclude that all of them exhibit a common denominator; they formalize what a Digital Twin is and they analyse what its impact is. Based on these two factors, we provide our short but complete definition.

\(^23\) [https://learn.g2.com/trends/digital-twins](https://learn.g2.com/trends/digital-twins)

Digital twin is a digital replica of an existing physical or non-existing virtual entity. It can represent a simple physical object such as a camera, but also a complicated system such as real-time face detection. The replica can be created using not only real-time sensor data, but also learning data derived by Artificial Intelligence (AI) units. The main purpose of a digital representation is to create a real mapping of the (non-) existing entity, which stays always in line and up-to-date across the entire life cycle of that entity. This goal has a great system-level impact; it allows the monitoring, management, simulation and assessing of any software solution or system from the early stage of development until the final stage of deployment.

Digital Twin in DECENTER

In this section, we justify and explain the reasons of adopting the Digital Twin technology in the context of DECENTER. Moreover, we present the Use Case requirements that guided the decision of developing the Digital Twin.

• Monitoring. Monitoring a real-life situation is a key requirement of all Use Cases. The Digital Twin, in collaboration with the IoT platform, will allow a quick and easy access to all connected “things”-devices, regardless of their location, hardware specifications or communication protocols. Moreover, the Digital Twin, in collaboration with the AI models, will allow the monitoring of other “non-things”, physical (e.g., human) entities. Last but not least, the Digital Twin will augment the information existing in the physical world with the results coming from the feature extraction process (e.g., calculated distances between objects), the inference rules (e.g., detected alerts) and other virtual entities (e.g., entities for AI configuration, external APIs). This concept is further discussed in Section 5.2.

• Inference. Aggregating information received from multiple sources, such as IoT devices and AI models, is particularly important for inferring interesting and critical conclusions. The Digital Twin will provide a unified data model (see Section 6.2) in order to enable a uniform way of accessing and retrieving data from several diverse sources. Moreover, the Digital Twin will continuously update its data model with the most recent values of these sources.

• Decision making. A backend logic, seen in some real-life scenarios, but also in Use Case 3, requires the ability of an application to make decisions on which AI or IoT device to use in order to favour the prediction task. For instance, to ensure the safety of workers in a construction site, it is required to make decisions at real time of which AI model (e.g., helmet or vest detection) and which camera (e.g., based on proximity) to use. Digital Twin will facilitate the decision making, by providing means for testing and validating different options for smart orchestration of AI and IoT resources before deployment. We will further elaborate on this idea in D4.2.

• Maintenance. A significant requirement of all the different Use Cases is that the system remains robust and thus able to perform even in extreme conditions (e.g., bad weather). The Digital Twin provides a near real-time information awareness of the status of the physical and the AI entities. This information will allow the Use Cases to prevent problems, such as long downtimes of the IoT devices, internet disconnections, hardware failures, misses or anomalies of IoT and inference data. Moreover, the Digital Twin (in collaboration with the IoT platform) will allow actuations over the physical devices and tuning of the AI models, in order to prevent or remotely solve potential maintenance problems.
• Data Collection. A requirement met in some Use Cases is the ability to store timestamped data in order to reproduce situations and assess the efficiency of different algorithms. The Digital Twin (in collaboration with the IoT platform) will be able to store, retrieve and represent, not only real-time, but also historical versions of an entity. Section 6.3 presents some technical details for storing historical data and an appropriate API for accessing them.

As opposed to closed and proprietary monolithic models for Digital Twin used in various industrial context, within DECENTER we will be using a more open and modular approach where our Digital Twin is made of components working together but which can be individually replaced if more powerful algorithms fitting better the monitored data can be found, if additional sensor data is found to improve the virtual representation of the real situation at hand etc.

5.2 Using multiple data sources for effective digital twin representation

Digital twins are simulated models of real situations that are of a certain interest or worth monitoring in order to infer how these situations might evolve and to assess how one can intervene in advance to prevent unwanted events from happening. Related activities focus on how it is possible to digitally extract meaningful features and reproduce a situation model where information coming from different data sources can be more easily combined and actuated on. While in the beginning of this document we focused on a top-down approach, looking at how monolithic models can be split, containerised and deployed over a distributed infrastructure, here we illustrate a more bottom-up approach, where different data sources individually feed various models that produce annotated and synchronised events and or inferences of some sort which are then collectively processed to produce intended and robust (i.e. high accuracy) interpretation outcomes and/or actuation inputs.

Advances in the IoT domain in terms of miniaturisation of sensors and ability to process and transmit data have certainly increased the monitoring fabric one can rely on in order to recreate accurate models which, fed by those data, can then be used to simulate under controlled conditions how the monitored system might evolve.

Figuratively speaking, to realise a digital twin, it needs data sources and a model that can process in order to produce a continuous outcome or some punctual notifications. Similar to what has been introduced in the first chapter, sending continuous data streams to a backend cloud model for further interpretation can indeed be highly inefficient for many application scenarios. The ability to process data rapidly is one of the key advantages that edge computing is bringing and DECENTER is therefore well-positioned as a project to produce relevant research outcomes also in this domain.

Digital twin representations have to model how real situations might evolve and how a system might react based on a combination of inputs. In particular, we aim to extract concepts, events and situations and reproduce digitally the represented real conditions. The more sources of data we have to represent reality, the better it is for the model designers to recreate a system that accurately reflects reality.

In fact this is where digital twins meet machine learning: our approach in some of DECENTER activities in this context is to use machine learning algorithms that interpret the data collected by the sensing units and underpin the models we use, paying particular attention to how the proposed distributed computing infrastructure can indeed improve the reactivity and the performance of the envisaged systems.

In particular, even though this approach has been designed around very specific use-cases requirements, it is clearly re-applicable in other contexts as well, simply changing the targets
for what in the real-life situation we decide will be our extrapolated concepts, events and situations.

In DECENTER we designed a system capable of reproducing the digital twin image of a pedestrian crossing with inherent objects (people, bicycles, cars etc.) and interfaced it with actuators deployed in a real environment, to generate the needed notifications to pedestrian, cyclists and drivers about to be involved in a safety high-risk situation.

From an AI and edge cloud perspective, the algorithms supporting such a system have been designed and trained offline and will be deployed in a real environment collecting metrics related to confidence levels achieved in object recognition. We expect the system to undergo periodic retraining in the backend and redeployment at the edge based on observed performance.

Going more into details, the designed system consists of two independent subsystems, one running in the background which we will refer to as Crossing Alert Level Assessment (CALA) and dedicated to a continuous assessment of the danger levels of the crossing (Crossing STATUS) and one, referred to as Imminent Danger Notification (IDN), dedicated to raising the audio-visual alerts in case of dangerous events at a particular moment in time (Crossing ACTIONS).

Both of these subsystems will be made of individual components which, once containerised, can be easily orchestrated in terms of where they will be executed, based on observed conditions and target performance.

In order to allow reinforcement learning (i.e. algorithms retraining), the overall framework will also keep a buffer of processed data in case of low-confidence levels which will then be further processed in the backend in an offline mode to allow retraining and continuous improvements of the models created which will then be regularly redeployed to keep the system continuously improving.

5.3 The concepts – events – situations framework for digital twin representation

In this section, we will present the framework to implement a digital twin which is used to analyze different situations occurring at a pedestrian crossing. The physical world is semantically represented in a digital twin using three main components: 1) concepts; 2) events; 3) situations and outcomes. A concept defines any physical entity existing in the physical world which is measured by the sensing infrastructure using various sensors such as camera, microphone, temperature and luminosity. In order to simplify the representation, we represent the concepts in the form of a set named according to the type of concept and its associated sensing system. Each concept has associated attributes, for example, a car detected by a camera has attributes such as its colour, time stamps of its motion, its speed, and its physical geographical location. Events are based on individual concepts or defined by relationships between multiple concepts, for example, a car is moving fast or a car is approaching the crossing are two different events. Situations are based on detected concepts and linked events. Situations are formally analyzed using a knowledge-base which contains rules representing a situation based on a concept, event, and contextual knowledge (i.e. rules which fire outputs upon determined inputs or combinations of events).
Sensors, cameras and microphones produce raw data; each feed is then separately processed and concepts (i.e. a car, a truck, a motorbike) are identified. Further interference leads to events (i.e. a car approaching fast, road conditions wet, car skidding, pedestrian about to cross) and the joint interpretation of events with a Knowledge Base made of basic rules triggers either a change of STATUS or an immediate ACTION such as a loud notification of danger.

The multiple inputs feeds from various IoT devices allows the system to detect anomalies with more reliability. Moreover, some anomalies are associated with one type of input feed. For example, the camera feed can be used to more reliably detect a moving car and a person walking on a crossing whereas audio feed increases reliability in the detection by the intelligent analysis of the sound feed. A detailed description of individual IoT devices along with their function is given as follows:

- **Video feed**: one or multiple cameras are used to capture real-time video of objects moving on or around a crossing. The objects of interest mainly include cars, trucks, cycles, and pedestrians. In this feed, each object is represented as a separate concept in our system. For example, a car is a concept and the video feed is processed to detect this concept in each frame.
- **Sound feed**: just like the video feed, the sound feed also involves one or an array of microphones. An array of microphones is in fact more useful as it can capture sound coming from multiple directions. Each sound generated by individual objects is
considered as a concept in the digital twin. For example, a car can produce multiple sound-based concepts such as car passing, tyre skid, horn, etc. Each of these different sounds are recognized using the intelligent processing of audio data.

- Sensor feed: it is mainly based on three types of sensors namely humidity, temperature, and luminosity. The purpose of using these sensors is to add contextual information to the system in the form of environmental and weather conditions of the surroundings. The information about weather conditions is typically provided by humidity and temperature sensors which include information such as dry, rainy, or foggy weather conditions.

Inference of Concepts, Events and Situations – a platform-oriented approach

When adapted to exploit the features of the underlying DECENTER platform, the proposed framework enables the hosting of different methods and algorithms which can then be easily replaced when better performant ones are found or become suitable to be run on the available infrastructure. The picture below illustrates an example tailored to the pedestrian crossing use-case. Extracting a concept (i.e. detecting an object such as a car, a person or a bicycle) can be done through combinations of different feature extraction methods, machine learning algorithms and deployed trained models.

![Figure 29 extraction of concepts from raw feeds and containerised components.](image1)

The figure below illustrates how events are then derived from concepts.

![Figure 30 extraction of events.](image2)

Once such a Digital Twin system becomes able to recognise concepts and events, which happen to be independent from the specific topology of the pedestrian crossing they are extracted from, on-site contextual information and location specific rules become very important in order to analyze a situation occurring at a crossing.
Based on the contextual information, as already mentioned above, the objective becomes to continuously assess the STATUS (green, yellow or red) of the crossing AND to trigger as necessary ACTIONS that warn of imminent danger (i.e. audio-visual notifications to pedestrians and/or drivers).

5.4 Applicability of Digital Twin concept in various DECENTER Use Cases

Digital Twin concept has gained further visibility with the rise of the Internet of Things (IoT). Real-world objects, such as cameras and environmental sensors are linked to their digital identical, providing real-time information about their status. Beyond the IoT context, a great potential is being studied in order for entities, other than “things”, to be associated with Digital Twin. Entities derived from all aspects of our world, but also from AI-based capabilities, can be dynamically connected and digitally represented. This representation allows the creation of advanced simulations, operations and analysis.

In this section, we study the digital representation for both IoT and AI entities in the context of four different use cases.

Data, Features and AI Methods for Digital Twin

Table 10 presents all interesting entities, which will be represented by Digital Twin, regarding the four different use cases. For each digital entity, the associated raw data are described along with the AI method used to support the life cycle of that entity. It is worth noting that the digital twin is highly concerned with privacy issues. To ensure privacy, different anonymization methods are adopted for the entities associated with sensitive data.

<table>
<thead>
<tr>
<th>Use Case (UC)</th>
<th>UC Scenario</th>
<th>Digital Twin Entity</th>
<th>Raw Data</th>
<th>AI Model</th>
<th>Data Anonymization</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC1 - Smart City Crossing Safety</td>
<td>SC1.1: objects detection, video interpretation</td>
<td>-Pedestrian -Pedestrians with buggies -Bicycles -Cars -Buses -Trucks</td>
<td>Video Stream</td>
<td>Object Detection</td>
<td>Vehicle plates if visible from video recording</td>
</tr>
<tr>
<td></td>
<td>SC1.2: events detection, sound interpretation</td>
<td>-Events based on sound</td>
<td>Sound recording</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SC1.3: events detection, IoT sensors data interpretation</td>
<td>-Road -Crossing -Weather conditions</td>
<td>IoT data streams</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SC1.4:</td>
<td>generation of alarms</td>
<td>-Generated alarm</td>
<td>Continuous interpretation of outputs from previous scenarios</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC1.5:</td>
<td>pedestrian crossing status assessment - hazardous conditions detection</td>
<td>-Alert Level (Red, Yellow, Green)</td>
<td>Interpretation outputs (counting of objects, events and feedback from IoT)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**UC2 - Robotic Logistics**

| SC2.1: Person Detection at the specific zone | Location of detected person | Image from a connected camera and measures from laser scan | Convolutional Neural Network (CNN) | Person’s shape |
| SC2.2: Robot Detection at the specific zone | Location of detected robot | Image from a connected camera and measures from laser scan | CNN | Robot’s shape |

**UC3 - Smart & Safe Construction**

| SC3.1: Vehicle type recognition | Original image and features: vehicle type, colour and registration plate, alarm notification | Image from a connected camera | CNN | Person’s face |
| SC3.2: Person detection & identification, Safety helmet and vest detection | Original image and features: person position and name, safety equipment list check, alarm notification | Image from a connected camera | CNN | Person’s face for construction site visitors |
| SC3.3: Hazardous work conditions detection | Temperature, humidity, wind, CO2 concentration warning | - Sensor data streams (temperature, humidity, wind flow, air pressure, CO2 concentration) - audio stream (for detection of noise level) | CNN |  |
| SC3.4: Stock control | Original image and features: stock control list | Image from a connected camera |  |  |

**UC4 - Ambience Intelligence for safety at home**

<p>| SC4.1: Person Detection at the specific zone | Location of detected person | Image from a connected camera | CNN | Person’s shape |</p>
<table>
<thead>
<tr>
<th>SC4.2: Member Verification</th>
<th>Registered member</th>
<th>Image from a connected camera</th>
<th>CNN</th>
<th>Person’s face</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC4.3: Indoor environment prediction</td>
<td>PM10, PM2.5 status in the future</td>
<td>Pandas Dataframe type dataset (including Indoor CO2/PM10/PM25/temperature/humidity/noise/VoCs information, outdoor PM10/PM25/temperature/humidity information)</td>
<td>LSTM+</td>
<td>LSTMP+</td>
</tr>
</tbody>
</table>
6 AI models and Digital Twin: Reciprocal Exchange of Data

It is evidence that the success of an AI Model is highly dependent on the data feeds. Further, the operation of a Digital Twin is only possible by feeding it with real-time data. In this section, we explain how the AI Model and the Digital Twin can co-operate, in order to achieve a reciprocal exchange of data.

Figure 32 illustrates the flow of information among the AI Model, the Digital Twin and the IoT platform. We notice that the Digital Twin is continuously receiving data coming from the inference results of the AI Model. Similarly, the AI Model is fed by the Digital Twin with data derived by devices connected to the IoT platform.

Given the necessity of Digital Twin to interact with the AI Model, the IoT platform but also with the end-user application (as we will see later), it becomes useful to equip DECENTER with a standard set of APIs for interfacing as many IoT entities as possible.

The main purpose of this chapter is to describe the role of the IoT platform, to propose a data model for the Digital Twin and to provide a technical guide of the Digital Twin management tool. It is worth mentioning that both, data model and management tool, are based and supported by eclipse sensiNact IoT platform27.

6.1 The role of IoT platform

Today, IoT platform is seen as an enabler to connect to smart physical devices or other non-physical APIs. However, the vision of tomorrow forces the IoT platform to enable the access and interaction, not only to the data, but also to the knowledge derived by them. Further, the most well-known IoT platforms, such as Microsoft Azure, Watson IoT, etc are operating on the cloud mainly for ensuring scalability. However, the vision of tomorrow foresees the need for an IoT platform, which operates at the edge closer to the smart devices.

In the context of DECENTER, we are interested in exploring the vision and rationale of incorporating Digital Twin technology into the IoT platform, running at the edge. Achieving such an integration will result in advancements for both; Digital Twin technology and IoT platform.

Digital Twin technology will be benefited by:

27 https://projects.eclipse.org/projects/technology.sensinact
- Bringing the data needed to understand how the physical twin behaves and operates in real world
- Ameliorating its performance, by operating at the edge where data can be accessed and analyzed fast, when needed

IoT platform will be benefited by:

- Providing, not only connectivity and data feeds of IoT devices, but access to knowledge-driven APIs
- Enhancing AI-based optimization related to the operational processes and the physical system
- Distributing insights of the product’s (or service’s) lifecycle and facilitating their testing and maintainability
- Contributing to the creation of business options, such as selling a service capability rather than a product

6.1.1 sensiNact IoT Platform

SensiNact middleware is a unified framework to integrate and manage IoT devices, collect their data and enable application development. SensiNact allows you to:

- Support state-of-the-art IoT protocols (ZigBee, CoAP, EnOcean, LoRa, SIGFOX, MQTT, XMPP, etc.)
- Access on-demand, periodically or event-based real-time data for online analysis
- Access historical data for offline analysis
- Use any protocol to remotely access unified data and action sources (HTTP REST APIs, WebSockets, MQTT, XMPP, etc.)
- Rapidly create new bridges to emerging protocols and dynamically integrate them to the running platform

The data is organized with a simple, although powerful data model, which is shown in Figure 33. It is organized around three main concepts: ServiceProvider, Service and Resource.

![Figure 33 sensiNact Data Model](image-url)
A ServiceProvider represents a physical or a virtual entity. For example, a physical device can be an environmental sensor, while a virtual device can be a weather API. A service is a structural organization of the information that is provided by a given ServiceProvider. As an example, the ServiceProvider of an environmental sensor provides as a service a real-time monitoring of the room’s ambience. The Resource can be a state (luminosity) or action variable (turn on/off lights) of a service.

Resources and services can be exposed for remote discovery and access using different communication protocols, such as HTTP REST, JSON-RPC, etc. A detailed classification of the various Resource types is shown in Table 11, while the access methods are described in Table 12.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SENSORDATA</td>
<td>Sensory data provided by a service. This is real-time information provided, for example, by the SmartObject that measures physical quantities.</td>
</tr>
<tr>
<td>ACTION</td>
<td>Functionality provided by a service. This is mostly an actuation on the physical environment via an actuator SmartObject supporting this functionality (turn on light, open door, etc.) but can also be a request to do a virtual action (play a multimedia on a TV, make a parking space reservation, etc.)</td>
</tr>
<tr>
<td>STATEVARIABLE</td>
<td>Information representing a SmartObject state variable of the service. This variable is most likely to be modified by an action (turn on light modifies the light state, opening door changes the door state, etc.) but also to intrinsic conditions associated to the working procedure of the service</td>
</tr>
<tr>
<td>PROPERTY</td>
<td>Property exposed by a service. This is information which is likely to be static (owner, model, vendor, static location, etc.). In some cases, this property can be allowed to be modified.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Get the value attribute of the resource</td>
</tr>
<tr>
<td>SET</td>
<td>Sets a given new value as the data value of the resource</td>
</tr>
<tr>
<td>ACT</td>
<td>Invokes the resource (method execution) with a set of defined parameters</td>
</tr>
<tr>
<td>SUBSCRIBE</td>
<td>Subscribes to the resource with optional condition and periodicity</td>
</tr>
<tr>
<td>UNSUBSCRIBE</td>
<td>Remove an existing subscription</td>
</tr>
</tbody>
</table>

Each access method is associated to a particular resource type. For instance, a GET method is associated to resources of type Property, StateVariable and SensorData. A SET method can only be associated to StateVariable and modifiable Property resources. An ACT method can only be associated to an Action resources. SUBSCRIBE and UNSUBSCRIBE methods can be associated to any resources.

6.2 Digital Twin Data Model

Figure 33 depicts the data model of the Digital Twin, which is designed to serve the needs of the use cases described in Section 5.4. Based on the analysis of the UCs, a first design of a common data model is proposed. The proposed model respects and is also built upon the
principles of sensiNact data model. Two basic ServiceProvider are identified; camera and sensor.

The camera is modelled as a provider of the following services:

- **Admin** is a service related to a physical IoT device. It provides information about the unique *identifier* and *location* of a particular camera.
- **ObjectDetection** is a service associated to the AI method. It provides information about the content of a frame captured by the camera. Particularly, it describes all *objects being detected*, within the frame, by the AI method and the *probabilities of being* these objects. Further, it triggers an event when the *number of detected objects* has changed or when a *specific object* is detected.
- **Images** is a service which provides access to the URI of the live stream and to the last image captured by the camera.

The sensor is modelled as a provider of the following services:

- **Admin** is a service related to a physical IoT device. It provides information about the unique *identifier* and *location* of a particular environmental sensor.
- **Monitor** is a service dedicated to real-time monitoring of sensor’s state. It provides the most recent value of any environmental sensor.
- **Control** is a service dedicated to controlling the *status* of a sensor, if permitted. This service performs an action over the sensor, allowing the modification of its status (e.g., turn on/off).

Figure 34 illustrates the aforementioned data model. It is worth noticing that all entities in light green colour are directly associated with data coming from a physical device (such as camera or sensor). Their values are updated based on the status of the physical device and these values are used to feed the AI models with data. On the contrary, entities in dark green are associated and being fed by the inference results of the AI models. In the next section, we focus on explaining the reciprocal exchange of data between AI models and Digital Twin entities.
6.2.1 Feeding AI Model and Digital Twin with Data

Figure 35 illustrates an example of the data flow among the IoT platform, the AI model, the Digital Twin and the end-user application. The presented flow of information is of great interest to those UCs, which are interested in feeding their AI Model with data from a camera and then interact with the Digital Twin to retrieve all upcoming events.

We notice that the Digital Twin serves as an interface between the end-user application and the underlying AI Model and IoT platform. As a first step the Digital Twin subscribes to the data feeds of all available IoT devices (steps 1-2). Then, any end-user application can request for the list of these IoT devices (steps 3-4) and use them in order to configure the AI Model (steps 6-8). Finally, any third-party application can request for being notified of the upcoming detection events.
In the next section we discuss the Digital Twin Service Management API, which is a REST API dedicated to developers. The proposed API suggests the usage of HTTP requests in order to allow the interaction between the Digital Twin and the user application. For instance, step 3 is a simple HTTP GET request (http://url:port/sensinact/providers), while task 9 is based on an HTTP POST request (http://url:port/sensinact/providers/camera/services/objectdetection/resources/DETECTED_OBJECTS/SUBSCRIBE).

6.3 Digital Twin Service Management API

Figure 36 illustrates a swagger web application, describing all useful interactions with the Digital Twin service. The swagger is provided as a sensiNact REST API.

The RESTful API provides the ability to perform HTTP requests in order to GET and POST data. A GET request can be used to retrieve the information related with a ServiceProvider, Service and Resource. The GET method is considered idempotent as it does not modify the state of the resource. Thus, multiply requests produce the same result if no other method (e.g., POST) has changed the value of the resource. The POST API can be used for changing (SET) and interacting (ACT) with the resource representation. The POST method is not idempotent and results into modifying the value of a resource, if it is permitted. The POST method gives also the possibility to a third-party application to be asynchronously notified for any change of a resource’s value, simply by performing a SUBSCRIBE request.
Figure 37 depicts an example of how to perform a GET HTTP request, in order to obtain the list of detected objects within the last frame captured by the connected camera. The obtained result is formatted in JSON and contains information about the type, value and timestamp of the requested resource.

The interaction with the Digital Twin enables the end-user and any third-party applications to retrieve and update their information in an almost real-time context. Further, the Digital Twin can be used to test, validate and assess the underlying AI system for object detection.

The source code of Digital Twin can be found in the public repository of eclipse sensiNact: https://git.eclipse.org/c/sensinact/org.eclipse.sensinact.gateway.git
It is worth noticing that the Digital Twin is an evolving digital profile of physical objects and virtual processes. Storing the evolution of this profile is particularly significant for the optimization of the use cases, as well as for prediction tasks (e.g., predict future environmental values). For this purpose, sensINact allows an automatic storage and retrieval of the digital twin profiles. For the purpose of storage, a CASSANDRA database is used. For the purpose of retrieving historical data, a REST API data is provided.
7 Conclusions

This report describes preliminary investigations on AI on the edge, and design and implementations on how AI service can be deployed onto Fog Platform of DECENTER which are performed in tasks T4.1, T4.2 and T4.4 during the first year of the project (Activities on T4.3 will be described separately in D4.2 at the month of M18).

To make AI more compatible with fog computing platform, the requirements for AI and cloud are derived from the investigation of existing AI platforms and issues on AI on the edge. Those preliminary investigations are presented in Chapter 2. Among many approaches to bring AI onto edge, partitioning method for the AI model in multiple sub-models has been selected for the first year, since it mostly fits to DECENTER platform which engages fog platform. The effects of model partitioning for the AI service have been investigated and their results are given in Chapter 3. DECENTER project is going to leverage cloud technologies including docker and Kubernetes for resource management, so AI Application Services on DECENTER platform shall be compliant to those technologies. The activities to make AI application and service more compatible to the cloud technologies are described in Chapter 4, including identification of the AI application flow, containerization of the AI and interaction between AI and other microservices. The utility of Digital Twin in representing AI entities at the edge is discussed in Chapter 5 and 6.

With those activities in the first year, we were able to provide a first version of application’s Artificial Intelligence methods and solutions of DECENTER. AI Application Services can be composed with various microservices, including AI microservice based on DECENTER AI Base Container. The AI developer can take benefits of DECENTER’s AI methods and solutions to (i) turn their AI application into microservice with uniform interfaces, (ii) build an application service from microservices and deploy them on the Fog Platform, and (iii) build an additional service such as Digital Twin at the edge with the help of data and features identified on the data path of an AI Application Service.

In the second year, the three tasks described in this report will continue the investigation on AI methods and solutions from the following perspectives: First, utilization of edge for AI service will be investigated further, including optimization of intermediate data and using an edge for the training. Furthermore, the privacy preservation at the AI on the edge will be investigated. Second, an AI solution will be developed to take more benefits from the cloud and IoT technologies. Integration of AI applications with IoT devices will become faster, easier and more intuitive for developers aiming to use IoT data into their AI services. This will be enabled by the use of a microservices architecture style, container technology and cloud-native infrastructure based on Kubernetes, which will be used in the use cases implementation including Digital Twins.
References


Abbreviations

WP  Work Package
IoT  Internet of Things
AI   Artificial Intelligence
ML   Machine Learning
VGG  Visual Geometry Group, name of an AI model for image classification
YOLO You Look Only Once, name of an AI model for object detection
CPU  Central Processing Unit
GPU  Graphics Processing Unit
GUI  Graphical User Interface
REST Representation State Transfer
HTTP HyperText Transfer Protocol
UC   Use Case
QoS  Quality of Service