Ref. Ares(2022)4860863 - 04/07/2022

Energy Efficient Manufacturing System Management

D2.2 – Final Version of EnerMan Data Collection and Management Components

Date	: 30/06/2022
Deliverable No	: 2.2
Responsible Partner	: ISI
Dissemination Level	: Public

HORIZON 2020
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Short Description
This deliverable gives an overview of the EnerMan project final activities of WP2 T2.1 and T2.4. It consists of 5 sections that provide a description of the EnerMan edge/end node architecture and updated on the execution environment, the implementation of intelligence related applications as well as the support for Federated Learning within the edge node. Also, the final and updated activities for security applications on cyber-attack prevention and detection as well as a demonstration of some of the described applications in action.

Project Information	
Project Acronym:	EnerMan
Project Title:	ENERgy-efficient manufacturing system MANagement
Project Coordinator:	Dr. Ing. Giuseppe D'Angelo CRF giuseppe.dangelo@crf.it
Duration:	36 months

Document Information & Version Management			
Document Title:		Final version of EnerMan Data Collection and Management Components	
Document Type:		Report and Demonstration	
Main Author(s):		Apostolos Fournaris (ISI) Evangelos Haleplidis (ISI) Thanasis Tsakoulis (ISI) Aris Lalos (ISI)	
Contributor(s):		Giannis Morianos (TSI) Andreas Miaoudakis (STS) Panagiotis Rodosthenous (ITML) Mina Marmpena (ITML) Konstantinos Bouklas (ITML) Christian Capezza (UNINA) Antonio Lepore (UNINA) Biagio Palumbo (UNINA)	
Reviewed by:		UoP team SUPM team	
Approved by:		Ing. Giuseppe D'Angelo (CRF)	
Version	**Date**	**Modified by**	**Comments**
V0.1	15/04/2022	Apostolos Fournaris (ISI)	ToC and first draft
V0.2	01/06/2022	Mina Marmpena (ITML)	ITML input provided
V0.3	05/06/2022	Apostolos Fournaris (ISI), Thanasis Tsakoulis (ISI), Evangelos Haleplidis (ISI)	ISI input provided
V0.4	08/06/2022	Andreas Miaoudakis (STS)	STS security input provided
V0.5	10/06/2022	Giannis Morianos (TSI)	Security related TSI inpuf provided

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

V0.6	12/06/2022	Apostolos Fournaris (ISI) Aris Lalos (ISI)	Final ISI input provided
V0.7	15/06/2022	Apostolos Fournaris (ISI), Christian Capezza (UNINA), Antonio Lepore (UNINA), Biagio Palumbo (UNINA)	First full version of the deliverable
V0.8	20/06/2022	Apostolos Fournaris (ISI)	Deliverable ready for internal review
V0.9	30/06/2022	Apostolos Fournaris (ISI)	Deliverable internal review comments addressed and final deliverable ready for submission
V1.0	01/07/2022	Kubra Yurduseven (INTRACT)	Format review
V1.1	04/07/2022	Ing. Giuseppe D'Angelo (CRF)	Submitted version

Disclaimer
This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation, or both. The publication reflects the author's views. The European Commission is not liable for any use that may be made of the information contained therein.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

TABLE OF CONTENT

EXECUTIVE SUMMARY 9

1. INTRODUCTION 10

2. ENERMAN INTELLIGENT END NODES/EDGE ARCHITECTURE 11

2.1. Overall Architecture and Usage 11

2.1.1. The Sensor architectural view 12

2.1.2. The Control architectural view 14

2.2. Edge/End Node Execution Environment 15

2.3. Interaction with the EnerMan framework - Communication Interfaces 16

3. EDGE NODE DATA COLLECTION, DATA PROCESSING DESIGN AND COMPONENT IMPLEMENTATION 17

3.1. Sensor based Data collection mechanism 17

3.2. Data Harmonization 21

3.3. Data post processing –Energy Consumption prediction 23

3.3.1. Dataset 23

3.3.2. Problem formulation and preprocessing 25

3.3.3. Designed and Developed Neural Networks models 26

3.3.4. Other approaches 33

3.4. Intelligence Acceleration using Hardware assistance 33

3.4.1. The open-source framework hls4ml 33

3.5. Federation based processing mechanism 38

3.6. Edge node functionality and Configuration updates (reconfiguration) 40

4. EDGE NODE SECURITY ASPECTS 42

4.1. Security Architecture 42

4.2. Security Mechanisms 43

4.2.1. Cybersecurity Attack Detection 43

4.2.2. Cybersecurity Attack Prevention 44

5. DEMONSTRATION REPORT 50

5.1. Federated Learning edge node processing mechanism 50

5.2. Hardware Security Token (HST) capabilities and Secure communication using Quantum-Safe TLS 57

5.2.1. HST concept and overall usage 57

5.2.2. Adding Quantum Safe TLS 1.3 in the HST 60

5.3. AI Industrial Intrusion Detection 64

5.3.1. Set-up the board 64

5.3.2. Set-up the host 64

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

5.3.3. Train a quantized MLP with Brevitas 64
5.3.4. Import model into FINN and compare it with Brevitas execution 66
5.3.5. Synthesis of the accelerator and generation of the bitfile 67
6. CONCLUSION 71
REFERENCES 72

TABLE OF FIGURES

Figure 1 EnerMan Intelligent Edge node Sensor architectural view 12
Figure 2 EnerMan Intelligent Edge node Data collection and pre-processing paths. 13
Figure 3 EnerMan Intelligent Edge node Control architectural view 14
Figure 4 Updated EnerMan Intelligent Edge Node Execution Environment 15
Figure 5 Intelligent Edge Node interactions 16
Figure 6 Ultra96v2 platform 17
Figure 7 Heterogeneous sensors supported 17
Figure 8 Edge data collection block diagram 18
Figure 9 On-board sensor data stream monitors 19
Figure 10 Rudimentary web server for collecting datasets 20
Figure 11 Simple protocol for sending dataset files 20
Figure 12 Data harmonization workflow between the edge and the cloud 21
Figure 13 UML diagram of the EnerMan harmonization module 22
Figure 14 Five first samples of the dataset 24
Figure 15 Various feature information 25
Figure 16 Data standardization 26
Figure 17 ANN structure 27
Figure 18 ReLU activation function 27
Figure 19 Number of parameters per layer 28
Figure 20 Training and Validation Process 29
Figure 21 Predicted vs Ground truth samples of test set 29
Figure 22 CNN structure 30
Figure 23 Tanh(x) activation function 31
Figure 24 MaxPooling Layer operation 31
Figure 25 Training and Validation Process 32
Figure 26 Predicted vs Ground truth samples of test set 32
Figure 27 yml file example 34
Figure 28 ANN configuration yml file 35
Figure 29 Latency of ANN design 35
Figure 30 Resource utilization of ANN design 36
Figure 31 Configuration yml file of CNN design 36
Figure 32 Latency of CNN design 37
Figure 33 Resource Utilization of the CNN design 37
Figure 34 Federation scheme 38
Figure 35 Federation learning process 39
Figure 36 Simple protocol for sending ML models 40

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Figure 37 Implementation details of federation process 40
Figure 38 Setup of the rpyc connection 41
Figure 39 Download and run functionality 41
Figure 40 The EnerMan Security Architecture 43
Figure 41 The Application Gateway overall architecture and the API Management. 44
Figure 42 Gateway Message Flow 45
Figure 43 Welcome screen on WSO2 APIM, showing the admin all the available API types. 45
Figure 44 Runtime Configurations for an API 46
Figure 45 APIs appearing on the developer portal 46
Figure 46 Available options for an API published on the devportal. 47
Figure 47 Configuration of a service provider with the different possible authentication protocols 48
Figure 48 Authentication Process and API protection 49
Figure 49 Initial setup of the server 50
Figure 50 Creation of the global model and test loader 51
Figure 51 Class model of our CNN 51
Figure 52 High level overview of server process 52
Figure 53 Server handling clients 53
Figure 54 Server initialization and connection from clients 54
Figure 55 Server termination 54
Figure 56 Client1 learning process 55
Figure 57 Client2 learning process 56
Figure 58 Client3 learning process 57
Figure 59 Created postquantum handshake version of TLS1.3 for embedded systems 62
Figure 60 Training loss per iteration 65
Figure 61 Test accuracy per iteration 65
Figure 62 The exported ONNX model in Netron 66
Figure 63 The steps towards the bitfile generation 67
Figure 64 Block design architecture in Vivado 68
Figure 65 Terminal Output 69

TABLE OF TABLES

Table 1. additional TLS1.3 Handshake codepoints for PQC schemes 63
Table 2 Metrcics of I2DS 69

This project has received funding from the European Union’s Horizon 2020
research and innovation programme under grant agreement **No 958478**

LIST OF ACRONYMS

AES	Advanced Encryption Standard
API	Application Programming Interface
BDAE	Big Data Analytics Engine
CPS	Cyber-Physical System
CPSoS	Cyber-Physical System of Systems
CWRU	Case Western Reserve University
DL	Deep Learning
DMZ	DeMilitarized Zone
DoA	Description of Action
DoS	Denial of Service
ENISA	European Union Agency for Cybersecurity
FIFO	First In First Out
FL	Federated Learning
FPGA	Field Programmable Gate Array
GPU	Graphic Processing Unit
HLS	High Level Synthesis
HST	Hardware Security Token
HTTP	HyperText Transfer Protocol
I2DS	Industrial Intrusion Detection System
ICS	Industrial Control Systems
IDS	Intrusion Detection System
iDSS	Intelligent Decission Support System
IETF	Internet Engineering Task Force
IFCA	Iterative federated clustering algorithm
IID	Independent and Identically Distributed
IIoT	Industrial Internet of Things
IP	Intellectual Property
IPS	Intrusion Prevention System
IR	InfraRed
JWT	JSON Web Token
MITM	Man In The Middle
ML	Machine Learning
MLP	MultiLayer Perceptron
MPSoC	MultiProcessor System on a Chip

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

MRT	Mean Radiant Temperature
OAuth	Open Authorization
ONNX	Open Neural Network Exchange
OT	Operational Technology
PKCE	Proof Key for Code Exchange
PKI	Public Key Infrastructure
PL	Programmable Logic
PLC	Programmable Logic Controller
PS	Processor System
QNN	Quantized Neural Network
ReLU	Rectified Linear Unit
REST	Representational state transfer
SCADA	Supervisory Control and Data Acquisition
SoC	System on a Chip
TLS	Transport Layer Security
UNSW	University of New South Wales
WP	Work Package
XRT	Xilinx RunTime
XSS	Cross-Site Scriptiong

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

EXECUTIVE SUMMARY

This deliverable is focused on the final activities of wp2 and especially in the activities of task 2.1 (T2.1) and task 2.4 (T2.4). The deliverable presents complementary components and modules as well as additions to the overall EnerMan intelligent edge node architecture that aim to fulfil the actions prescribed in the DoA mainly for tasks 2.1 and 2.4. After a short introduction, we provide a recap as well as updates on the EnerMan intelligent edge node architecture and overall functionality. Then in section 3 we describe in detail the activities that have been performed to realize this additional to d2.1 functionality and architectural components including any necessary link with the more theoretical work on t2.2 and t2.3, the data harmonization mechanism updates and the design and implementation flow of hardware assisted intelligence (deep learning and federated learning) operations. Section 4 is focused explicitly on the security activities taking place mainly at the edge (reporting the activities of t2.4) but also any additions that are made to extend the EnerMan security to the overall EnerMan framework. This includes extensions of the security architecture, its components, and their functionality. Finally, section 5, describes indicative demonstrations/tutorials/hands-on for the various EnerMan intelligent edge node functionalities.

Since the deliverable is a continuation of D2.1 where the preliminary activities of D2.1, D2.2 (and partially D2.3) are reported, we tried not to repeat the same information that appear in D2.1. There are several exceptions to this rule for activities that require some background information to become available to the reader in order to showcase the additional work that has been done till m18.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

1. INTRODUCTION

Work Package 2 of the EnerMan project is focused on the EnerMan framework activities that are performed at the edge layer of the Industrial Internet of Things (or the Factory of the Future concept). Thus, it is meant to achieve appropriate data collection from the various in-field deployed devices within the industrial environment or from dedicated data collection points that are already deployed at the industrial site. They concept of edge computing that we are promoting in the EnerMan project is also focused on performing as much computation as possible at the edge without the need to include the extra delay of transmitting the data to a cloud/system layer. This concept is manifested in the WP2 activities through the design and implementation of an EnerMan Intelligent edge node that not only collects data but also performs computation/processing on those data as well. The overall concept has already been described in D2.1 and includes the necessary computation/execution environment that will support the required by the industrial/factory partners, computations (additionally supporting any future such computations that may arise) as well as the implementations of prescribed software and hardware assisted computations. In D2.1 we focused our analysis on the design and implementation of the EnerMan intelligent edge node execution environment, some initial intelligent operations related to the execution environment (that are expanded and further refined in a dedicated D2.3 deliverable) and the preliminary security architecture. In this deliverable, we report any refinements on the execution environment and the data harmonization mechanism, but we mostly focus on the final data collection mechanisms supported by the EnerMan Intelligent Edge node as well as the implementation/support of intelligence based (e.g., ML/DL, federated learning) research work done mostly in T2.2. Apart from that we also provide the final security architecture and its usage for cybersecurity attack prevention and detection (complementing the D2.1 reported work).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

2. ENERMAN INTELLIGENT END NODES/EDGE ARCHITECTURE

2.1. Overall Architecture and Usage

As already described in D2.1 the EnerMan Intelligent Edge node architecture is an heterogenous embedded system device that can perform multiple activities within the manufacturing infrastructure in an efficient manner. This edge device collects the data that are provided by various sensors existing inside the industrial domain, harmonize those data in order to be compatible with the data expectations required by the EnerMan big data analytics engine and in parallel also processes those data so as to offer the EnerMan platform as well as the operator appropriate information that will help them make informed decisions on the optimal energy sustainability options.

The Data Collection and Processing operation that is performed at the edge using the EnerMan intelligent edge node, is efficient, versatile/flexible to the end user needs and easily configurable. As described in D2.1 and the EnerMan Description of Action, the EnerMan Edge node is able to collect and process the following data:

- **Machine energy consumption**
- **Multiple sensory data (temperature, pressure, humidity, etc.) from existing pilot deployed sensors**

The intelligent functionality that the EnerMan edge node can provide is the following:

- **Machine Health status monitoring to detect anomalous, faulty states of a given machine.** Such states can impact the energy consumption of a given machine and eventually the energy sustainability of the overall factory.
- **Local Data energy consumption predictions.** We consider important been able to provide to the factory operators in-field, local, coarse grain energy consumption predictions to help the operator visualize energy consumption trends that can impact locally the production (eg. production on a single room or on a single array of machines etc). This mechanism can complement the systemwide EnerMan energy consumption prediction mechanism and fine tune the prediction results.
- **Data Completion (when missing data appear in a timeseries).** Given that in any Industrial system there may be loss of single sensor data values within a given timeseries, the EnerMan intelligent Edge node data collection mechanism is able to use DL models (based on autoencoders) in order to complete missing data values within a timeseries period. This is described algorithmically in more details in D2.3

The EnerMan edge node architecture for Data collection and processing as already described in the preliminary version of T2.1 outcomes (the deliverable D2.1) is meant to be used in various ways. Also, the node should be able to process data in a fast/efficient manner but also the node should be easily adjustable/flexible so as to be readily updated and be adapted in general to various different industrial environments. To achieve these goals, we use both hardware (for acceleration purposes) as well as software means.

The final embedded system platform that covers best all aspects of the EnerMan edge node is a MultiProcessor System on Chip (MPSoC) embedded system device that can host in its core, multiple processors (usually multicore processors) to achieve efficiency but also specialized hardware components for specific applications (e.g., Graphic Processing Units (GPU) or real-time processors). To further support hardware reconfigurability, our hardware platform choice is embedded system

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

MPSoCs that include in their SoC architecture, reconfigurable hardware programmable logic in the form of an FPGA fabric. The latest FPGA manufacturer solutions are perfectly capable of supporting the above-described setup. In EnerMan we opt for Xilinx manufacturer devices focusing on the Xilinx -AMD Zynq Ultrascale+ MPSoC design as this is realized in two characteristic embedded system boards i.e., the Xilinx-AMD ZCU 104 or 102 development board and the Avnet ULTRA96 development board that both use the Xilinx-AMD Zynq ULTRASCALE+ ZU9EG MPSoC.

The overall EnerMan Intelligent Edge node architecture is designed to support the above requirements. The architecture has two different views, the sensor data collection view, and the control view

2.1.1. *The Sensor architectural view*

This view includes all the necessary architectural components of the EnerMan Intelligent Edge node that allow it to act as an edge data collection and pre-processing module of the EnerMan overall architecture. Some of these components have been preliminary described in D2.1 but till the end of the WP2 (reported in the current deliverable) new components have been added and the existing ones have been updated in order to better capture the specification of the EnerMan overall architecture. An overview of the EnerMan Intelligent Edge node sensor view can be seen in Figure 1

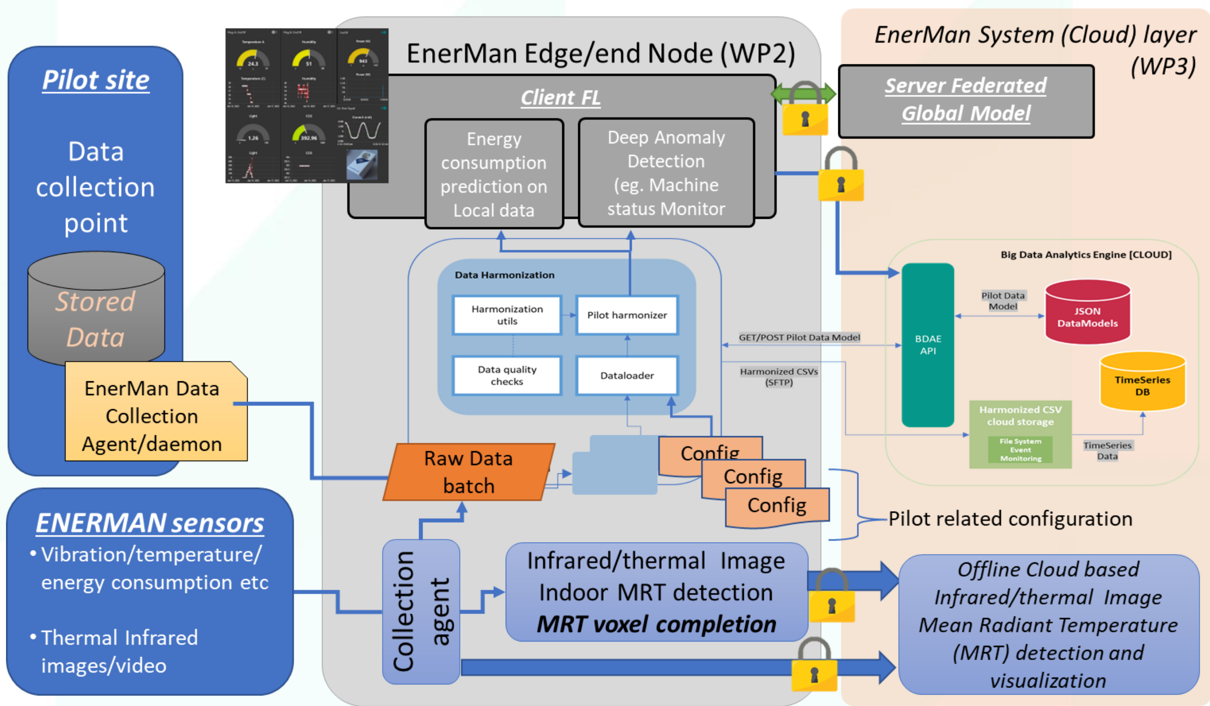

Figure 1 EnerMan Intelligent Edge node Sensor architectural view

The EnerMan Intelligent Edge node has three different mechanisms to collect and forward data to the rest of the EnerMan architecture and more specifically to the big data analytics EnerMan engine (designed and implemented in the context of the WP3 activities). The first approach is direct sensor data collection from the various industrial pilot sites. This is practically done by installing a dedicated EnerMan data collection agent/daemon on the pilot data collection site. This component is meant to constantly monitor the industrial data collection point for new data and when such data are identified to forward them to the EnerMan Intelligent Edge node for further processing and analysis. Given the various EnerMan pilot use cases and the EnerMan pilots' feedback provided in WP1 and WP6, the sensory data are stored in the industrial sites as csv files that are periodically generated when enough

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

of incoming sensor values are aggregated. The EnerMan daemon detects such newly generated csv files and forwards them accordingly to the EnerMan Intelligent Edge node. As expected, when new data reach the EnerMan Intelligent Edge node the Data harmonizer is automatically executed so as to harmonize the data as needed for the overall EnerMan framework.

Apart from the above, industrial site driven data collection mechanism, the EnerMan Intelligent Edge node is capable of deploying its own end device EnerMan sensor using various different industrial communication protocols (e.g MQTT, Backnet, Modbus, IEE802.15.4 etc) in order to collect in-field sensor measurements (without relying on the existing industrial sensors or by complementing existing industrial sensors). This approach enables more direct data collection compared to the previous collection mechanism.

The third data collection approach that the EnerMan Intelligent Edge node is employs, is based on data that are generated within the node itself after some pre-processing from the first of second data collection approach. While these are not raw data (but rather metadata or features) they are still forwarded to the EnerMan system and more specifically to the big data analytics engine. These metadata are mainly the outcome of the EnerMan Intelligent Edge node supported intelligence and can be sensor data predictions (e.g., energy consumption predictions) as well as anomalous machine behaviour that can be attributed to machine faulty states (evaluation of machine health status).

Note that from a research perspective, we are capable of supporting intelligent processing (ML and/or DL based) that can follow the Federated Learning paradigm. This approach assumes that we can deploy within the Industrial environment several EnerMan Intelligent Edge nodes that collect data and monitor a subset of the overall factory machines (eg, machines within a given room or of a given industrial process). Such nodes are acting as clients that collect similar data but due to the limited dataset size cannot achieve individually significant inference (classification/prediction/imputance) accuracy. Using the Federated Learning paradigm, we are able to aggregate the various clients (i.e., individual EnerMan Intelligent Edge nodes) ML/DL models in order to create a global federated supermodel (with increased accuracy) that can be then forwarded back to the clients so as to help them make more accurate inferred results. While the Federated Learning algorithmic work is described in D2.3, in D2.2 (i.e., the current deliverable) we provide the practical backbone that allows the algorithms to work in a close to real environment (assuming an IP network communication infrastructure). An overall view of the three above described paths can be seen in the following figure (Figure 2)

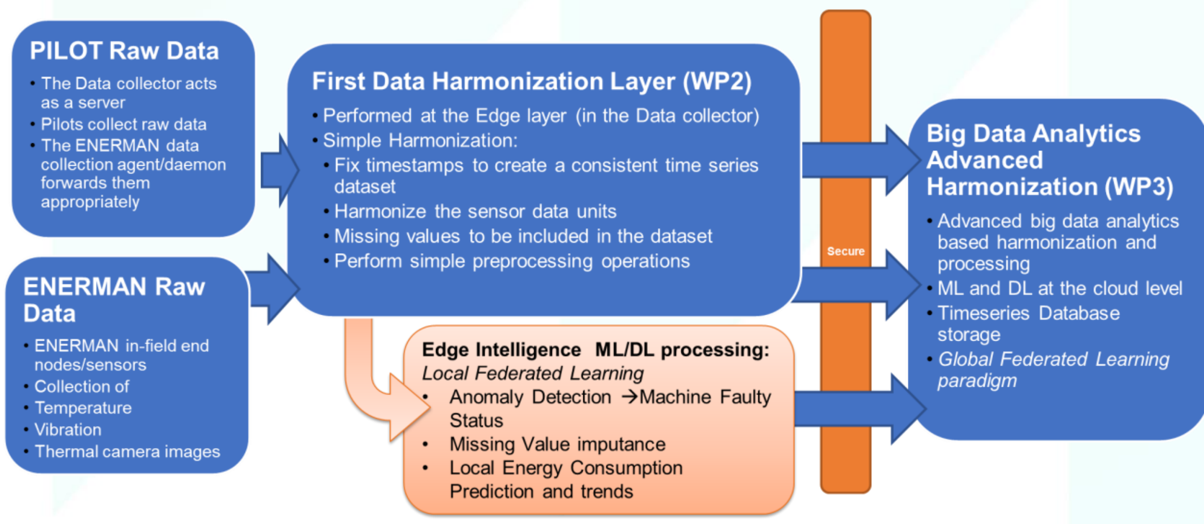

Figure 2 EnerMan Intelligent Edge node Data collection and pre-processing paths

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

It should be noted that since the EnerMan framework may not necessarily reside as a whole in the Industrial site premises (it may be offered as private or public cloud-based service), there a significant risk of cybersecurity attacks on the data to be sent or received as well as on the Industrial infrastructure that is associated with the EnerMan platform as a whole. Thus, all data transmissions to or from the EnerMan Intelligent Edge node are secured while the overall system is designed to prevent cybersecurity attacks as well as detect such attacks when they are initiated.

2.1.2. *The Control architectural view*

This view of the EnerMan Intelligent Edge node is exploring the control/configuration capabilities of the node itself. Given the task and DoA description regarding control on the EnerMan Intelligent Edge node, we can split this control type into two different paths. The first path has to do with the control of the EnerMan Intelligent Edge node itself while the second path has to do with the capability of the EnerMan Intelligent Edge node to propagate industrial device configuration decisions that stem from informed choices made by the industrial personnel (with the assistance of the EnerMan i-DSS subsystem). Thus, in the first adopted path the control target is the EnerMan Intelligent Edge node while in the second adopted path the control target is some industrial subsystem (or machine). The overall control architectural view can be seen in Figure 3

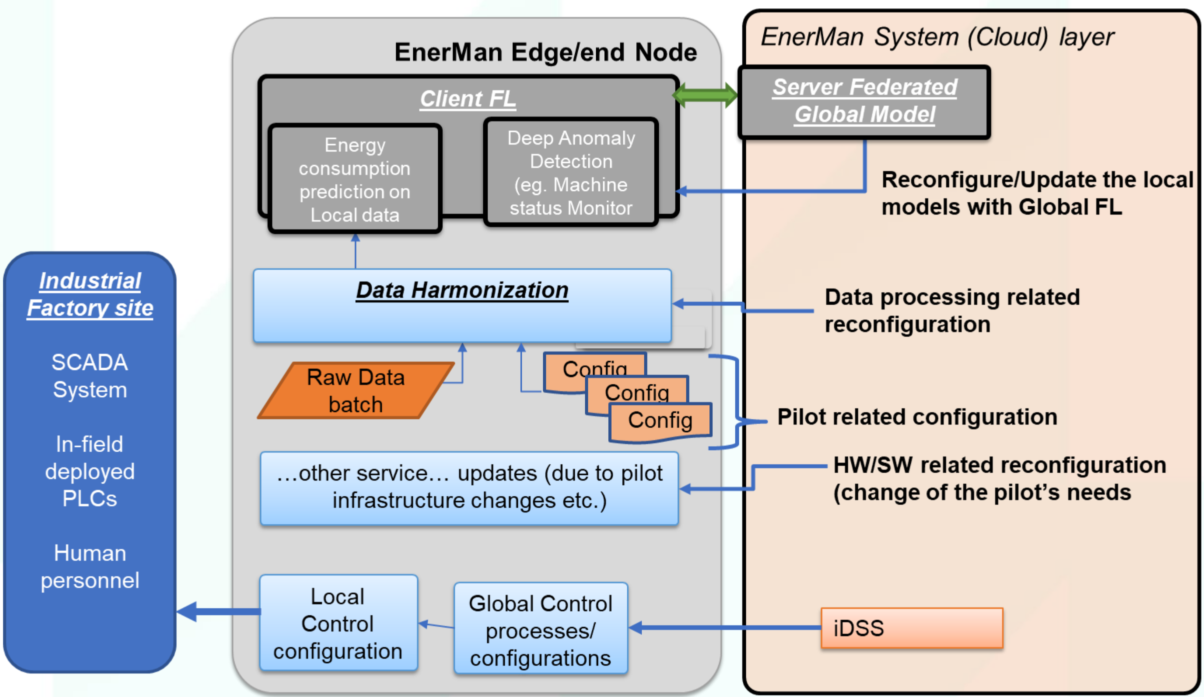

Figure 3 EnerMan Intelligent Edge node Control architectural view

As seen in Figure 3, there are several ways that can be used to adjust the configuration of the EnerMan intelligent Edge node. Algorithmically, the intelligence of the node can be controlled using the Federated Learning paradigm when the client ML/DL models are updated/enhanced from the Global Federated Model (that takes into account all local client models to achieve high accuracy). More information on this concept can be found in D2.3. Apart from that, the Data Harmonization process is customized according to the Industrial factory (eg. any EnerMan pilot) at hand, thus appropriate configuration files can be provided to the Data harmonization mechanism in order to set it up for a given industrial domain and factory. Finally, it should be taken into account that the EnerMan intelligent Edge node device is a heterogenous MPSoC with multiple processing capabilities in the form of CPU, GPU as well as FPGA fabric. This allows for a fundamental reconfiguration that can be extended

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

to software and hardware IP cores. This latest approach has been discussed in D2.1 where the EnerMan Intelligent Edge node execution environment is detailed.

Given these three ways of configuration control over the EnerMan Intelligent Edge node, we have designed and implemented a mechanism (extended the existing execution environment of the EnerMan Intelligent Edge node) that can support this reconfiguration in a remote way. This is in line with the overall EnerMan concept, where configurations are provided by the EnerMan system to the edge nodes and then they are propagated (if needed) to the industrial actuators/controllers (if those are available). The mechanism is further described in section 3 of this deliverable.

Regarding the EnerMan Intelligent Edge node control configurability, it shouldn't be omitted its capability to collect through the remote configuration mechanism user chosen configurations (Global control processes/configurations) using the EnerMan i-DSS component that are communicated to the Industrial control/actuation. This procedure is supported by the EnerMan Intelligent Edge node as mentioned above but it is also enhanced with local control configurations (that are further described in D2.4). However, in practice due to the sensitive nature of Industrial operations within the EnerMan pilot sites, it would be extremely difficult to actually deploy such configurations in the pilot sites. Thus, we envision to showcase this kind of configuration deployment either as dedicated control guidelines to the factory in-field workers or as configurations to be realized in a virtual factory.

2.2. Edge/End Node Execution Environment

In Deliverable D2.1 we have provided detailed description and usage of the EnerMan intelligent edge node execution Environment that also included demonstration of its creation using the Xilinx tools as well as demonstration of its usage. Although the Execution Environment has practically been already finalized by M14 of the project (as reported in D2.1), we have nevertheless provided some additions to its overall structure so as to increase its simplicity and usability. More specifically, we have migrated all firmware and OS functionality from the original Petalinux environment (supported through dedicated tools by Xilinx-AMD) to a more open Debian distribution that can simplify the introduction of new OS and application components. The new version of the execution environment is fully compatible with the original one and in practice we can use both approaches. The updated version of the execution environment can be seen in Figure 4

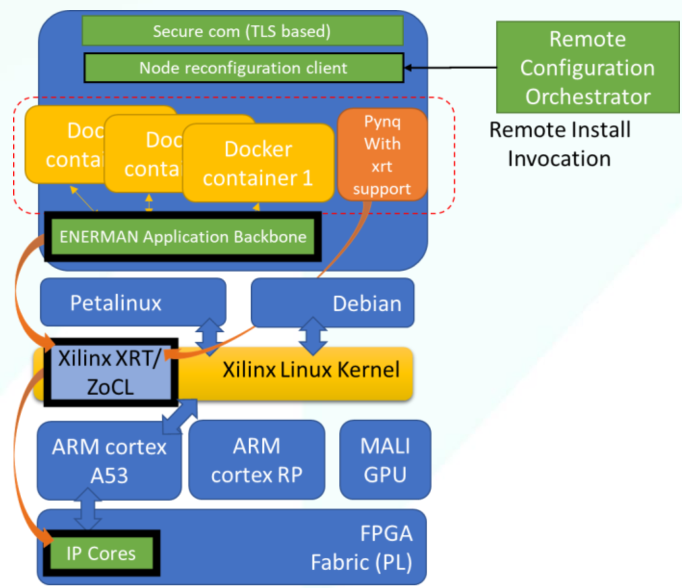

Figure 4 Updated EnerMan Intelligent Edge Node Execution Environment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

As can be seen from the above figure, a new module has been added in the environment, denoted as Node reconfiguration client, that is meant to support the configuration control view of the EnerMan Intelligent Edge node. The functionality and usage of such a module (along its interaction with a necessary reconfiguration server i.e., the remote configuration orchestrator) is described in more details in subsection 3.5

Apart from the above, as it has briefly already been described, the final Edge node Execution Environment does not necessary rely on the Petalinux distribution (that has several usability drawbacks) but is rather built on top of the generic Xilinx Linux kernel. This allows the usage of other Linux distributions like the Debian distribution that is currently used on the EnerMan Intelligent Edge node device (using the Avnet ULTRA96 board or the similar Xilinx ZCU104 board).

2.3. Interaction with the EnerMan framework - Communication Interfaces

The EnerMan intelligent edge node interaction with the rest of the EnerMan framework is mainly made with the actual industrial pilots sites (where data are generated) and the Big Data Analytics engine, BDAE, (designed in T3.1) that acts as the prime data collector and consumer, as can be seen in Figure 5. Pilots provide the main input to the EnerMan framework and interact with the Intelligent Edge Node, identified as step 1 in Figure 5, by providing generated data, either by having sensors providing direct info, or by providing their own files, either by copying them into one specific folder to be captured by a folder daemon, or by uploading it on a Web Server on the node. This interface is discussed in detail in Section 3.2. As also described in section 2.1, data can be collected using the EnerMan Intelligent Edge node end devices (sensors) that use their own communication protocol related interface.

The Intelligent Edge Node takes the raw data input, runs it through an initial harmonization process in order to process this raw data and turn it into a unified and standardized data representation. To perform the harmonization, the module uses an HTTP GET request to read JSON Data Model files from the BDAE API and an HTTP POST request to modify their content. These Data Models contain metadata information, dedicated to each pilot's data characteristics. Finally, the harmonized data in csv format are securely transferred to the BDAE server using the SSH File Transfer Protocol, identified as step 2 in Figure 5. This interface is discussed in more details in Section 3.2.

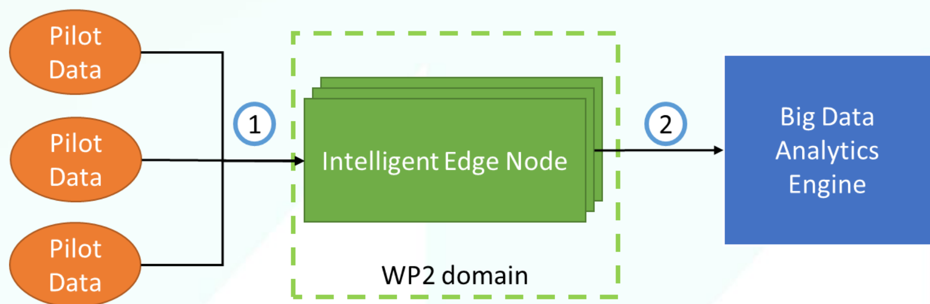

Figure 5 Intelligent Edge Node interactions

Note, that in the presence of security measures, the above interactions can be altered so as to fully support the EnerMan security capabilities. More specifically, while the original data transmission between the pilot site and the EnerMan Intelligent Edge node is performed using the TCP protocol, using the EnerMan security realized capabilities, the transmission is upgraded to quantum safe TLS 1.3 employed a dedicated Hardware Security Token that is integrated inside the EnerMan Intelligent Edge node.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

3. EDGE NODE DATA COLLECTION, DATA PROCESSING DESIGN AND COMPONENT IMPLEMENTATION

3.1. Sensor based Data collection mechanism

The Edge data collection has been based around the core reconfigurable platform of AvNet Ultra96v2 (Figure 6) that defines a potable, multi-protocol, multi-function data aggregator and edge processing facility.

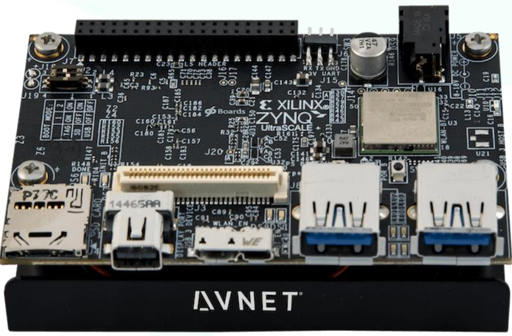

Figure 6 Ultra96v2 platform

The initial Petalinux environment and OS kernel have been upgraded to a customized porting of the Bullseye Debian distribution (Debian 11), over which, a number of data acquisition protocols and mechanisms are supported. Initially, based on the various indicative project use cases, it supports a core set of industrial standards, such as Modbus, KNX and BACNet, as well as high-end, industrial grade, high-frequency IEPE vibration sensors and relative acquisition interface boards. Besides these, modern IIoT protocols like MQTT and CoAP, are supported as well, over wired, or wireless media, including Bluetooth/BLE, IEEE 802.11b/g/n, IEEE 802.15.4 TSCH or CSMA, in 2.4GHz or long-range, sub-GHz bands (IEEE 802.15.4g). In this way, it can directly communicate with many available standard COTS sensor or actuator devices, but also with modern, low power, multi-sensor IoT components (Figure 7).

Figure 7 Heterogeneous sensors supported

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

The block diagram of the main edge node, data collection mechanism is presented in Figure 8. The downstream communication to field devices can be done through all feasible combinations of protocol components and physical medium interfaces, according to the particular use case and the field device specification. The edge node can also retrieve real-time data streams, if needed, from upstream links, using typical industrial protocols over TCP/IP (Ethernet or WiFi).

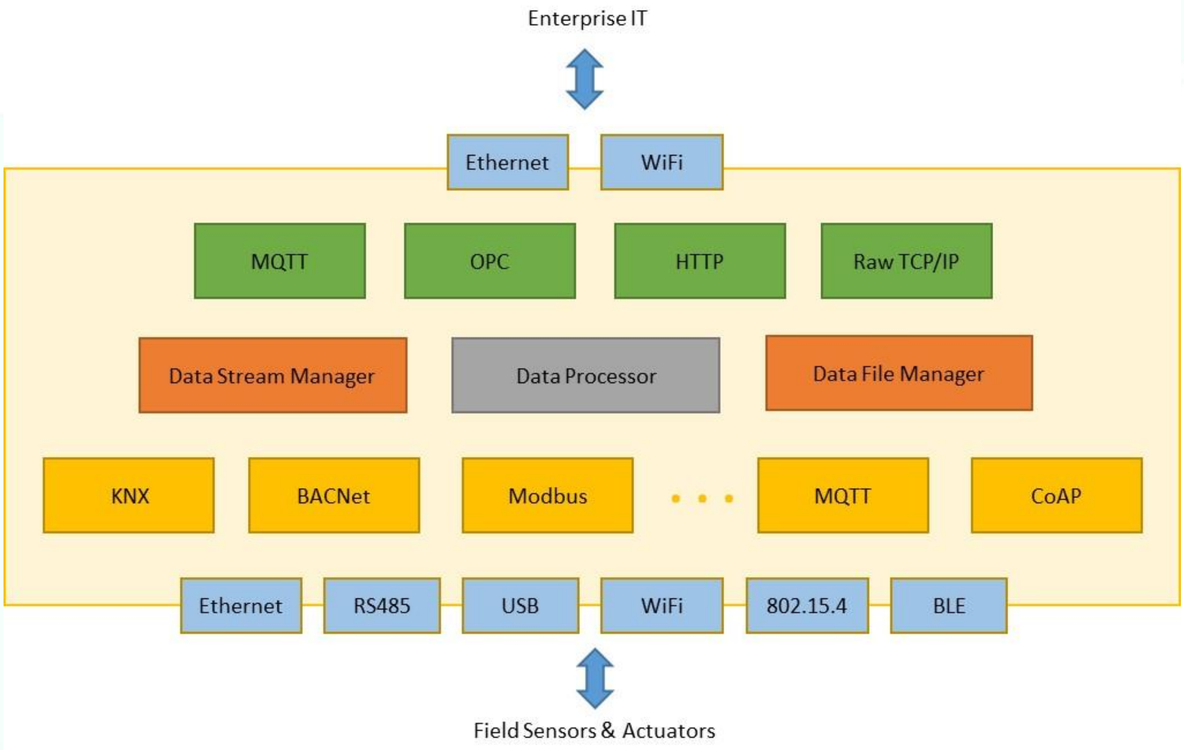

Figure 8 Edge data collection block diagram

The management, collection setup and routing of data towards the Data Processing subsystems is implemented by the Data Stream Manager and Data File Manager components, while the sensor data streams can be also accessible directly through a monitoring web interface implemented on board by the Data Stream Manager (Figure 9).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Figure 9 On-board sensor data stream monitors

In addition, in order to support different data collection practices e.g., in cases that security, privacy and confidentiality policies do not allow for direct acquisition of sensor data streams, data can be also collected and stored in text or excel files in regular periods and in chunks. This file collection can be done by uploading data to a specific folder location and collecting them through some folder daemon mechanism that checks whether new files have been inserted, or through a web interface, that subsequently sends the uploaded file to the appropriate next step in the EnerMan process.

Towards that end goal, two basic Python programs have been developed that are able to run in the EnerMan Intelligent Edge node. The first program is a folder daemon that has a constant network connection with the EnerMan Intelligent Edge node and is deployed in the data repository of any industrial manufacturing system. The daemon continuously monitors a specific folder for new files. Once a new file has been copied into the folder the daemon will detect it and will send it over the TCP connection to EnerMan Intelligent Edge node. The second program is a rudimentary web portal, based on python's Flask web server, where pilots can upload their files. Once these files are uploaded, they are sent via the same mechanism as the folder daemon to the EnerMan Intelligent Edge node data harmonization mechanism.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Figure 10 Rudimentary web server for collecting datasets

To correctly transport any number of files present into the directory or submitted by the dataset producer, we designed a very simple application protocol, using TCP as transport protocol (which as part of T2.4 is upgraded to a highly secure TLS1.3 communication flow), as shown in Figure 11. The client will send the number of files along with the total length of the segment and then iteratively for each of newly discovered files will send the file number along with its filename's length, the filename, the file's length, and finally the file. The server, on the other end, supports connection from multiple clients, and will forward the received data to the next step in the EnerMan process.

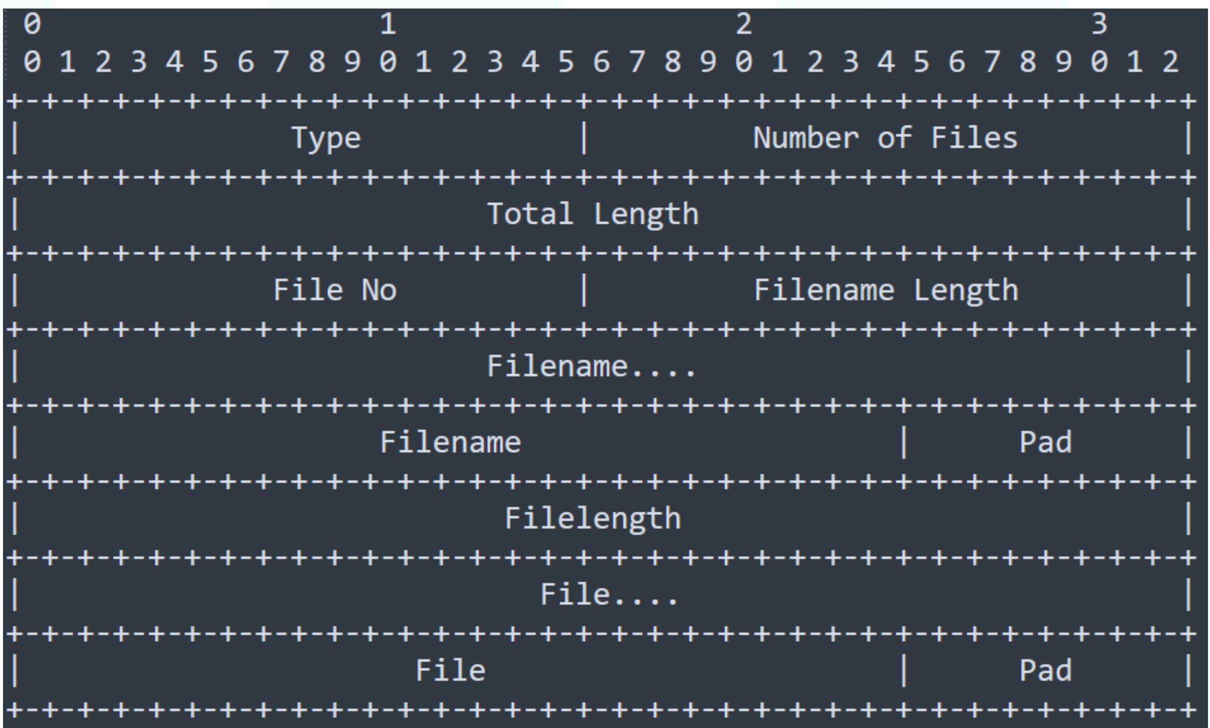

Figure 11 Simple protocol for sending dataset files

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

3.2. Data Harmonization

Data harmonization workflows aim to process the raw data collected at the edge nodes and turn them into a unified and standardized data representation. This is the first layer of pre-processing, taking place at the edge nodes, so that the various data sets follow a standardized form when they are ingested in the cloud, where the second layer of pre-processing will be applied. Data harmonization is an important stage in a data preparation pipeline since the raw data are produced in custom formats which are diverse and pose a challenge for the application of advanced pre-processing techniques. For example, raw provided datasets can have varying file formats (e.g., .csv, .xlsx, .eps) and missing values indications (e.g., N/A, #NA, -1.#IND, -NaN, -nan , N/A, NA, NULL, NaN, n/a, nan, null), non-standard headers (e.g., multirow, with spaces, too long), duplicates, localized indications for decimal and thousands, heterogeneous timestamp formats and time zones, empty rows or columns. These are all issues that need to be addressed at the collection edge nodes, so that cloud processing functions can be applied uniformly.

The Data Harmonization module (initially presented in D2.1 Preliminary version of EnerMan Data Collection and Management Components, M12) handles all the above-mentioned data harmonization requirements. Figure 12 depicts the harmonization module deployed in the data aggregator of the EnerMan intelligent nodes and how it communicates with the Big Data Analytics Engine (BDAE) cloud server (more details on the BDAE architecture can be found in D3.1 Big Data Collection and Analytics Platform and Analytics Report, M18). The module uses a GET request to read the JSON Data Model files from the BDAE API and a POST request to modify their content whenever the schema of the raw data changes. These Data Models contain metadata information, dedicated to each pilot's data characteristics, to guide the harmonization processing. After the harmonization is complete, the harmonized data in CSV format are securely transferred to the BDAE server using the SSH File Transfer Protocol, which ensures data integrity via encryption and cryptographic hash functions, while it authenticates both the client and the server.

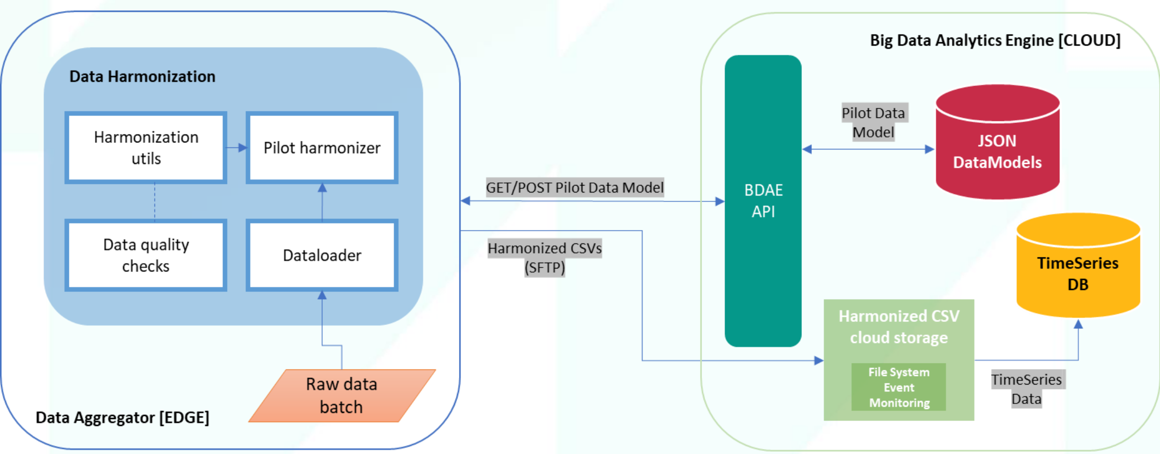

Figure 12 Data harmonization workflow between the edge and the cloud

The harmonization module is written in Python and is comprised of the following components: a *DataLoader* class to load the raw batch data, a *BaseHarmonizer* which serves as an abstract class to enforce the implementation of the required harmonization methods in the child harmonizer classes. A dedicated harmonizer class is built for each organization as presented in Figure 13, e.g., *CRFHarmonizer*, *YIOTISHarmonizer*, *3DNTHarmonizer*, *STOMANAHarmonizer*. This separation was

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

necessary because of the heterogeneous data representations of the different pilot organizations. The harmonizers share a lot of common fields and methods, but they also employ different approaches to implement them behind the scenes. Following a *one size fits all* approach, was rejected because it would lead to a heavy weight monolithic class, difficult to maintain. Essentially, the motivation was to have some extra modularity and repetition in the edge implementation, but to completely avoid it in the cloud processing.

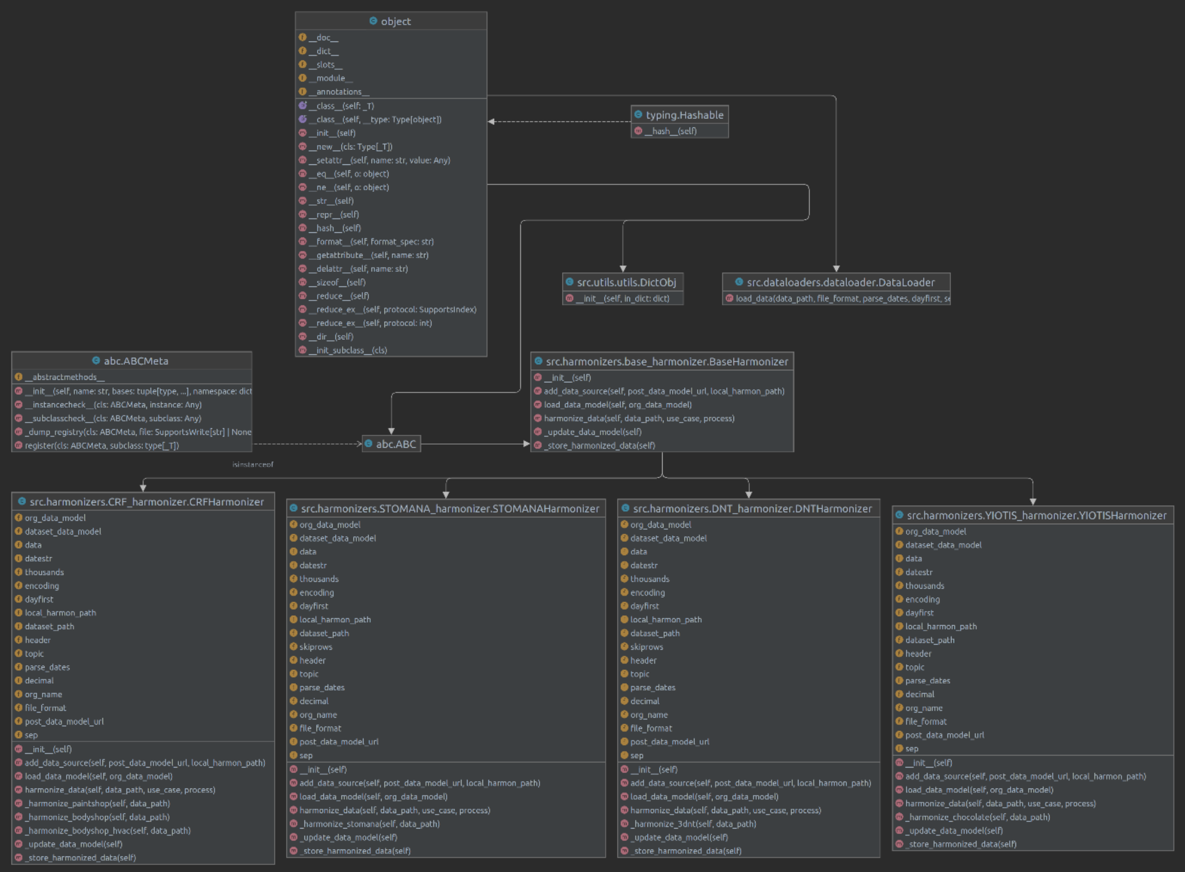

Figure 13 UML diagram of the EnerMan harmonization module

The final component of the module is a library with harmonization functions and quality checks operations which is available to all harmonizers:

```
NAME
    src.utils.harmonization_utils - # -*- coding: utf-8 -*-
FUNCTIONS
    clean_column_names(data, new_features)
        Clean column names and returns a list of the standardized column names.

    clean_dups_nan(data, missing_indication)
        Drop duplicates and empty rows.

    concat_date_time_columns(data)
        Concat date and time column in a single column.

    drop_unnamed_empty_columns(data)
        If dataframe has unnamed empty columns drop them.

    fix_column_names_nonum(data)
        If a column name doesn't contain its corresponding number, add it.
```


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
get_new_features_dict(new_feature_structure, new_features, std_column_names)
   Creates a dict of new features dicts and standardized names.

local_to_utc(tz_info, df)
   Convert local timezone datetime columns to UTC.

merge_new_features(features, new_features_dicts)
   Merge the existing features' dictionary with the new features.

optimize(df, datetime_features)
   Optimize floats, ints and objects.

replace_with_booleans(data, contains_boolean)
   If dataset contains boolean column replace ON OFF with 1 0.

resampling_dups(resample, data)
   Handle duplicate timestamps (keep first, last, avg, or median.

standardize_cols(data, features_dict)
   Set standardized feature names and dtypes.

to_utc_aware(df)
   Make UTC-aware datetime columns.

utc_to_local(tz_info, df)
   Convert UTC timezone columns to local timezone.

update_org_data_model(org_name, post_data_model_url, org_data_model)
   Write the updated organization's data model to <ORG>.json file.
```

3.3. Data post processing –Energy Consumption prediction

The problem of energy consumption prediction is rising nowadays as a crucial one due to the high impact on various operations at buildings or whole factories. Acquiring information for possible high energy consumption in the near future, could assist the administrators to take the appropriate countermeasures in order to adapt the building/factory operations. Moreover, a possible outlier could cause future damage in the factory machines thus it needs to be detected in time. In EnerMan, we address this problem at the various layer of the EnerMan architecture. At the Edge layer, in the EnerMan Intelligent edge node, we introduce a local energy consumption prediction mechanism that relies on Deep Learning techniques and does not need a fine-grained energy consumption model for each Industrial environment machine. In subsection 3.3 we provide all necessary design details on our approach while in subsection 3.4 we showcase how the generated DL models can and have been implemented using hardware acceleration.

3.3.1. *Dataset*

An enormous problem that is inherent in neural networks is the selection of the appropriate dataset which could boost the model's accuracy significantly. A dataset with poor quality (high noise, few features, small number of entries etc.) could never reach high levels of accuracy. We have chosen an open-source dataset from the relevant research literature [1] This choice was made in order to compare our models with some published results. We plan to use the same methodology described in this subsection to the EnerMan pilot datasets that fit the dataset requirements. The used datasets may be a little different from those that pilots could provide but the workflow is similar and the incorporation of them is a straight forward process.

The dataset is referred to a building near to the Chievres Airport, Belgium and consisted of 19735 samples and 28 features. Every sample in this timeseries dataset is recorded per 10 minutes and the features are analytically explained below

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

1. Appliances, energy use in Wh
2. Lights, energy use of light fixtures in the house in Wh
3. T1, Temperature in kitchen area, in Celsius
4. RH_1, Humidity in kitchen area, in %
5. T2, Temperature in living room area, in Celsius
6. RH_2, Humidity in living room area, in %
7. T3, Temperature in laundry room area
8. RH_3, Humidity in laundry room area, in %
9. T4, Temperature in office room, in Celsius
10. RH_4, Humidity in office room, in %
11. T5, Temperature in bathroom, in Celsius
12. RH_5, Humidity in bathroom, in %
13. T6, Temperature outside the building (north side), in Celsius
14. RH_6, Humidity outside the building (north side), in %
15. T7, Temperature in ironing room , in Celsius
16. RH_7, Humidity in ironing room, in %
17. T8, Temperature in teenager room 2, in Celsius
18. RH_8, Humidity in teenager room 2, in %
19. T9, Temperature in parents room, in Celsius
20. RH_9, Humidity in parents room, in %
21. To, Temperature outside (from Chievres weather station), in Celsius
22. Pressure (from Chievres weather station), in mm Hg
23. RH_out, Humidity outside (from Chievres weather station), in %
24. Wind speed (from Chievres weather station), in m/s
25. Visibility (from Chievres weather station), in km
26. Tdewpoint (from Chievres weather station), Â°C
27. rv1, Random variable 1, nondimensional
28. rv2, Random variable 2, nondimensional

The five first samples are illustrated in Figure 14. Appliance and lights are in Wh while temperatures are in Celsius and relative humidity is given as a percentage value.

	Appliances	lights	T1	RH_1	T2	RH_2	T3	...
0	60	30	19.89	47.596667	19.2	44.790000	19.79	...
1	60	30	19.89	46.693333	19.2	44.722500	19.79	...
2	50	30	19.89	46.300000	19.2	44.626667	19.79	...
3	50	40	19.89	46.066667	19.2	44.590000	19.79	...
4	60	40	19.89	46.333333	19.2	44.530000	19.79	...

Figure 14 Five first samples of the dataset

The first and second feature are the appliances and lights energy consumption in Wh. Then, features 3 to 20 show the temperature and humidity of all rooms of the building. Next, feature 21 to 26 are metrics given from Chievres weather station (air temperature, humidity etc.). Lastly, two random variables are induced to the dataset from the constructor. This work is focused to predict the appliances energy consumption in the short-term future combining knowledge from previous samples. In the table below various information about the five first features are depicted

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

	Appliances	lights	T1	RH_1	T2	...
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	...
mean	97.694958	3.801875	21.686571	40.259739	20.341219	...
std	102.524891	7.935988	1.606066	3.979299	2.192974	...
min	10.000000	0.000000	16.790000	27.023333	16.100000	...
25%	50.000000	0.000000	20.760000	37.333333	18.790000	...
50%	60.000000	0.000000	21.600000	39.656667	20.000000	...
75%	100.000000	0.000000	22.600000	43.066667	21.500000	...
max	1080.000000	70.000000	26.260000	63.360000	29.856667	...

Figure 15 Various feature information

3.3.2. *Problem formulation and preprocessing*

The most critical aspect of creating an appropriate and accurate DL solution is the problem formulation because different approaches lead to huge variations in metric accuracy. The main restriction at this stage is based on the hardware resources that could support such Neural Network models. This affects the sequence length (how many samples) are fed to the model. Since every sample is per 10 minutes, sequence length was chosen to be set at 12 (observing 2 hours before from the current time), so *SequenceLength* = 12. The value that must be predicted is in the appliances column after *targ* (target) timesteps. In order to predictthe values after 1 hour from the current timestep, *targ* was set to 6.

An additional important parameter is the sampling rate (or step) that must be chosen wisely. The dataset length (total timesteps or samples) could be characterized as small and there is no other choice from setting the sampling rate equal to 1. So, every timestep is being utilized to train, validate and test the Neural Networks models. The *targ* could set accordingly in the desirable time slot that the designers want to predict, i.e., *targ* = 12 for prediction after 2 hours, *targ* = 30 for prediction after 5 hours and so on. Likewise, the SequenceLength could be set accordingly, but through a large number of experiments, optimal value was found at 12.

The preprocessing state is quite crucial for the model accuracy. In this work none of the features were omitted. Thus, each feature was utilized in the experiments. A different approach could be to drop some of them like *rv1* and *rv2* in order to reduce the problem complexity, but it was found through testing that there is, at some point, a kind of correlation between them. Another decision that needs to be made is how to split the dataset in training set, validation set and testing set. The traditional approach suggests 60%, 20%, and 20% respectively for train, validation and test sets. It was found that a better option for this particular dataset is to be split as 55%, 25% and 20%. So far, the preprocessing parameters have been set as follows:

- *SequenceLength* = 12
- *Targ* = 6
- *SamplingRate* = 1
- *Num_training_set* = 10854
- *Num_validation_set = 4933*
- *Num_test_set = 3948*

The last option that must be set is if whether to adopt either the standardization or the normalization technique. Studying the international bibliography for analogous problems, the data

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

standardization is promoted as the proper one. For each feature of the dataset, the mean value is calculated based only on training set. Then, is subtracted from the total dataset. Thereafter, the std value of each feature from the new dataset is calculated based again only on training set and divided to the whole dataset. In that way, each feature gets a mean value around zero and a std close to one. These calculations are depicted below as python code where *float_data* is the NumPy array containing the original dataset. At this point, it is important to note that, the mean and the std values are calculated based only on the training data set. No data should be extracted from the validation or test dataset and induced to the model.

```
mean = float_data[:num_train_samples].mean(axis=0)
float_data -= mean
std = float_data[:num_train_samples].std(axis=0)
float_data /= std
```

Figure 16 Data standardization

3.3.3. *Designed and Developed Neural Networks models*

The Python neural networks models that were adopted for this work can be divided into two different categories. First, an artificial neural network (ANN) was employed in order to obtain some baseline metrics such as *Mean Absolute Error (MAE)*. Various forms and structures were tested thoroughly during the model tuning phase. The final one is constituted of 3 dense layers of 16 nodes each. The detailed architecture is shown in Figure 17 below

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

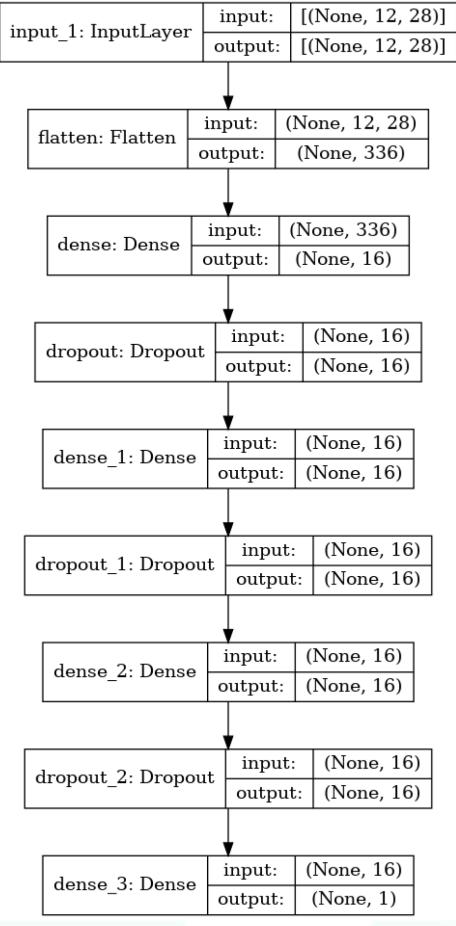

Figure 17 ANN structure

As it was analytically declared in the problem formulation, a matrix 12x28 is fed to the model at input layer. Then, a *Flatten* layer is applied to transform the input matrix to a vector of 336 elements. Next, a dense layer of 16 nodes is inserted sequentially. Thereafter, a dropout layer is applied in order to prevent overfitting. After various experiments, best values over multiple experiment (optimal) for the dropout layers found to be at 40% and the number of nodes were set to 16. A 40% dropout means that 30% of extracted weights will be set to 0. Also, adding more nodes has no significant impact to the model performance. The activation function that was employed is the ReLU one (Figure 18). ReLU sets to zero inputs less than zero and makes no modification to those greater than 0. This sequence of layers (dense layer of 16 nodes followed by a dropout layer of 30%) was repeated two more times until the model reaches a final single node with no activation function to get the predicted value.

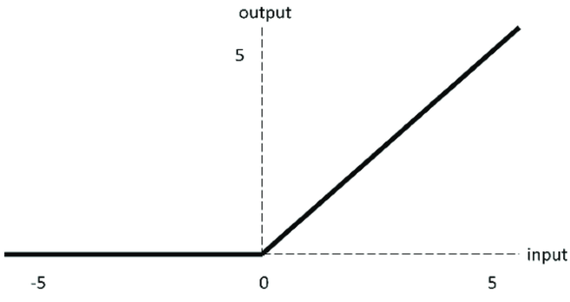

Figure 18 ReLU activation function

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

An important aspect of the design process is the utilization of a reasonable number of parameters to be implemented in an FPGA. As it is depicted on Figure 19, the total number of parameters is 5,953. The first dense layer is the most computationally intensive as it produces 5,392 parameters. Then, the second (*dense_1*) and third (*dense_2*) dense layers add 272 parameters each. Lastly, the last layer (*dense_3*, single layer node) adds only the final 17 parameters. The number of parameters of a layer is calculated by multiplying the current layer nodes with previous layer nodes plus one. Thus, **layer_*params = layer_nodes*(prev_layer_nodes + 1).***

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 12, 28)]	0
flatten (Flatten)	(None, 336)	0
dense (Dense)	(None, 16)	5392
dropout (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 16)	272
dropout_1 (Dropout)	(None, 16)	0
dense_2 (Dense)	(None, 16)	272
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 1)	17

Total params: 5,953
Trainable params: 5,953
Non-trainable params: 0

Figure 19 Number of parameters per layer

The work in [1] that is compared to the one presented in this deliverable, has found that MAE equals to 0.45 using GRU (Gated Recurrent Unit) units while the MAE equals to 0.40 based on LSTM (Long Short Term Memory) units. Neither of those units could be adopted for our work at the current phase due to designing tools restrictions of the HLS4ML toolbox. LSTM or GRU cells could detect in an enhanced way the temporal dependencies from a timeseries dataset. So, the purpose is to find solutions amenable to hardware implementation/acceleration such as ones based on ANN (artificial neural network) and CNN (convolution neural network) in order to achieve results as close as possible to LSTM/GRU ones.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

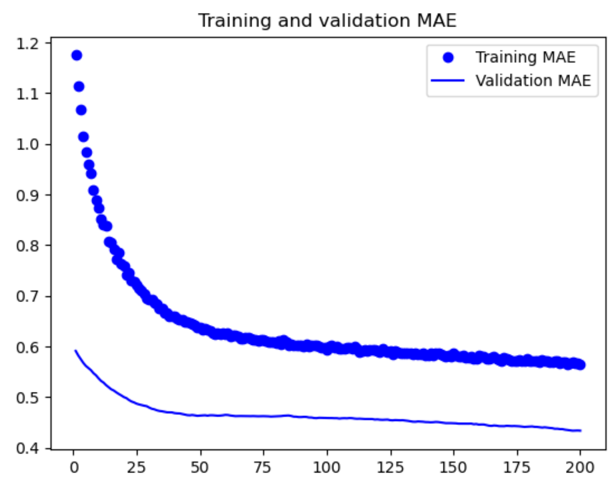

Figure 20 Training and Validation Process

The training and validation processes are illustrated in Figure 20 through 200 epochs. The test **MAE** was found to be **0.45**. This is a not a great result overall but can easily outperform the GRU solution due to its simpler architectural complexity. Compared to LSTM solution, our approach lags in accuracy but if a tradeoff needs to be made, this work preferable due to the fact that it is faster and consume far less resources compared to the LSTM design. Figure 21 shows the predicted values in comparison with the original ones for every sample of the test set.

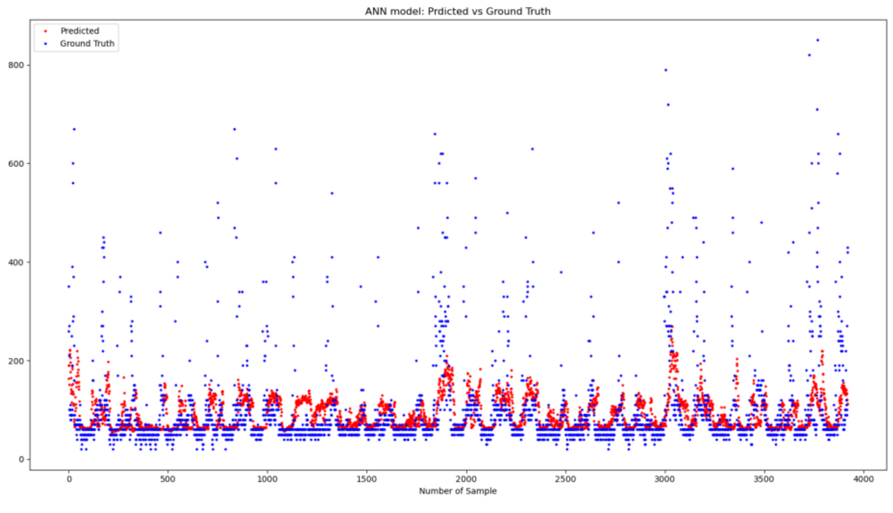

Figure 21 Predicted vs Ground truth samples of test set

The predicted values are illustrated with red dots while the ground truth data are depicted with blue dots respectively. It is obvious that the implemented ANN model is focused on predicting values around zero. Data points that diverge significantly from zero, are mainly responsible for the *MAE* final performance. Maybe a different normalization technique should be adopted, like min-max normalization but in a great number of prediction problems, the standard normalization, s preferred. Regarding the ANN model, it exhibits analogous performance with the GRU one while at the same time has quite simpler architecture.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

The second approach that was employed for this work in order to exploit all the feasible neural network solutions is based on a CNN (convolution neural network) architecture. This model is actually, a combination of a CNN and ANN in the sense that on top of a CNN an ANN is connected. The model's architecture is illustrated in detail in Figure 22. The input Is a 2d tensor (12x28, *SequnenceLength* x *number_of_features*) that is fed into a convolution layer of one dimension (Conv1D). This layer was set to produce 32 different kernels (*filters = 32*) and kernel size was tuned to 1 (*kernel_size = 3*). Various values were tested to tune the filter and kernel size for optimal results. The filter and kernel sizes that were utilized were [32, 64, 128] and [3, 5, 7, 9] respectively. There was no improvement though in the test MAE.

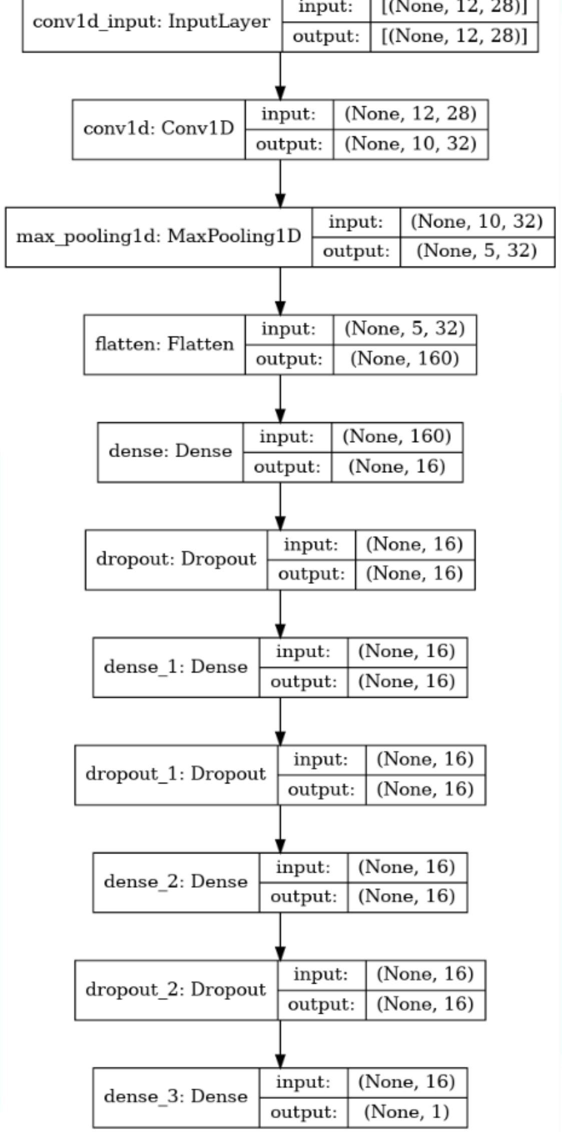

Figure 22 CNN structure

The performance that has been achieved for this architecture is 0.44 of test MAE. There was a slight enhancement compared to the ANN solution, but the CNN model is much more computationally intensive than the previous one. The shape of the Conv1D is (None, 10, 32). The first dimension (None) corresponds to the batch size and is configured to 64 (*batch_size = 64*) at runtime.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

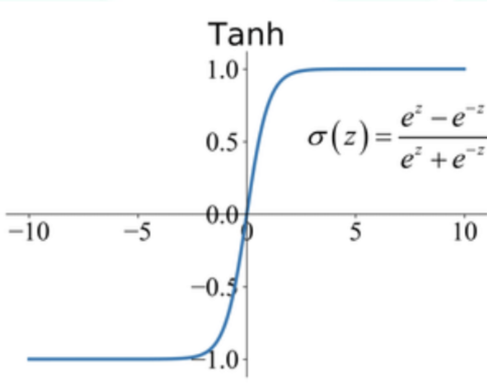

Figure 23 Tanh(x) activation function

The second dimension is 10 and is calculated as follows. Let *S* be the second dimension of the output convolution layer, then *S = SequenceLength – kerne_size + 1*, based on the fact that padding and stride parameters were set as 'valid' and 1 respectively. As it was previous declared the value of the third dimension is determined by the desired number of the filters layer. The convolution layer is followed by a single dimension *MaxPooling* Layer of size 2. The operation of this filter is quite simple. For each of the extracted filters, every two elements are compared and the bigger one is chosen. An example of this process is shown in Figure 24 as follows

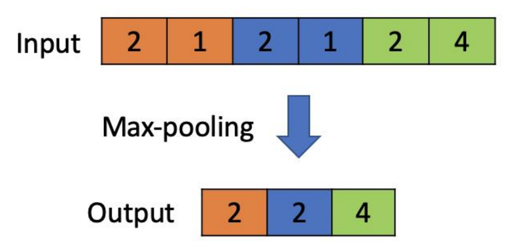

Figure 24 MaxPooling Layer operation

This structure of a convolution (Conv1D) layer followed by a pooling layer (MaxPoolingLayer1D) was utilized repeatedly 2 and 3 times (Conv1D + MaxPooling1D + Conv1 +…) but none of the repetitions worked effectively (based on the performance metric). It is important to note that the activation function of the Conv1D is *ReLU* Consequently a Flatten layer was used to convert the input 1D tensor (matrix) 5 x 32, into a 1D tensor (vector) of 160 elements. Then, the ANN architecture from the previous design was employed and placed on top of CNN model. The only difference of this ANN model compared to the previous one, is the utilization of the *tanh* activation function.

Since the remaining architecture of the CNN model is based on the previously presented ANN approach (3 dense layers of 16 nodes combined with dropout layers of 40%), no further explanation needs to be given. The sum of all the given parameters is equal to 5,847 total parameters. The number of parameters of each dense layer is calculated as, ***dense*_layer_*params* = *layer_nodes*(prev_layer_nodes + 1).*** On the other hand, the parameters of the convolution layer is computed as, ***conv_params = kernel_size * num_features * filters_size + filters_size***. Thus, the total number of parameters in the convolution layer are equal to 2,720 elements.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

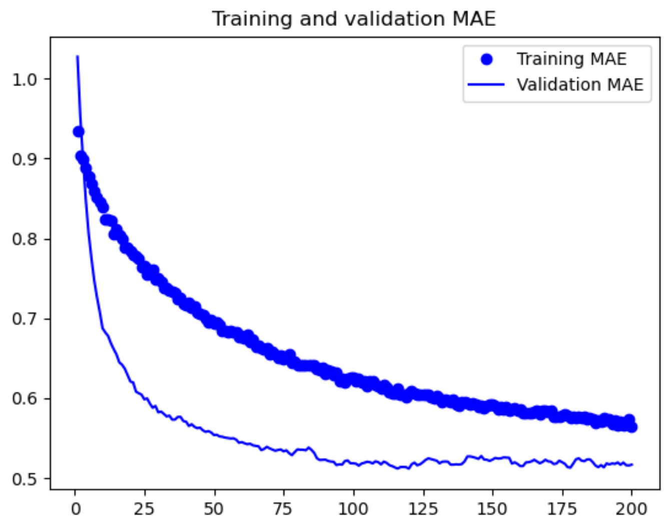

Figure 25 Training and Validation Process

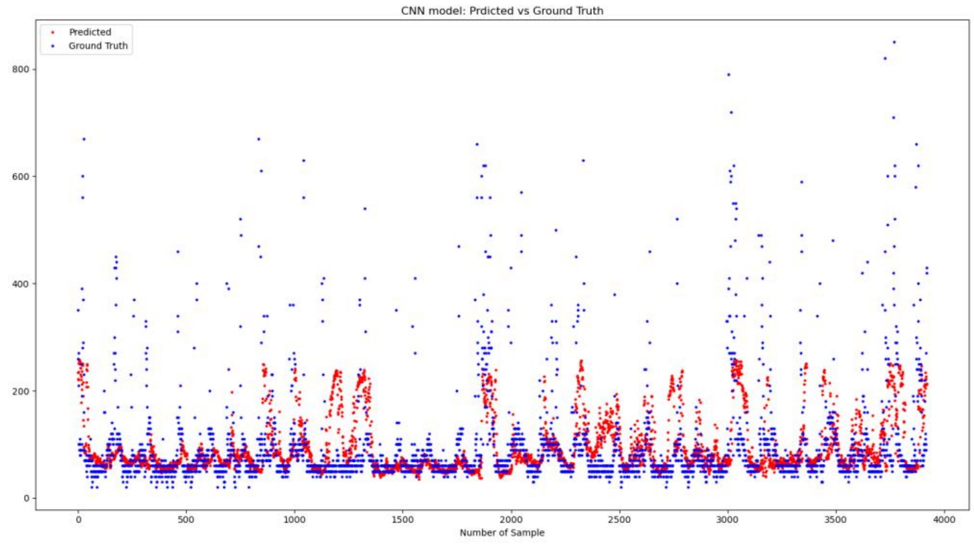

Figure 26 Predicted vs Ground truth samples of test set

The training and the validation process are illustrated in Figure 25 for each epoch, 0 until 200 of them. The test MAE (Mean Absolute Value) is equal 0,44. There is a minor improvement here in terms of performance, but the first approach (ANN) is selected due to its inherit simpler design complexity. As it could be seen, the validation metric converges in a more efficient way than the training one. Figure 26 depicts the predicted versus the ground truth samples for each of the training samples. Red dots (predicted values) vary from minus one to plus one mainly, which means that the model cannot predict values that diverge significantly from that region.

Finally, it is important to summarize all the parameters settings that were adopted to tune our models to get optimal results

- *Sequence Length = 12*
- *Target = 6*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

- *Sampling rate = 1*
- *Batch size = 64,*
- *Optimizer = Adam,*
- *learning rate = 10e-5,*
- *Loss function = MSE (Mean Square Error),*
- *Accuracy = MAE (Mean Absolute Error),*
- *Epochs = 200*

3.3.4. *Other approaches*

Energy consumption prediction is often focused on forecasting daily consumption profiles through regression and time series models. In the Industry 4.0 framework, the experimental measurements for the energy consumption are in fact characterized by high dimensional formats that can be well represented as functional data or profiles for each day [2]. To predict energy consumption profiles, the generalization of the multivariate regression analysis to case where the covariates and/or the response have a functional form, can be made (it is referred to as functional regression [3]). Functional egression models allow for the prediction of the energy consumption of the next day based on the information available up to the current instant and covariates. The main advantage of these models is their suitability for dealing with high-dimensional data observed on possibly uneven, non equidistant time points, with noise. The application of these models in the EnerMan project will be shown in D4.1, Section 5.3 (Integrated approach for system description and statistical-deterministic analysis).

3.4. Intelligence Acceleration using Hardware assistance

The main purpose to implement an algorithm in hardware is either to accelerate its performance or for portability reasons. A hardware implementation can be placed on edge and run it independently from other applications. It is important that there is no need to dedicate either a GPU or CPU for that application, although, the main reason is to achieve optimal performance in terms of speed. The process that was followed to obtain hardware implementations is based on an open-source framework called ***hls4ml.*** In this subsection we describe analytically the hls4ml framework, all of its capabilities and the whole workflow process that has been employed in order to obtain the RTL (register transfer level) design of the subsection 3.3 selected DL model.

3.4.1. *The open-source framework hls4ml*

hls4ml is a Python package for implementation of machine learning inference in FPGAs. It creates firmware implementations of machine learning algorithms using high level synthesis language (C++) with the backend based on Vivado-HLS. It translates traditional open-source machine learning package models into HLS compatible code format that can be configured accordingly, depending on the case. A list of supported ML codes and architectures, including a summary table is below. Dependences are given in the Setup page. hls4ml supports Keras, Tensorflow, QKeras, PyTorch and Onnx libraries. For this work, Keras package was employed. The neural network architectures that the framework supports are convolution based, either 1D or 2D, and fully connected networks (ANN, multi-layer perceptron).

The workflow procedure is straight forward and could be described as follows. First, a *yaml* file (.yml) needs to be created containing the configuration of all the necessary parameters. YAML file is a data serialization language that is used for writing configuration files for various programming languages. Figure 3.6.1 shows a basic example of *yml* configuration. The only files that need to be provided into *yml* file are, first the python trained model as a json file and second, the extracted weights as h5 file. *Keras* or *h5* are two different types of files that Keras package can store the model weights.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
KerasJson: keras/KERAS_layer.json
KerasH5:   keras/KERAS_layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xcku115-flvb2104-2-i
ClockPeriod: 5
Backend: Vivado

IOType: io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,6>
    ReuseFactor: 1
  LayerType:
    Dense:
      ReuseFactor: 2
      Strategy: Resource
```

Figure 27 yml file example

In the above figure, a basic example is illustrated. There are several configuration options that must be declared and are analytically described below:

- ***KerasJson/KerasH5***: for Keras, the model architecture and weights are stored in a json and h5 file. There is support for keras model's file obtained by *model.save()* Python command. In this case, the h5 file is supplied in KerasH5: field.
- ***InputData/OutputPredictions***: path to the input/predictions of the model. If none is supplied, then *hls4ml* will create artificial data for simulation. *Numpy* data files are supported.
- ***OutputDir***: the output directory where the HLS project appears
- ***ProjectName***: the name of the HLS project IP that is produced
- ***Device***: the FPGA part number targeted, here it's a Xilinx Virtex-7 FPGA
- ***ClockPeriod***: the clock period, in ns, at which the algorithm runs
- ***IOType***: options are *io_parallel* or *io_stream* defines if the algorithm uses pipeline technique or not
- ***ReuseFactor***: in the case that pipeline is used, this defines the pipeline interval or initiation interval
- ***Strategy***: Optimization strategy on FPGA, either "Latency" or "Resource". If none is supplied, then hl4ml uses "Latency" as default. Note that a reuse factor higher than 1 should be specified when using "resource" strategy.
- ***Precision***: this defines the fixed-point precision of the inputs, outputs, weights and biases. It is denoted by ap_fixed<X,Y>, where Y is the number of bits representing, the integer part, and X is the total number of bits. Additionally, integers in fixed precision data type (ap_int<N>, where N is a bit-size from 1 to 1024) can also be used.

At *HLSConfig* field, the general configuration of the model is described. But there is an option to further configure it in a more fine manner employing a per-layer configuration. As it is shown, the dense layers have a different setup when using resource *Strategy* where *ReuseFactor* equals 2, while the remaining model utilizes a pipeline approach.

The *yml* file that has been created for our ANN architecture described in section 3.6 is depicted in Figure 28. The clock period has been set to 5ns and *IOType* was selected for the *io_stream* (no pipeline) and the strategy was configured as *Resource* in order to minimize the resource utilization of

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

the FPGA device. A ReuseFactor equal to 4 was selected meaning that for every *X* concurrent operations, *x/4* modules (i.e. DSPs) are employed. Finally, the precision that was chose is equal to 14 bits total, with 6 MSBs representing the signed integer part.

```
KerasJson: model_ann_5.json
KerasH5: model_ann_5_weights.h5
OutputDir: my-hls-ann5
ProjectName: project_ann5
XilinxPart: xc7vx690tffg1927-2
ClockPeriod: 5ns

IOType: io_stream # options: io_stream/io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<14,6>
    ReuseFactor: 4
    Strategy: Resource  # options: Latency/Resource
```

Figure 28 ANN configuration yml file

Thereafter, using command line interface (*CLI*) and yml file, the HLS project is created. Then, the project needs to be built using the *hls4ml build* command. Although, there are many options which may be enabled during the *built* process, the *–all* one (means that all steps are to be done) was chosen for the ANN and CNN architecture. A detailed list of them that the designers can enable/disable are presented below

- -c, --csimulation: run C simulation.
- -s, --synthesis: run C/RTL synthesis
- -r, --co-simulation: run C/RTL co-simulation.
- -v, --validation: run C/RTL validation.
- -e, --export: export IP (implies -s)
- -l, --vivado_synthesis: run Vivado synthesis (implies -s).
- -a, --all: run C simulation, C/RTL synthesis, C/RTL co-simulation and Vivado synthesis.
- --reset: remove any previous builds

The ANN architecture design project has been successfully built. The desired clock period of 5ns have been achieved and RTL implementation can operate at 200MHz frequency. The Latency of the design is 104 cycles. Given that the design runs at 200MHz, this leads to an impressive time delay of only 520 nsec.

```
CO-SIMULATION RESULT:
Report time       : Sat Jun 18 16:20:34 EEST 2022.
Solution          : solution1.
Simulation tool   : xsim.

+----------+---------+-----------------------------------------+
|          |         |                 Latency                 |
+   RTL    + Status  +-----------------------------------------+
|          |         |     min     |     avg     |     max     |
+----------+---------+-----------------------------------------+
|      VHDL|       NA|           NA|           NA|           NA|
|   Verilog|     Pass|          104|          104|          104|
+----------+---------+-----------------------------------------+
```

Figure 29 Latency of ANN design

The resources that are utilized to implement the ANN design are shown in Figure 30 analytically. In this context, an explanation of the available resources of an FPGA is briefly given. Look-Up-Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

(LUTs), can perform arbitrary logic functions on small bit width of 2-6 inputs. These can be used for Boolean and arithmetic operations as well as for memory purposes.

```
================================================================
== Utilization Estimates
================================================================
* Summary:
```

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	34	-
FIFO	0	-	480	2496	-
Instance	573	1476	87033	53313	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	573	1476	87519	55879	0
Available	2940	3600	866400	433200	0
Utilization (%)	19	41	10	12	0

Figure 30 Resource utilization of ANN design

Flip-Flops registers on the other hand can store data in sync with the clock pulse. The most desirable FPGA components for Neural Networks architectures are Digital Signal Processors (DSPs). DSPs are specialized units for arithmetic operations like multiplications and additions. They are faster and more efficient than using LUTs for these types of operations. BRAMs are small, fast memories - RAMs, ROMs, FIFOs (18Kb each in Xilinx). Memories that are created using BRAMs are more efficient than using LUTs. An FPGA like Virtex-7 has nearly 100Mb of BRAMs.

```
KerasJson: model_CNN_5.json
KerasH5: model_CNN_5_weights.h5
OutputDir: my-hls-CNN5
ProjectName: project_CNN5
XilinxPart: xc7vx690tffg1927-2
ClockPeriod: 8ns

IOType: io_stream # options: io_stream/io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<14,6>
    ReuseFactor: 8
    Strategy: Resource  # options: Latency/Resource
```

Figure 31 Configuration yml file of CNN design

The implemented RTL design employs 573 BRAM of 18K, 1476 DSP units, 85,519 Flip Flops and 55,879 LUTs. In terms of FPGA available resources, the design occupies 19%, 41%, 10% and 12% of the aforementioned resource types respectively. It is obvious that the high utilization of DSPs units is the main reason of the low latency result.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
CO-SIMULATION RESULT:
Report time        : Sat Jun 18 18:07:42 EEST 2022.
Solution           : solution1.
Simulation tool    : xsim.

+----------+---------+------------------------------------------------+
|          |         |                    Latency                     |
+   RTL    +  Status +------------------------------------------------+
|          |         |      min       |      avg       |      max     |
+----------+---------+------------------------------------------------+
|      VHDL|       NA|              NA|              NA|            NA|
|   Verilog|     Pass|            4418|            4418|          4418|
+----------+---------+------------------------------------------------+
```

Figure 32 Latency of CNN design

The second architecture which was implemented in hardware was the CNN based one described in detail in section 3.3. As it is shown in the yml configuration file in Figure 31, it was necessary to alter two configuration option compared to the yml file of the ANN design (Figure 28). First, the clock period needs to be changed to 8 ns and the resource factor was set at 8. The resource factor implies the number of multiplications performed by each multiplier. Those changes were necessary in order to implement the RTL project successfully.

The CNN implementation constrained at 8ns, produce an RTL design of 125MHz frequency but the main issue of this hardware model is the achieved latency of 4,418 cycle (see Figure 32) which is far higher compared to the ANN implementation. In Figure 33, the resource utilization of the CNN implementation model is illustrated. This work makes use of 325 Block RAMs, 770 DSP units, 56,493 Flip Flops and 79,740 Look Up Tables. Based on the FPGA available resources these results mean that 11%, 21%, 6% and 18% of the above resources are utilized respectively.

In terms of performance, the ANN implementation outperforms the CNN one, because it operates at a higher frequency (60% faster) and needs far less cycles to output the result. Based on the resource utilization though, the CNN hardware outperforms the ANN implementation since it utilizes almost 42% less BRAMs, 49% less DSPs and 40% less Flip Flops and only in terms of LUTs, the CNN circuit uses 50% more.

Overall, the ANN hardware design is suggested since the main targets for that work were the frequency and latency metrics.

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+--------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |   LUT  | URAM|
+-----------------+---------+-------+--------+--------+-----+
|DSP              |        -|      -|       -|       -|    -|
|Expression       |        -|      -|       0|      34|    -|
|FIFO             |        0|      -|    1024|    5120|    -|
|Instance         |      325|    770|   55463|   74550|    -|
|Memory           |        -|      -|       -|       -|    -|
|Multiplexer      |        -|      -|       -|      36|    -|
|Register         |        -|      -|       6|       -|    -|
+-----------------+---------+-------+--------+--------+-----+
|Total            |      325|    770|   56493|   79740|    0|
+-----------------+---------+-------+--------+--------+-----+
|Available        |     2940|   3600|  866400|  433200|    0|
+-----------------+---------+-------+--------+--------+-----+
|Utilization (%)  |       11|     21|       6|      18|    0|
+-----------------+---------+-------+--------+--------+-----+
```

Figure 33 Resource Utilization of the CNN design

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

3.5. Federation based processing mechanism

The existing methods require large amounts of high-quality supervised data of the testing machine for training an effective diagnostic model. In real industrial scenarios, labelled condition monitoring data are usually difficult and expensive to collect. Different companies and factories have similar types of working machines, and they usually have their own supervised datasets for fault diagnosis. It is thus promising to integrate the data of similar devices across different parties for establishing a powerful fault diagnosis model.

We assume that multiple clients are included in the federated learning system, and each client has insufficient training data which fail to build its fault diagnosis model independently. In order for the federation scheme to succeed, it is imperative that the fault diagnosis tasks of all the clients are identical, which indicates that different clients share the same label space and that the same fault diagnosis model is shared by the server and different clients.

Thus, the federation scheme, as seen in Figure 34, is comprised of a federation server and multiple clients. The federation learning process, as seen in Figure 35, starts with the server creating the ML model architecture, such as the number of CNN layers, Fully Connected layers and their corresponding activation function amongst others. The server will instruct the clients with the model to work. The clients having a view only of their own dataset will generate a local limited model which in turn is sent to the federation server. The federation server will aggregate all the models into a new one and will send it to the clients to continue the learning process. This will continue until the global model has reached an acceptable accuracy level and will terminate the learning phase.

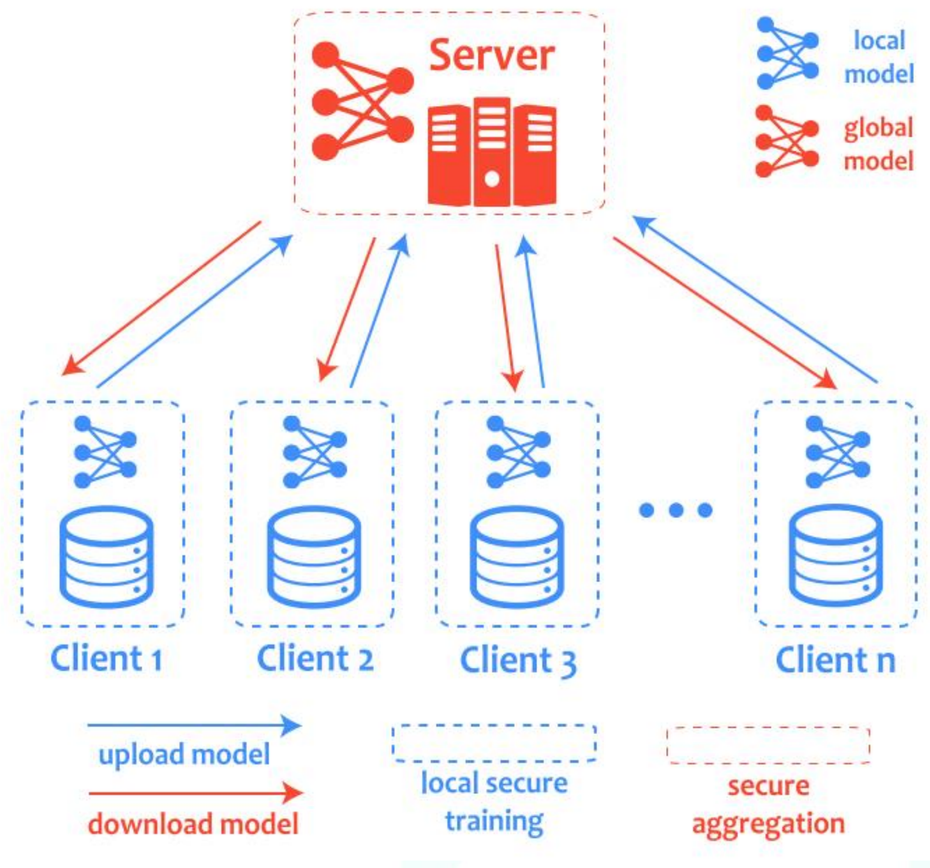

Figure 34 Federation scheme

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

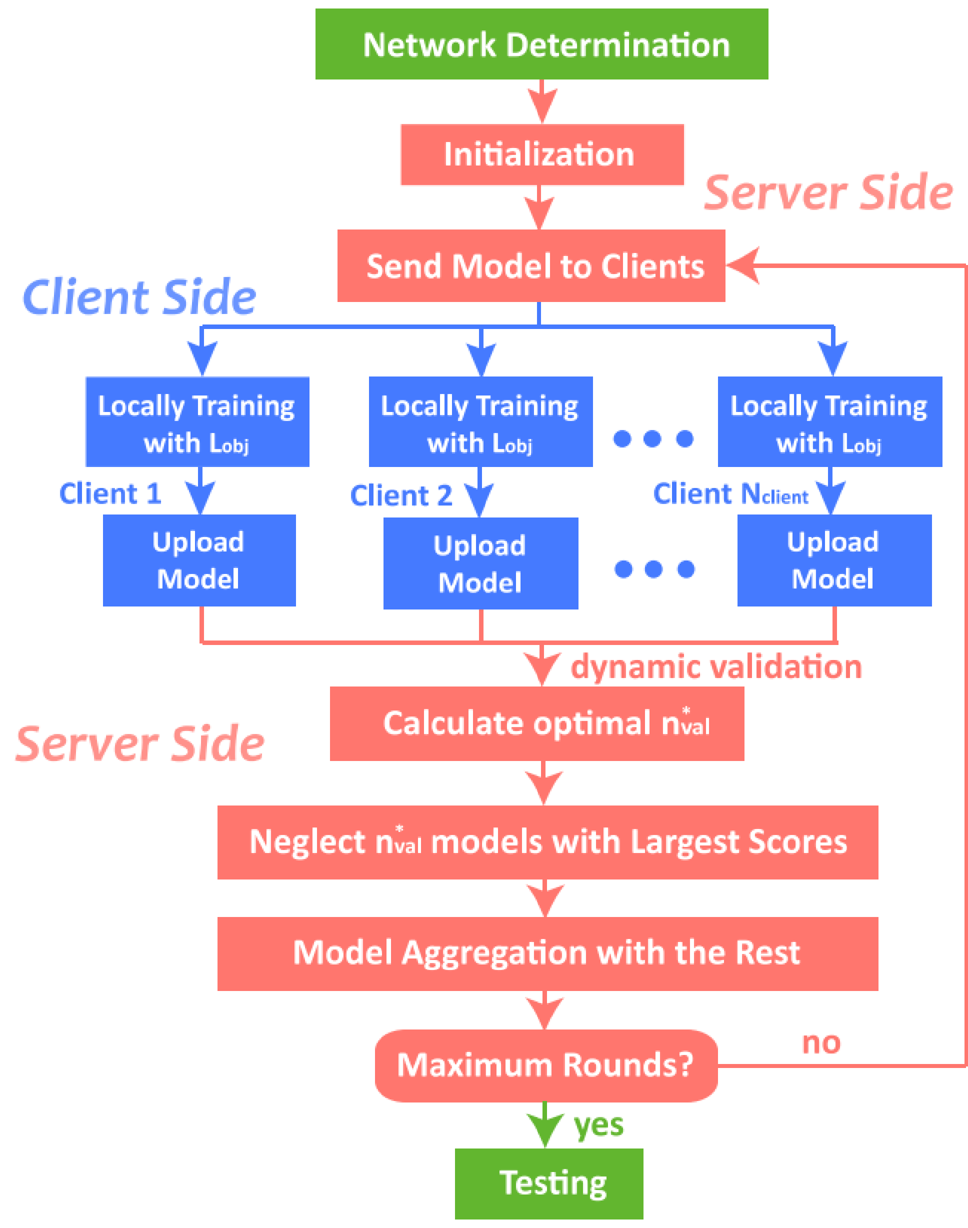

Figure 35 Federation learning process

To correctly transport all models to and from the server and clients, we designed a very simple application protocol, using TCP as transport protocol, as shown in Figure 36. Depending upon the Type, the messages have different connotations. For Type equal to:

0. The server is sending the initial model to the newly connected client.
1. The server is sending the updated global model to all connected clients.
2. The client is sending a learned model.
3. The server is notifying the clients that the learning phase is complete.
4. A client is requesting connection to the server and a Client ID.
5. The Server is sending the Client ID (Sender ID will contain the Client ID)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

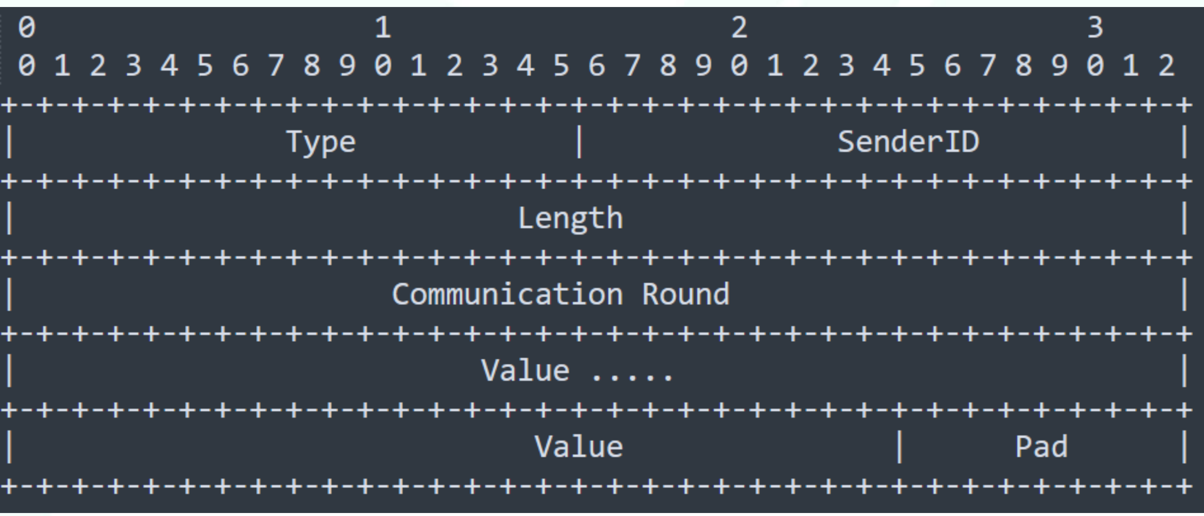

Figure 36 Simple protocol for sending ML models

The implementation of the complete process is shown in Figure 37. Initially (#1) the server and clients are instantiated. Upon instantiation of a new client, it will attempt to connect to the server (#2). Any newly connected client will be assigned a ClientID, which will be sent to the client via the SenderID field, in order for the server to keep track of which client has sent a model and to which it has already sent any response. The server having created the initial global model or having the evolved global model after a couple of learning phases have run, will send the current global model to any newly connected client (#3). The clients then train the next round of their local model (#4) and send the results to the server (#5). The server will aggregate all models and will calculate the next round of the global model (#6). Then the server will either send a termination message to the clients (#7) if the accuracy or rounds have reached a specific limit or will send the current global model (#7) and the process will continue from step 3.

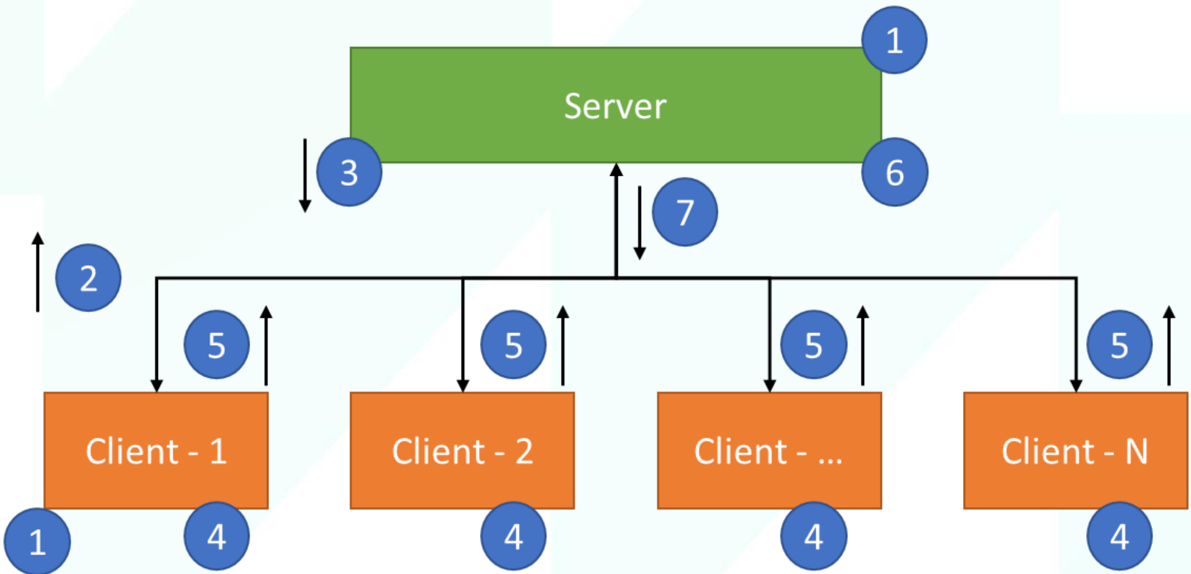

Figure 37 Implementation details of federation process

3.6. Edge node functionality and Configuration updates (reconfiguration)

One of the key characteristics in EnerMan is the ability to add new functionality in Edge nodes or reconfigure already installed ones. New edge node functionality can reside on a remote server and

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

would require a mechanism to install this functionality. This functionality could either be programs compiled for the specific edge node, e.g., the Ultra96v2, or hardware functionality for on board FPGAs built by tools such as Xilinx's Vitis. The latter can be defined as xclbin, IP cores bundled in a unified container that can hold outputs of hardware compilers (SDAccel xocc) as well as software compilers (processor ELF formats for MPSoc).

To download code and receive the output we leveraged Python's remote processing call, the rpyc module. Using this module, the client, which is manifested by the remote server hosting the program, can upload the necessary code to the rpyc server, which is manifested by the edge node waiting for the new functionality, and then start the program as if the client was local to the edge device.

Figure 38 depicts the initial setup, where the edge device's IP is an input argument, and the setup creates the rpyc connection. Figure 39 demonstrates the code for uploading software (vadd_hls) and hardware (vadd.xclbin) code to the edge node and then executing and receiving the result.

```
import rpyc
import sys
import os

if len(sys.argv) < 2:
    exit("Usage {} SERVER".format(sys.argv[0]))

server = sys.argv[1]

conn = rpyc.classic.connect(server)
rsys = conn.modules.sys
print(rsys.version)

ros = conn.modules.os
print(ros.uname())
```

Figure 38 Setup of the rpyc connection

```
print("Copying vadd_hls")
result = rpyc.utils.classic.upload(conn, l_cwd+'\\vadd_hls', r_cwd+'/vadd_hls')
print(result)
print("Copying vadd.xclbin")
result = rpyc.utils.classic.upload(conn, l_cwd+'\\vadd.xclbin', r_cwd+'/vadd.xclbin')
print(result)
print("Changing vadd to executable")
ros.system('chmod +x '+r_cwd+'/vadd_hls')
print(result)

print("Running code")
stream = ros.popen(r_cwd+'/vadd_hls '+r_cwd+'/vadd.xclbin')
output= stream.read()
print(output)
```

Figure 39 Download and run functionality

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

4. EDGE NODE SECURITY ASPECTS

As described in D2.1, traditionally, the Industrial Control Systems (ICSs) that are employed to control an industrial process, often referred to as Supervisory Control and Data Acquisition (SCADA) systems, are based on implementations lacking cyber-security considerations and practices. Reasons for this can be found in the lack of interoperability between different vendors and/or the adoption of proprietary protocols and data formats. Also, because, in general, these systems are "geographically isolated" with little to none connection to the outside networks. Ensuring interoperability between platforms and devices has two major challenges, i.e., their seamless operation and their security. The weakest link in this chain, from a cyber-security perspective, are the endpoints on SCADA systems, i.e., the Programmable Logic Controllers (PLCs) with their sensors and actuators. Not only is their firmware full of flaws with no regular update policy, but also, many of the most popularly used communication protocols lack authentication or encryption [4].

In legacy industrial deployments, the isolation of the SCADA deployments had been a viable option, however, in today's interconnected and technologically mobile world, true isolation is nearly impossible. It is, therefore, crucial that in EnerMan we tackle edge node security aspects efficiently since we are planning to collect data form the use case sites by interconnecting the edge nodes to the targeted ICSs.

4.1. Security Architecture

A high-level representation of the EnerMan security architecture is shown in Figure 40. Overall, its purpose is twofold. On the one hand, it is aimed at preventing malicious activities from becoming successful, i.e., it aims at reinforcing the flow of data such that they become immune to infection by malicious activity. The second involves the aspect of detection and, in this particular case, what we have is a mechanism for picking out unwanted activity that has managed to become part of the data flow.

The architecture is comprised of several levels, each of which correspond to a different part of a typical system that is going to use the EnerMan framework. Hence, there is the Industrial Data, which originate from the fringes of the architecture, e.g., sensor modules as edge devices in a factory, the Secure Gateway that is a little further up the hierarchy, i.e., the edge node of the system, and, finally, the cloud server.

At each of these points, as well as in-between, EnerMan is going to implement security features that will setup a strong security mechanism. Hence, starting from the edge devices, EnerMan utilises an intrusion detection mechanism named I2DS. This is implemented on MPSoC technology using the EnerMan edge/end node execution environment and is positioned right at the entry to the Data Aggregator, also co-hosted at the MPSoC. The I2DS operates on the data that are flowing in from the various edge devices used in the context of the various EnerMan use case providers' industry setups. Having filtered the data and flagged any potentially malicious activity, the data is then processed inside the MPSoC by the data aggregator and, subsequently, it is encrypted for cyber attack prevention purposes.

The encrypted data are going to fulfil TLS secure session communication protocol requirements, which will assist in the consolidation of a prevention mechanism between the MPSoC (edge devices) and Gateway (edge node) layers of the architecture. Just prior to the introduction of the MPSoC data into the gateway, a second Intrusion Detection System (IDS) mechanism is deployed. Hence, a detection mechanism just prior to the EnerMan gateway ensures that the encrypted data have indeed not been

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

corrupted. Subsequently, an Intrusion Prevention Systme (IPS) mechanism follows on the IDS-processed data at the gateway-level of the architecture.

Hence, and similar to the security steps followed at the EnerMan edge/end node MPSoC, the EnerMan gateway(s) will encrypt the data that are to be propagated further up the architecture, i.e., the EnerMan cloud devices, by ensuring that the TLS protocol standards are met for prevention purposes in the context of edge node and cloud communication.

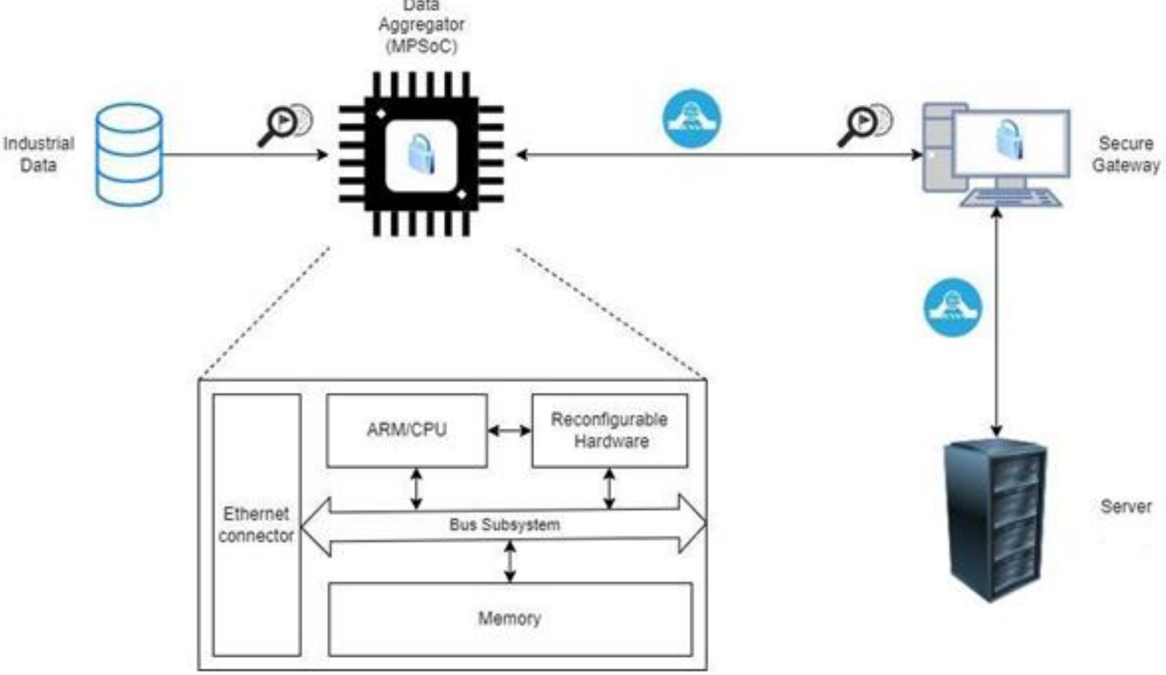

Figure 40 The EnerMan Security Architecture

4.2. Security Mechanisms

In this subsection we extend the work that has been presented in D2.1. Please refer to the D2.1 deliverable for some of the concepts described in section 4.2

4.2.1. *Cybersecurity Attack Detection*

I2DS: Industrial Intrusion Detection System

I2DS is the EnerMan intrusion detection mechanism that will be deployed across the system and at its very edges, i.e., it is the intrusion detection mechanism that will operate on the data coming in from the architecture's edge devices. I2DS is going to be hosted by the EnerMan edge/end node Multi-Processing System-on-Chip (MPSoC) devices, which employ Field-Programmable Gate Array (FPGA) technology. Hence, this first layer of intrusion detection capability is going to be implemented directly on (reconfigurable) hardware using the EnerMan edge/end-node execution environment described in Section 2.

Specifically, I2DS is comprised of optimised modules that implement machine learning models for intrusion detection. The modules use rules for string searching that are appropriate for the particular industrial environments' data. The implementation of the ML model's architecture is to be developed using suitable frameworks so that the final design not only fulfills the functional criteria of the ML

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

model, but also offers satisfactory performance, such as a high data throughput as well as reduced power consumption.

4.2.2. *Cybersecurity Attack Prevention*

Application Gateway

The WSO2 API Manager (APIM) is an API Gateway. Every call to APIs is rerouted to first pass through the WSO2 APIM which then forwards the request to the appropriate endpoint. This provides utilities such as the ability to allow/deny requests based on authorization status, enforce policies, modify the responses and requests' contents, and limit traffic, among others.

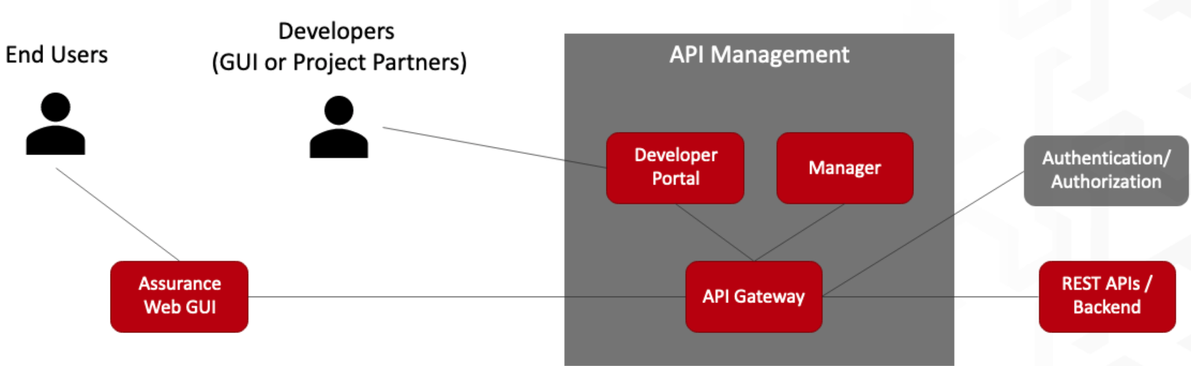

Figure 41 The Application Gateway overall architecture and the API Management.

Figure 41 shows the structure of the API Management component. Users can make requests to the API Gateway either through a frontend, or directly. The gateway stands between the user and the backend, handling the authentication/authorization aspects. Developers may also use the endpoints in their own applications. This is made easier through the developer portal, a "storefront" for APIs where their specification is published to help developers incorporate it into their own tools.

Messages that reach the API Gateway (Figure 42) are processed as follows:

1. When a request hits the API Gateway, it is received by the HTTP/HTTPS transport, that is responsible for carrying messages in a specific format. The transport provides a receiver and a sender (for receiving and sending messages accordingly).
2. The receiving transport selects a message builder, based on the message's content type, and uses the selected one to process the message's raw payload data and convert it into a common XML, which the Gateway mediation engine can then read and understand.
3. The request is passed through a set of handlers that applies the quality of services on the request message. Furthermore, it enforces security, rate-limiting, and transformations on API requests if applicable.
4. After all the requests are routed to the backend endpoint, a message formatter (selected based on the message's content type) is used to build the outgoing stream back into its original format depending on the message.
5. The transport sends the message out from the Gateway.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

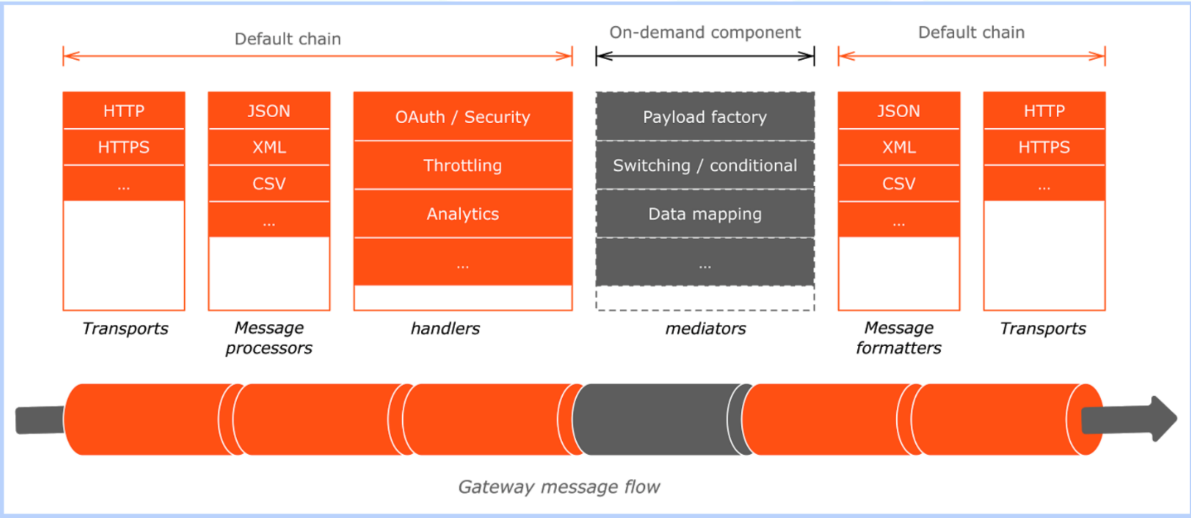

Figure 42 Gateway Message Flow

WSO2 APIM supports REST, SOAP, GraphQL, or Streaming APIs. For REST APIs, the WSO2 admin can either import already existing APIs in OpenAPI (formerly Swagger) format, or design and prototype a new API from scratch (eg. Figure 43).

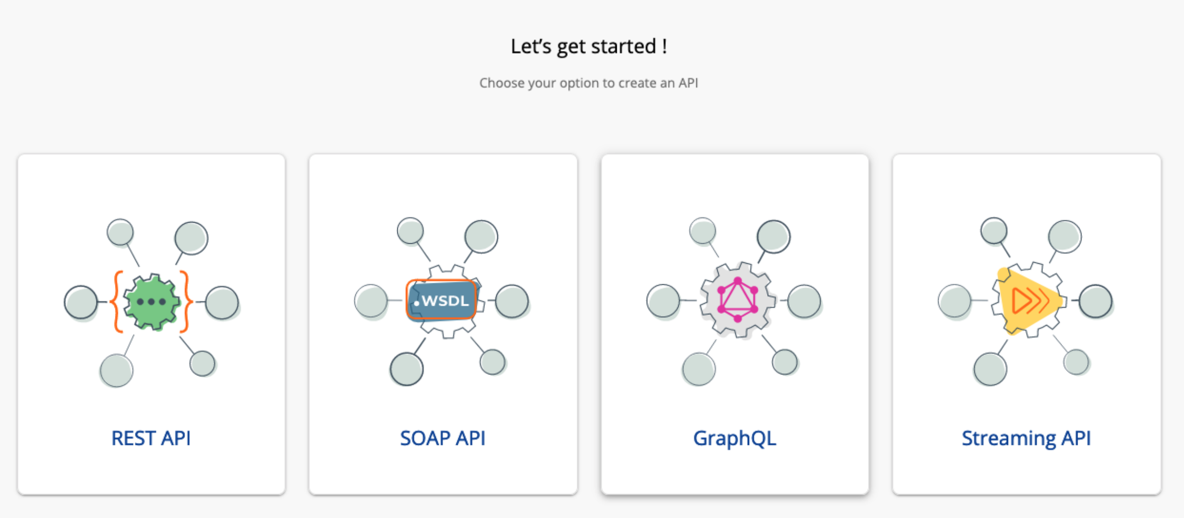

Figure 43 Welcome screen on WSO2 APIM, showing the admin all the available API types.

Following either import or creation, the admin can then alter the Runtime Configurations for this specific API (Figure 44). Options exist for enforcing security policies (HTTP, HTTPS, or both), the type of security (OAuth, Basic, API Key), Cross-Origin Resource Sharing (CORS) configuration, request and response validation against the API definition, and Message Mediation for acting on the content of requests and responses.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Runtime Configurations

Request

Transport Level Security

Application Level Security

CORS Configuration

Schema Validation

Message Mediation none

Response

Message Mediation none

Response Caching

Figure 44 Runtime Configurations for an API

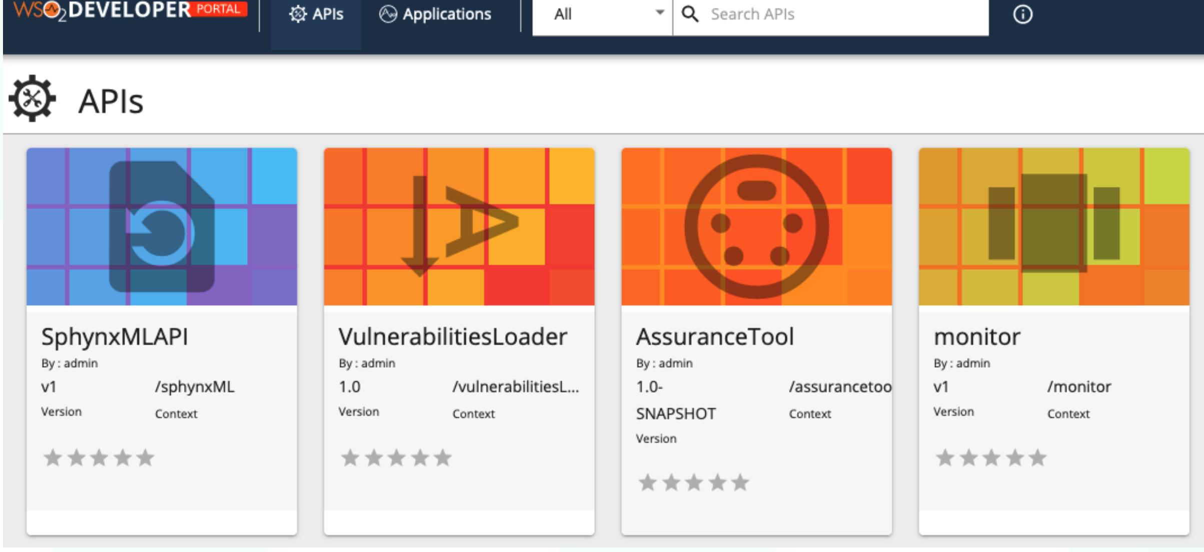

Figure 45 APIs appearing on the developer portal

APIs created by or imported to the WSO2 APIM, also appear on the WSO2 developer portal (Figure 45). There, developers can search APIs, view their documentation, try them out through a swagger-like interface, and also view available SDKs (Figure 46).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

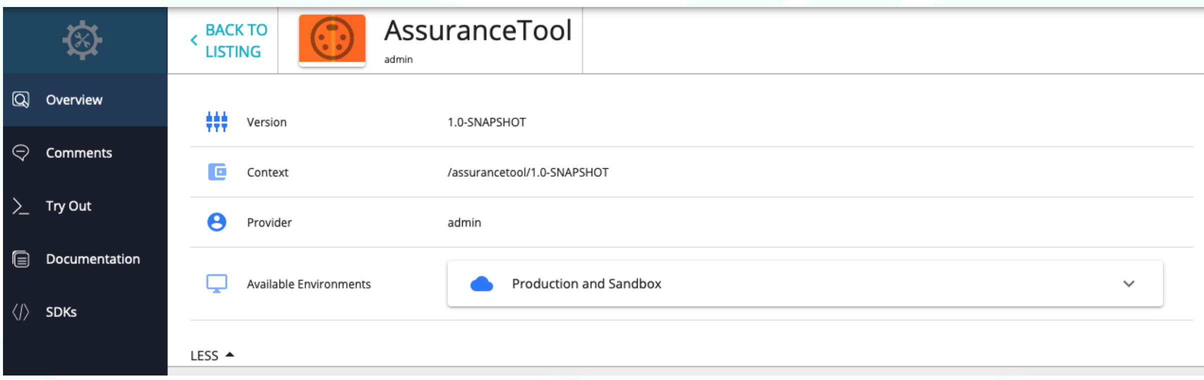

Figure 46 Available options for an API published on the devportal.

Main Features

The Gateway supports the following features to control access and enforce security.

- Supports JWT, OAuth2.0, Basic Auth, API Key, Mutual TLS, and more.
- Supports in-memory subscription validation, that decouples runtime dependency on the Key Manager.
- Provides multiple Key Manager support for authentication.
- Restricts API access tokens to domains/IPs.
- Validates APIs payload content against schemas.
- Applies additional security policies to APIs (authentication and authorization).
- Supports all standard OAuth2.0 grant types and allows extensions and additions to grants.
- Works seamlessly with third-party OAuth2.0 providers, standard, or proprietary.
- Allows blocking subscriptions due to non-payment, API abuse, etc.
- Associates API to system-defined service tiers for quotas and rate-limits.
- Generates JSON web tokens for consumption by back-end servers.
- Leverages XACML for entitlements management and fine-grain authorization.
- Provides threat protection, bot detection, and token-fraud detection.
- Supports detection of abnormal system use through artificial intelligence and machine learning.

Identity Server

In addition to the Application Manager, WSO2 provides identity and access management through its Identity Server (IS). The IS can be used directly by administrators (or other users if proper authorization was provided), through its Management Console. Apart from the registered users, IS can be used as an identity provider for third party systems that have their own set of users.

There are 3 main services the IS offers:

- Authentication, the process of identifying an individual. Authentication merely ensures that the individual is who he or she claims to be.
- Identity Administration, the process of creating new and modifying or deleting existing identities as well as managing the security entitlements associated with those identities.
- Security, the process to provide secure access to resources based on standards and model for security

All these functionalities are centralized to support the coordinated usage of the components and allow their interoperability. Once the users provide their credentials, they are

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

automatically authenticated on all applications enabling a Single Sign On (SSO) scenario between multiple heterogeneous authentication protocols.

The IS needs to be configured and customized to integrate the different components, applications and services. The various components need to be registered in the IS as service provider and linked to the involved authentication protocol. The Open ID Connect and the SAML 2.0 authentication protocols are involved in the integration of the current components.

The following figure (Figure 47) highlights one of the configuration steps needed to register one of the components on the IS.

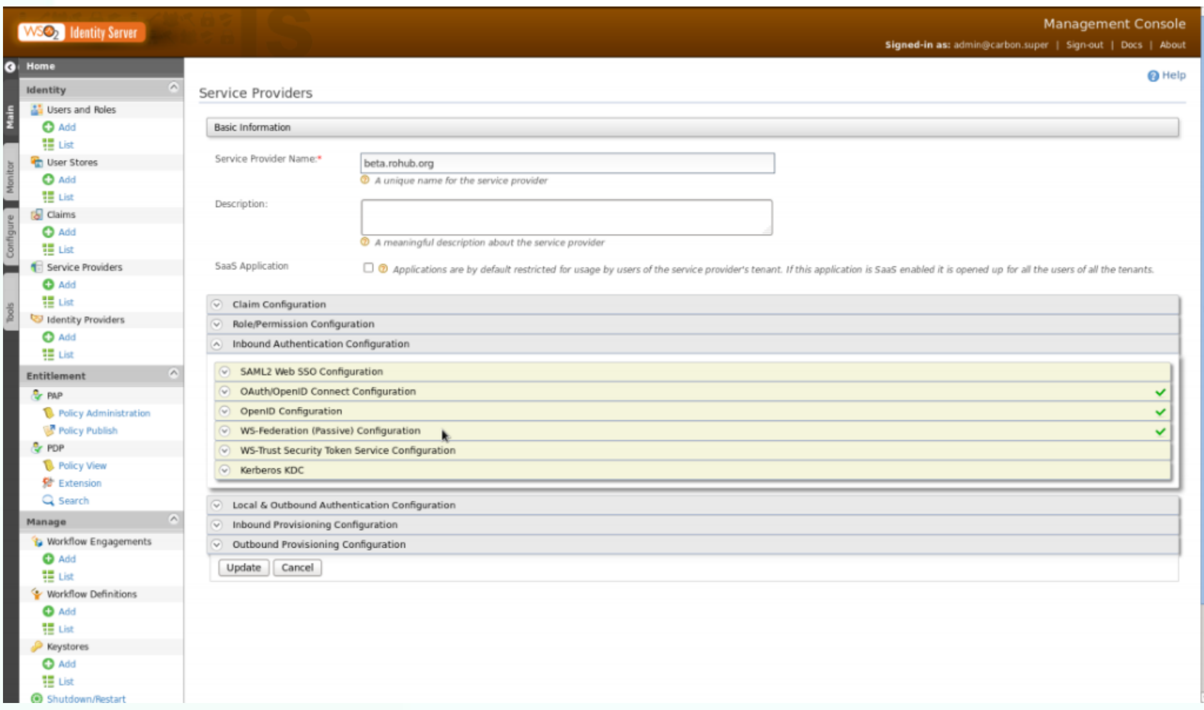

Figure 47 Configuration of a service provider with the different possible authentication protocols

To start using the services offered by the IS the generic application needs to be registered in the IS. The registration process of an Open Id Connect (OIDC) relying party (RP) has the following steps:

- Registration of the RPs in the IS (redirect URL and Relying Party name needed).
- Communication of ClientId and secret values to the Relying Party.

The registration process of a Service provider using the SAML 2.0 authentication protocol has the following steps:

- Registration of the SP in the IS.
- IS/SP Metadata exchanging.

The logic

This API provides a secure method to protect resources. When an API consumer want to access these resources the API owner, following the OAuth2.0 specification and using the exposed API can validate the access to the resources.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Following the OAuth2.0 specification, the API owner (Resource owner) is responsible for preventing unauthorized users from accessing an API (or a Resource in general). For that to be possible the specification prescribes that API consumers send a valid OAuth2.0 access token along with each API call. Such access token must be sent as the HTTPS request parameter. Once the API Server gets the call it needs to validate the access token against the OAuth2.0 Authorization server, before processing the request and replying to the consumer API. The following figure (Figure 48) provides an overview of the process, it contains the authentication process, steps 1-8, and the protected API call, steps 9-12.

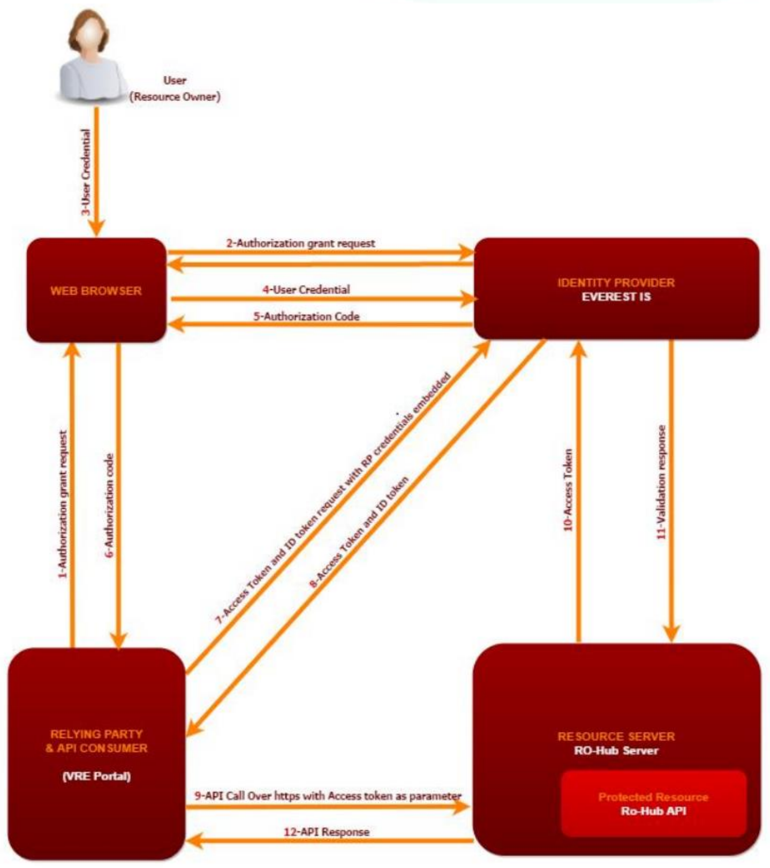

Figure 48 Authentication Process and API protection

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

5. DEMONSTRATION REPORT

5.1. Federated Learning edge node processing mechanism

As discussed in Section 3.5 and seen in Figure 37, the Federated Learning edge node processing mechanism is comprised of a federation server and multiple clients. In this demonstration report, we will showcase two clients connecting to a federation server and will attempt to reach an accuracy threshold of 70% within at most 5 rounds (we consider this accuracy level sufficient for the given problem). In effect, for the federation server to signal the end of the learning phase, either the global model will exceed an accuracy level on the validation data of 70% or the federation process, i.e. clients sending their model to the server and the server handing out a new global model, will stop after five rounds.

For testing purposes, the dataset is one big spreadsheet file that is appropriately split into regions. Each client is randomly sampling different part of that excel file and in effect each client sees a complete different set of data.

Figure 49 shows the initial setup of the server that calls the creation of the global model, as seen in Figure 50 which initiates an instance of the CNN class defined in Figure 51. Our CNN model is comprised of two sequential layers of 1D convolution with a batch normalization through a leaky ReLU activation function with a dropout probability of 50%, followed by two linear transformations. Once these have been setup, the server is ready to accept connections from clients.

```
def main():
    #Clients Directory
    clients={}
    serverPort = 5000
    print("Server starting.")
    conn = createServerPort(serverPort)

    ### read Data ###
    data_path = os. getcwd()+"//data.csv" #set the path for the data
    #create a global model and the test dataset for evaluating its accuracy
    global_model, test_loader = create_global_model_and_test_loader(data_path)

    T_acc_thres = 0.7 # Accuarcy Threshold
    round_threshold = 10 # Round Threshold
    communicationRound=5
    finished = False
```

Figure 49 Initial setup of the server

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```python
def create_global_model_and_test_loader(data_path):
    #read the data from the csv
    df_all = pd.read_csv(data_path)
    #extract the features and labels from the datag
    features = df_all.columns[2:]
    target = 'label'
    X_features = df_all[features]
    Y_target =  df_all[target]
    #Split the data into train and test set
    xtrain, X_test, ytrain, y_test = train_test_split(X_features, Y_target,
                                        test_size=0.1, shuffle=True)

    #create the test dataloader for the evaluation of the global model
    test_ds = TensorDataset(torch.tensor(X_test.values, dtype=torch.float32),
                                torch.tensor(y_test.values, dtype=torch.long))
    test_loader = DataLoader(test_ds, 2*16, shuffle=False)

    #Create the global model
    global_model = CNN_model(len(features)).to(device)

    return global_model, test_loader
```

Figure 50 Creation of the global model and test loader

```python
class CNN_model(nn.Module):
    def __init__(self, n_in):
        super().__init__()
        self.n_in = n_in

        self.layer1 = nn.Sequential(
            nn.Conv1d(1, 10, (9,), stride=1, padding=4),
            nn.BatchNorm1d(10),
            nn.LeakyReLU(),
            nn.Dropout(p=0.5),
        )
        self.layer2 = nn.Sequential(
            nn.Conv1d(10, 10, (9,), stride=1, padding=4),
            nn.BatchNorm1d(10),
            nn.LeakyReLU(),
            nn.Dropout(p=0.5),
        )

        self.linear1 = nn.Linear(5000, 64)
        self.linear2 = nn.Linear(64, 12)

    def forward(self, x):
        x = x.view(-1, 1, self.n_in)
        x = self.layer1(x)
        x = self.layer2(x)
        x = torch.flatten(x, 1)
        x = F.leaky_relu(self.linear1(x))
        x = self.linear2(x)
        return x
```

Figure 51 Class model of our CNN

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Figure 52 showcases the high-level overview of the process. While the server has not finished, it will initially wait for new clients to connect or will wait for new models. While the server is awaiting for clients to complete their learning phase, as seen in Figure 53, it can also accept new incoming clients cand send them the current global model. Once all models have been received it will aggregate the clients' models into the new global model and check the accuracy. This process will continue as long as the accuracy or the rounds threshold has not been reached.

```
while not finished:
    #Wait model from all clients
    waitForModels(conn, communicationRound, clients, global_model)

    #Collect the models from the clients
    clients_models = []
    for i in clients:
        clients_models.append(clients[i]['model'])

    #Aggregate the clients models and update the global model
    server_aggregate(global_model, clients_models)

    #Validate global common model
    accuracy = test(global_model, test_loader, CrossEntropyLoss())
    print("The accuracy of global model is:" , accuracy)

    communicationRound=communicationRound+1
    if accuracy > T_acc_thres or communicationRound>round_threshold:
        if(accuracy>T_acc_thres):
            print("Accuracy exceeded threshold:"+str(T_acc_thres))
        else:
            print("Communcation rounds exceeded threshold:"+str(round_threshold))
        finished = True

    #Else continue sending models to clients
    if not finished:
        #send model to clients
        sendModel(clients, communicationRound, global_model)

sendTerminationSignal(clients)
```

Figure 52 High level overview of server process

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
def waitForModels(conn, round, clients, global_model):
    print("Waiting for new clients or updated models")
    incomingSocks = [conn]
    outgoingSocks = []
    for i in clients:
        #clients[i]['socket'].setblocking(0)
        incomingSocks.append(clients[i]['socket'])

    while not allModelsReceived(clients) or not clients:
        reable, writable, exceptional = select.select(
            incomingSocks, outgoingSocks, incomingSocks)
        for s in reable:
            if s==conn:
                try:
                    newconn = handleNewClient(s, clients, round, global_model)
                    if(newconn!=NULL):
                        incomingSocks.append(newconn)
                except:
                    print("Connection Failed")
            else:
                try:
                    model, round, senderID, ret = receiveClientModel(s)
                except:
                    clID = findClientBySocket(clients, s)
                    incomingSocks.remove(s)
                    if clID > 0:
                        clients.pop(clID)
                    ret = -2
                if ret==-1:
                    print("Error in receiving model")
                elif ret == -2:
                    print("Client was disconnected - Removing from client list")
                else:
                    clients[senderID]['model']=model
                    clients[senderID]['done']=True
    print("All models received")
```

Figure 53 Server handling clients

Figure 54 and Figure 55 showcase the server lifecycle. Initially the server starts and await connections at the TCP port 5000. Two clients connect and are assigned specific IDs, 1 and 2. The clients (1 and 2) once connected and identified receive their models and start the training process. Once they are done, they send the model to the server, which aggregates and tests the new global model. If the accuracy level is low, the model is sent back to the clients and the process continues. At some point a new client join, is connected and assigned number 3. In this demo, as a case study, the accuracy threshold of 70% is not reached and after 5 iterations the learning process ends.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
(AI) C:\Workspace\Athena Research Center\Apostolos Fournaris - Enerman\Code\Anomaly Detection\Phase 2 - Complete>python server.py
Server starting.
Server port is 5000
The server is ready to receive at port 5000
Waiting for new clients or updated models
New client connected
Client ID 0 connected. Requests ClientID
New Client's ID is 1
New client connected
Client ID 0 connected. Requests ClientID
New Client's ID is 2
Received Message from Client 1
Client Sending Learned Model
Receiving model
Model received. Unpacking
Model written to buffer
Model Transfered ok.
Received model from client: 1
Received Message from Client 2
Client Sending Learned Model
Receiving model
Model received. Unpacking
Model written to buffer
Model Transfered ok.
Received model from client: 2
All models received
The accuracy of global model is: 0.4535853251806559
Sending model to client 1 with port 63327 and IP 127.0.0.1
Sending model to client 2 with port 63329 and IP 127.0.0.1
Waiting for new clients or updated models
Received Message from Client 2
Client Sending Learned Model
Receiving model
Model received. Unpacking
Model written to buffer
Model Transfered ok.
Received model from client: 2
Received Message from Client 1
Client Sending Learned Model
Receiving model
Model received. Unpacking
Model written to buffer
Model Transfered ok.
Received model from client: 1
All models received
The accuracy of global model is: 0.47915508615897723
Sending model to client 1 with port 63327 and IP 127.0.0.1
Sending model to client 2 with port 63329 and IP 127.0.0.1
Waiting for new clients or updated models
New client connected
Client ID 0 connected. Requests ClientID
New Client's ID is 3
Received Message from Client 1
```

Figure 54 Server initialization and connection from clients

```
Client Sending Learned Model
Receiving model
Model received. Unpacking
Model written to buffer
Model Transfered ok.
Received model from client: 2
All models received
The accuracy of global model is: 0.6325736520289049
Communcation rounds exceeded threshold:5
Sending termination signal to client 1 with port 63327 and IP 127.0.0.1
Sending termination signal to client 2 with port 63329 and IP 127.0.0.1
Sending termination signal to client 3 with port 63349 and IP 127.0.0.1
```

Figure 55 Server termination

Figure 56 and Figure 57 demonstrate the connections of Client1 and Client2 to the server, getting the initial model, starting the training process, and ending after five rounds have passed.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
(AI) C:\Workspace\Athena Research Center\Apostolos Fournaris - Enerman\Code\Anomaly Detection\Phase 2 - Complete>python client.py
Client 0 starting.
Client 0 server IP: 127.0.0.1 server port is 5000
Creating connection to Server
5000
127.0.0.1
Attempting to connect
Connection to Server made
Sending initial info to Server
My ID=1
Client1: Retrived client ID
Client1: Waiting for Initial Model from Server
Received Message from Server
Initial model incoming
Receiving model
Model Transfered ok.
Client1: Model retrieved from Server. Round:1
Client1: Training model for next round:2
Client1: Model trained. Round now:2
Client1: Sending model to Server
Client1: Model sent to Server
Client1: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client1: Global model received. Round now:2
Client1: Training model for next round:3
Client1: Model trained. Round now:3
Client1: Sending model to Server
Client1: Model sent to Server
Client1: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client1: Global model received. Round now:3
Client1: Training model for next round:4
Client1: Model trained. Round now:4
Client1: Sending model to Server
Client1: Model sent to Server
Client1: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client1: Global model received. Round now:4
Client1: Training model for next round:5
Client1: Model trained. Round now:5
Client1: Sending model to Server
Client1: Model sent to Server
Client1: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client1: Global model received. Round now:5
Client1: Training model for next round:6
Client1: Model trained. Round now:6
Client1: Sending model to Server
Client1: Model sent to Server
Client1: Waiting for Model or Termination from Server
Received Message from Server
Server finished training
Client1: Global model received. Round now:0
Client1: Gracefully Termination
```

Figure 56 Client1 learning process

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
(AI) C:\Workspace\Athena Research Center\Apostolos Fournaris - Enerman\Code\Anomaly Detection\Phase 2 - Complete>python client.py
Client 0 starting.
Client 0 server IP: 127.0.0.1 server port is 5000
Creating connection to Server
5000
127.0.0.1
Attempting to connect
Connection to Server made
Sending initial info to Server
My ID=2
Client2: Retrived client ID
Client2: Waiting for Initial Model from Server
Received Message from Server
Initial model incoming
Receiving model
Model Transfered ok.
Client2: Model retrieved from Server. Round:1
Client2: Training model for next round:2
Client2: Model trained. Round now:2
Client2: Sending model to Server
Client2: Model sent to Server
Client2: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client2: Global model received. Round now:2
Client2: Training model for next round:3
Client2: Model trained. Round now:3
Client2: Sending model to Server
Client2: Model sent to Server
Client2: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client2: Global model received. Round now:3
Client2: Training model for next round:4
Client2: Model trained. Round now:4
Client2: Sending model to Server
Client2: Model sent to Server
Client2: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client2: Global model received. Round now:4
Client2: Training model for next round:5
Client2: Model trained. Round now:5
Client2: Sending model to Server
Client2: Model sent to Server
Client2: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client2: Global model received. Round now:5
Client2: Training model for next round:6
Client2: Model trained. Round now:6
Client2: Sending model to Server
Client2: Model sent to Server
Client2: Waiting for Model or Termination from Server
Received Message from Server
Server finished training
Client2: Global model received. Round now:0
Client2: Gracefully Termination
```

Figure 57 Client2 learning process

Figure 58 showcases the connection of Client3 after a couple of rounds have passed. Client3 does not start from the beginning, rather catches up with the Client1 and Client2 and continues the training process from the current global model that the federation server has. Client3 will terminate at the same time as Client1 and Client2 as the federation server has general knowledge of the current round and when the learning process has to end.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
(AI) C:\Workspace\Athena Research Center\Apostolos Fournaris - Enerman\Code\Anomaly Detection\Phase 2 - Complete>python client.py
Client 0 starting.
Client 0 server IP: 127.0.0.1 server port is 5000
Creating connection to Server
5000
127.0.0.1
Attempting to connect
Connection to Server made
Sending initial info to Server
My ID=3
Client3: Retrived client ID
Client3: Waiting for Initial Model from Server
Received Message from Server
Initial model incoming
Receiving model
Model Transfered ok.
Client3: Model retrieved from Server. Round:3
Client3: Training model for next round:4
Client3: Model trained. Round now:4
Client3: Sending model to Server
Client3: Model sent to Server
Client3: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client3: Global model received. Round now:4
Client3: Training model for next round:5
Client3: Model trained. Round now:5
Client3: Sending model to Server
Client3: Model sent to Server
Client3: Waiting for Model or Termination from Server
Received Message from Server
Updated global model incoming
Receiving model
Model Transfered ok.
Client3: Global model received. Round now:5
Client3: Training model for next round:6
Client3: Model trained. Round now:6
Client3: Sending model to Server
Client3: Model sent to Server
Client3: Waiting for Model or Termination from Server
Received Message from Server
Server finished training
Client3: Global model received. Round now:0
Client3: Gracefully Termination
```

Figure 58 Client3 learning process

5.2. Hardware Security Token (HST) capabilities and Secure communication using Quantum-Safe TLS

5.2.1. *HST concept and overall usage*

The HST has been briefly described in D2.1 regarding its overall concept and capabilities. In this subsection we provide a more practical insight of its usage and how it is extended to support quantum safe security.

The HST is configured as a hardware/software codesigned application on the EnerMan intelligent edge node which is based on ARM microprocessors (e.g., ARM Cortex A family) and the EnerMan execution environment (an appropriate Linux distribution). The HST communicates with the underlying Linux OS through an external communication API. This interface is responsible for setting up the correct usage configuration of the HST during runtime through Command Line Input (CLI) commands issued for execution.

A software-based Crypto-Library is the core of the HST functionality, handling the high-level processes required for the implementation of most of cryptographic protocols and functions available by the HST. These functions include, but are not limited, to the generation and validation of digital signatures (ECDSA and postquantum cryptography algorithms) and certificates (X.509), key agreement schemes such as Elliptic Curve Diffie Hellman Ephemeral (ECDHE), Key encapsulation Mechanisms, encryption/decryption based on AES (AES-CBC, AES-CCM, authenticated AES-GCM) and authenticated message integrity schemes (HMAC). Depending on the availability of a reconfigurable FPGA Fabric by the device in use by the HST, the various cryptographic operations are hardware accelerated by

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

dedicated IP cores. These cores accelerate asymmetric cryptography primitives and can be either fully customized hardware designs developed with the help of a Hardware Description Language (VHDL, Verilog) or accelerated algorithms with the use of High-Level Synthesis (HLS) tools. At runtime, the included driver for the cores in the software stack handles the propagation of the necessary data to an accelerator and polls for a ready signal to read back the correct output and complete the cryptographic operation. Adopting an HLS approach, where operations are hardware accelerated using built-in processes, can lead to a much faster development time compared to a fully custom design without sacrificing a lot of performance speedup.

HST General Commands

The basic usage of the HST can be summarized by the following command structure:

./hst [options] [parameters]

By executing the above command, the HST polls the Linux CLI for the following argument operators (options) coupled with their corresponding parameters.

Options:
- -h Display usage syntax
- -i Select input file
- -o Select output file
- -l Set syslog remote server IP
- -p Set syslog remote server port
- -c Parse cryptographic command

Analyzing the above arguments into more detail, it is observed that with the[-i] operator, the HST loads an input file to be utilized for the execution of the various cryptographic protocols. Likewise, with the [-o] operator, a file is generated containing the output of a cryptographic command. In both cases, the type of files loaded or generated by the HST are defined accordingly by the nature of the cryptographic command under operation. The next pair of arguments is usually utilized together, as with [-l] and [-p] the IP and the port of the remote server is defined that will be used as log entry data collector, receiving JSON log alerts generated by the HST Logger in case of an unwanted event.

The variety of cryptographic protocols and algorithms that the HST supports cannot be adequately described and executed by the general format of the CLI commands presented above. For this reason, the [-c] argument encapsulates by itself a second level of CLI inputs and arguments linked exclusively with the execution of the various cryptographic functions and the necessary data for their completion. This encapsulated command is being parsed as a parameter of the [-c] argument, in the format [-c] “crypto_command”. There is a broad range of cryptography commands as shown below:

Symmetric Encryption/Decryption

For encryption and decryption functionality, the HST supports the common AES standard. The available modes of operation are both the Block Cipher Mode of AES-CBC and the authenticated encryption mode of AES-GCM. In order to speed up the encryption and decryption performance of these functions, the HST includes the acceleration component embedded as a coprocessor to the ARM-A53 processor. For example, a typical AES-GCM encryption command would be given as follows:

./hst [-i] [data_file] [-o] [ciphertext_file] [-c] “aesgcm [-e] [key_file] [nonce] [aad]”

In this example, an authenticated encryption using AES-GCM is taking place, encrypting the data stored in the *data_file* and using the key stored in the *key_file.* The nonce and the Additional Authentication Data (AAD) are parsed from the command line as HEX numbers with the prefix “0x”. This prefix is added to any variable that needs to be used as a raw number, and the HST handles the

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement **No 958478**

internal parsing and tokenization of the whole cryptographic command embedded in the quotation marks. The encrypted output can either be directly printed on the command line or stored in a file marked by the [-o] operator (in this case the *ciphertext_file*).

Message Integrity

Most embedded security applications require data integrity capabilities. The HST supports a variety of message integrity algorithms, like SHA128, SHA256, SHA384 or authenticated message integrity in the form of HMAC_SHA256. A sample command syntax follows:

./hst [-l] [dest_IP] [-p] [dest_port] [-i] [data_file] [-c] "hmac [key_file] [hash]"

In this example, it is assumed that we want to validate a given hash product with the same key it was produced. The hash is parsed in HEX form, and upon completion, the HST computes the hashed value of the *data_file* and compares it with the parsed hash. In case of a hash mismatch, the HST Logger generates a JSON log for a "Message Integrity failure" event.

Elliptic Curve Cryptography

Used mainly for authentication purposes, Elliptic Curve Cryptography (ECC) is widely adopted in the embedded domain by offering lightweight solutions for digital signatures and certificates compared to the more traditional RSA algorithm. Thus, the HST supports digital signatures using the Elliptic Curve Digital Signature Algorithm (ECDSA), hybrid encryption with the Elliptic Curve Integrated Encryption Scheme (ECIES), as well as key agreement protocols with Elliptic Curve Diffie-Hellman (ECDH). A sample command for issuing digital signatures with ECDSA is presented below:

./hst [-l] [dest_IP] [-p] [dest_port] [-i] [data_file] [-o] [signature_file] [-c] "ecdsa [key_file] [-s]"

The file containing the ECDSA signature is the product of this command, where an EC key (*key_file*) is used for digitally signing the data contained in *data_file*. The prefix [-s] denotes the signing procedure. Likewise, an ECDSA digital signature verification is shown in the below example command:

./hst [-l] [dest_IP] [-p] [dest_port] [-i] [data_file] [-c] "ecdsa [key_file] [-v] [signature_file]"

As can be easily deduced, the *signature_file* generated by the signing function is now an input argument of the [-v] operator. If the result of this function deems the signature invalid, the HST Logger generates a log alert with an "Authentication Failure" identifier. This log is both stored locally and sent to a remote server with IP and port issued by the [-l] and [-p] operators respectively.

Certificate Generation/ Validation

An additional functionality of the HST is the fact that it can act as a pseudo Certificate Authority (CA). The X.509 certificate protocol stack, alongside the encoding requirements presented by the ASN.1 encoding algorithm, are present in the HST and provide the functionality of generating and validating X.509 certificates. This capability has the potential to aid use-cases and application scenarios where a lightweight certificate management solution can provide authentication and trust between multiple entities. Given a scenario of that type, an assumption that must be made is that there must be an HST considered as a pseudo-CA, able to provide basic and lightweight certificates. A different HST that wants to own a certificate should firstly generate and transmit a valid Certificate Signing Request towards the CA. After that step, the CA can generate a certificate containing the requester's Public Key (PK). In the example below, a certificate validation operation takes place, validating with the CA's PK key whether a X.509 certificate is valid. If the validation is successful, the certificate's owner's PK is considered to truly belong to this owner, and it is extracted into its own key file ready to be utilized quickly and with established trust for various authentication purposes.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

./hst [-l] [dest_IP] [-p] [dest_port] [-i] [certificate] [-o] [public_key] [-c] “x509 [CA_key_file] [-v]”

With a method that has already been presented in the previous cryptographic examples, the certificate file is loaded as a parameter of the [-i] operator. The [-v] enables the validation mode for the *x509* command, and upon success the output file that is generated contains the certificate’s owner’s PK. If validation proves false for data integrity reasons or mismatches in either the CA’s PK or the certificate PK, then the HST Logger is triggered with an “Authentication Failure” identifier.

PostQuantum TLS 1.3 additions

The functionality of the Hardware Security Token has been enhanced by offering a modified version of TLS 1.3 using post-quantum resilient Digital Signature Schemes. Basis for this implementation is the robust and lightweight WolfSSL, an Embedded TLS Library, which has been in-house modified and adapted to utilize post-quantum schemes. More details on how the Wolfssl library was enhanced with quantum safe algorithms can be found in subsection 5.2.2

In order to properly setup a TLS connection, the certificate generation and validation capabilities of the HST were upgraded with the addition of lightweight X.509 post-quantum certificates. Built for ease of use in resource-constrained environments, these X.509 certificates are based on the Dilithium Digital Signature Algorithm and contain only the necessary fields required for a correct certificate validation.

Thus, WolfSSL’s source code was modified to handle the Dilithium certificates, and the library was compiled onboard the HST using the *arm-gcc-none-eabi* toolchain for the ARM Cortex - A53 chipset. The inclusion of the library to the HST is realized by the creation of a new command, focused on instantiating the TLS connection between the host and the client. A sample format of such command for the host side to setup a TLS connection and start listening for clients is presented below:

./hst96 -c "tls -svl TLS13-AES256-GCM-SHA384"

Likewise, for a secure file transfer to be realized to the host, a client can execute the following command, where [input_file] is the path to the desired file and [Host_IP] denotes the Host IP:

./hst96 -i [input_file] -c "tls -cvh [Host_IP] -l TLS13-AES256-GCM-SHA384"

By using this format, we take advantage of the functionality of WolfSSL through the CLI provided by the HST itself. In this instance, the agreed upon TLS encryption algorithm is the ARM accelerated version of AES256-GCM.

5.2.2. *Adding Quantum Safe TLS 1.3 in the HST*

As quantum computers are expected to become realistic in the next few years, the security/cryptography landscape will become completely different since all the traditional, widely used, public key cryptography algorithms will no longer be considered secure. This expected dystopic reality has pushed NIST (National Institute of Standards and Technology) to start in 2018 a competition for quantum computing safe cryptography algorithms to could act as secure public key cryptography standards in the presence of quantum computer attacks (stemming from Schor’s algorithm). The competition draws to a close in 2022 and the new Postquantum Cryptography standard will be announced. This will create an imminent need for implementation of the finalist (if not only the single finalist NIST candidate) into existing security protocols. In EnerMan, taking into account that new industrial control systems need to be secure and remain secure for a long period, we believe that this new cryptography reality should be included in all EnerMan related solutions. This rational has led to

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement **No 958478**

the adaptation of the extremely popular TLS 1.3 security protocol to a quantum safe version. In order for the TLS protocol to operate with post-quantum algorithms, the first action to be performed is to integrate an existing, secure, postquantum cryptography implementation like the PQclean library to the protocol. Similarly, the PQM4 library provides implementations of PQC algorithms that are optimized for ARM Cortex-M4 processors. As a starting point for a TLS library, we use the Wolfssl library that is one of the few TLS libraries supporting the latest 1.3 version of the protocol even for low-end embedded systems.

In the wolfssl library we integrate the lwIP library for supporting of network connectivity and then we adapt the TLS protocol itself in order to support the Key Encapsulation Mechanism (instead of the traditional key agreement schemes) provided by PQC schemes.

Apart from the code itself, the PQ algorithms require the usage of some cryptographic primitives. Specifically, all of the adopted algorithms make use of the Keccak primitives, SHA-3 and SHAKE-256 [6] The provided implementations of these algorithms from the mupq project [5]has been adopted to our work (a simple, C code implementation).

In order to make the wolfSSL TLS 1.3 messages on the handshake layer post-quantum compatible, the following architectural adaptations/adjustements have been made in the wolfSSL library:

- Extension of ``Supported Groups",
- Extension of ``Signature Algorithms",
- Key Encapsulation Mechanism changes/support and
- Post-Quantum Digital Certificates support.

In Figure 59, the overall post-quantum adapted handshake message exchanges are presented in detail indicating the PQ operations that are performed in each phase of the TLS handshake. One important thing to note, is that wolfSSL, when acting as a server using RSA, immediately after creating a signature, runs the Verify operation to check for signature faults. We mimicked this behaviour on our PQ TLS adaptation. Although it may be redundant, it only adds a minor overhead on our measurements. In the following subsections the changes are described in more details.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Figure 59 Created postquantum handshake version of TLS1.3 for embedded systems

Supported Groups

The ClientHello message, the first message that the client sends to initiate the handshake, consists of several fields, one of which is the Extensions field. In this field, the client extends the information provided by the rest of the ClientHello fields and plays a crucial role in TLS 1.3. One of the fields among the Extension field, as shown in Figure 59, is the field *Supported Groups*. In this field, the client sends a list of key exchange algorithms, as encoded identifiers (codepoints), in order of preference, so that the server can select one of them to be used in the handshake. These identifiers are called, Named Groups, and are defined for each supported algorithm by the protocol itself. To use post-quantum algorithms, we have introduced our own Named Groups. In order for the wolfSSL library to be inter-operable with other popular libraries, we decided to choose the codepoints that are being used by Open Quantum Safe fork of OpenSSL library. The codepoints for the post-quantum algorithms are shown in the following Table along with some traditional algorithms' codepoints.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Table 1. additional TLS1.3 Handshake codepoints for PQC schemes

Algorithm	*NIST Level*	*Codepoint*
FFDHE [1]	0	0x0100
ECDHE [2]	0	0x0017
Kyber512	1	0x023A
LightSaber	1	0x0218
Kyber768	3	0x023C
Saber	3	0x0219
Kyber1024	5	0x023D
FireSaber	5	0x021A

Algorithm	*NIST Level*	*Codepoint*
RSA [3]	0	0x0285
ECDSA [2]	0	0x0206
Dilithium2	1	0xFEA0
Dilithium3	3	0xFEA3
Falcon512	1	0xFE0B
Falcon1024	5	0xFE0E

[1] 3072-bit, [2] secp256r1 curve, [3] 2048-bit, [4] k

Signature Algorithms

Another useful field on the Extension field, is the ``Signature Algorithms'' field where the client provides its preference on the signature algorithms that it supports, regarding the CertificateVerify field. This means that this signature algorithm will be used to sign the transcript of the data exchanged by the server and to be verified by the client. Similar to the extension ``Supported Groups'', apart from the predefined codepoints for each algorithm we introduce our own codepoints for the post-quantum digital signature algorithms that the PQ TLS wolfSSL can support. The codepoints that have been added are compliant with the Open Quantum Safe fork of OpenSSL library, as shown in Table 1 along with some of the traditional algorithms' codepoints.

Key Encapsulation Mechanism Support/Adaption

All the post-quantum algorithms that resemble traditional key exchange schemes participating in the NIST competition are Key Encapsulation Mechanism (KEM) schemes. However, the key exchange method that is used in TLS is the traditional (Elliptic Curve) Diffie-Hellman Key Exchange. In order to adapt the key exchange to the post-quantum environment, in our proposed work the key exchange mechanism of TLS 1.3 is transformed into a KEM scheme through some architectural adaptation.

Initially, the client generates a key pair and sends the public key to the server with the ClientHello message. The server, using the client's public key, calls the Encapsulation function that produces a Ciphertext, which is sent to the client with the ServerHello message, and a Shared Secret, that the server keeps, as it is the actual shared key. The client, upon receiving the Ciphertext calls the Decapsulation function, together with its Secret Key and produces the same Shared Secret as the server. Now, both the client and the server, share the same key and have completed the key exchange scheme. These exchanged messages are shown in Figure 59 as the Ephemeral PQ Key Generate, PQC Encapsulate and PQC Decapsulate operations.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Digital Certificates Support

Another important object that needs to be modified in order for the TLS to work with post-quantum algorithms, is the digital certificate. These are objects that bound an entity, for example a server, with its public key by introducing a signature from a trusted third party. This can occur repeatedly by intermediate third parties, forming what is known as a ``chain of certificates''. The X.509, the standard that digital certificates usually follow on protocols like TLS., contains useful information about the entity, like for example: the entity's name, email, web address etc, the issue and expiration date of the certificate, the public key of the owner, the digital signature algorithm code that is used, the digital signature itself, etc.

For the production of these digital certificates using post-quantum cryptographic algorithms, the Open Quantum Safe's fork of OpenSSL was used. Through this library we generated digital certificates using the OpenSSL's API with support for all the post-quantum algorithms that are evaluated in this paper. Our goal is to produce a digital certificate for the server, as it is the only one to authenticate itself. To achieve that, we introduced a base ``Certificate Authority'' (CA) that can issue other certificates making a chain of trust up until the server. In our paper, this chain is of length two, as the server's certificate is directly signed by the CA. To accomplish this, we created a digital certificate for the CA, which is self-signed and then we produced a digital certificate for the server which is then signed by the CA. Thus, a server certificate is produced, verifiable by our basic CA.

For the sake of simplicity, all the certificates in the chain employ the same signature algorithm each time. This is also the case for both the certificate's signature and the signing operation on the CertificateVerify message. For example, when measuring the performance of Dilithium2, CA's certificate and server's certificate have Dilithium2 signatures and the CertificateVerify message is signed using Dilithium2, as well.

5.3. AI Industrial Intrusion Detection

5.3.1. *Set-up the board*

For this demo we are using Pynq, which is an open-source initiative that facilitates the use of Xilinx hardware devices, such as the EnerMan MPSoC. PYNQ offers and supports Python Productivity bootable images that can be used on a variety of Xilinx development boards, which host the Zynq MPSoC, e.g., Pynq-Z1, Pynq-Z2 and ZCU104. In addition, using the Pynq python library we can deploy Python scripts on the reconfigurable hardware (FPGA) of the MPSoC. In this demo we use the ZCU104 evaluation board.

5.3.2. *Set-up the host*

The training phase is conducted on a host PC using the Xilinx FINN docker container with all the tools and libraries for training the target AI model. Xilinx provides *Jupyter* Notebooks within the docker for easier development. The host also needs Xilinx Vivado HLS installed, for the generation of the bitfile that will be downloaded into the reconfigurable part (Programmable Logic (PL)) of the MPSoC, which is linked with the docker container. To run the docker we just have to run on the host the command *./run-docker.sh notebook*.

5.3.3. *Train a quantized MLP with Brevitas*

Quantize the dataset

The first step for a Quantized Neural Network (QNN) training is to binarize the dataset. In this demo we are using the TON_IoT modbus dataset created by the UNSW Sydney. This can be achieved with a python script called dataloader_quantized.py. This script drops irrelevant columns of the dataset such as date, time, and the type of the attack and binarizes all the useful data for the training and keeps

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

the label that shows whether we have an attack or not for this data. With this dataset, for every input of four integers, the dataloader creates 80 bits. The final quantized dataset is saved in a numpy compressed format (.npz). This script also divides the dataset into training dataset (~80%) and test dataset (~20%).

Define and train the QNN with Brevitas

For the training phase we use the quantization-aware training (QAT) capabilities offered by Brevitas. Our MLP has four fully-connected (FC) layers in total: three hidden layers with 64 neurons, and a final output layer with a single output, all using 2-bit weights. We also use 2-bit quantized ReLU activation functions and apply batch normalization between each FC layer and its activation. The number of epochs is 10 and the learning rate 0.01. The notebook gives us post-training information about the training loss and test accuracy. The final test accuracy of the model is 0.88.

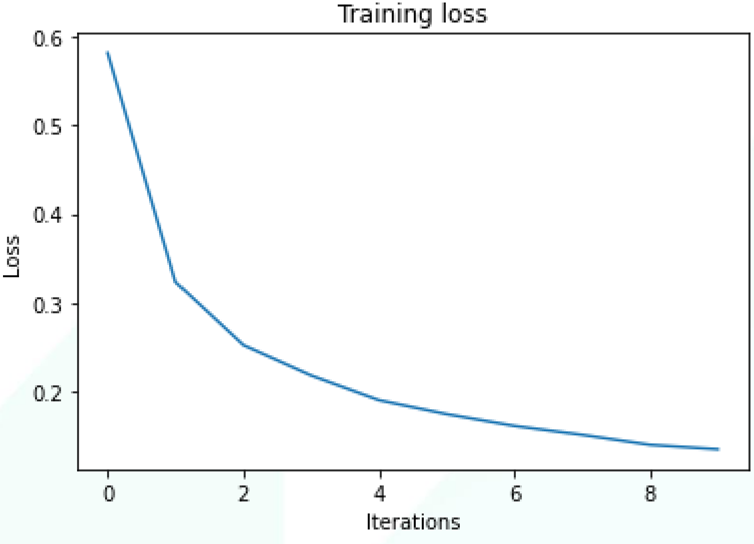

Figure 60 Training loss per iteration

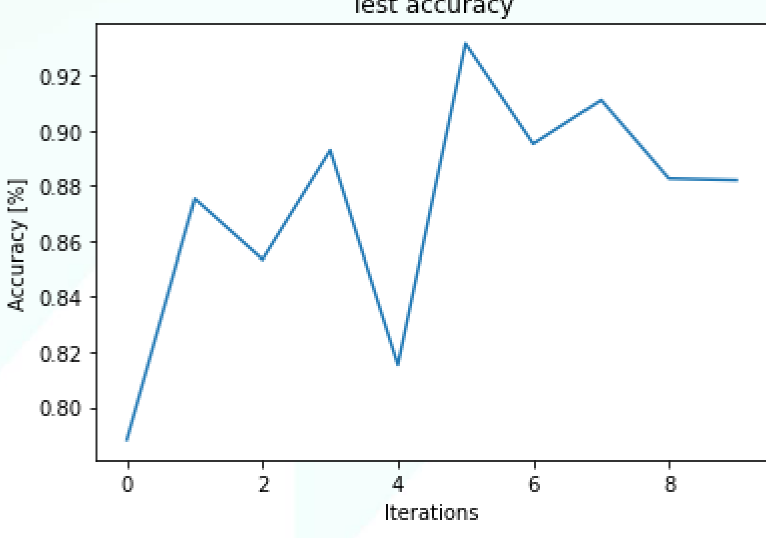

Figure 61 Test accuracy per iteration

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

Export ONNX model

Before exporting we can make some changes to our trained network (network surgery). In this case we are padding the input. Our input vectors are 80-bit. The FINN compiler expects an ONNX model as input. ONNX is an open format built to represent machine learning models and the output of our model is presented below (Figure 62).

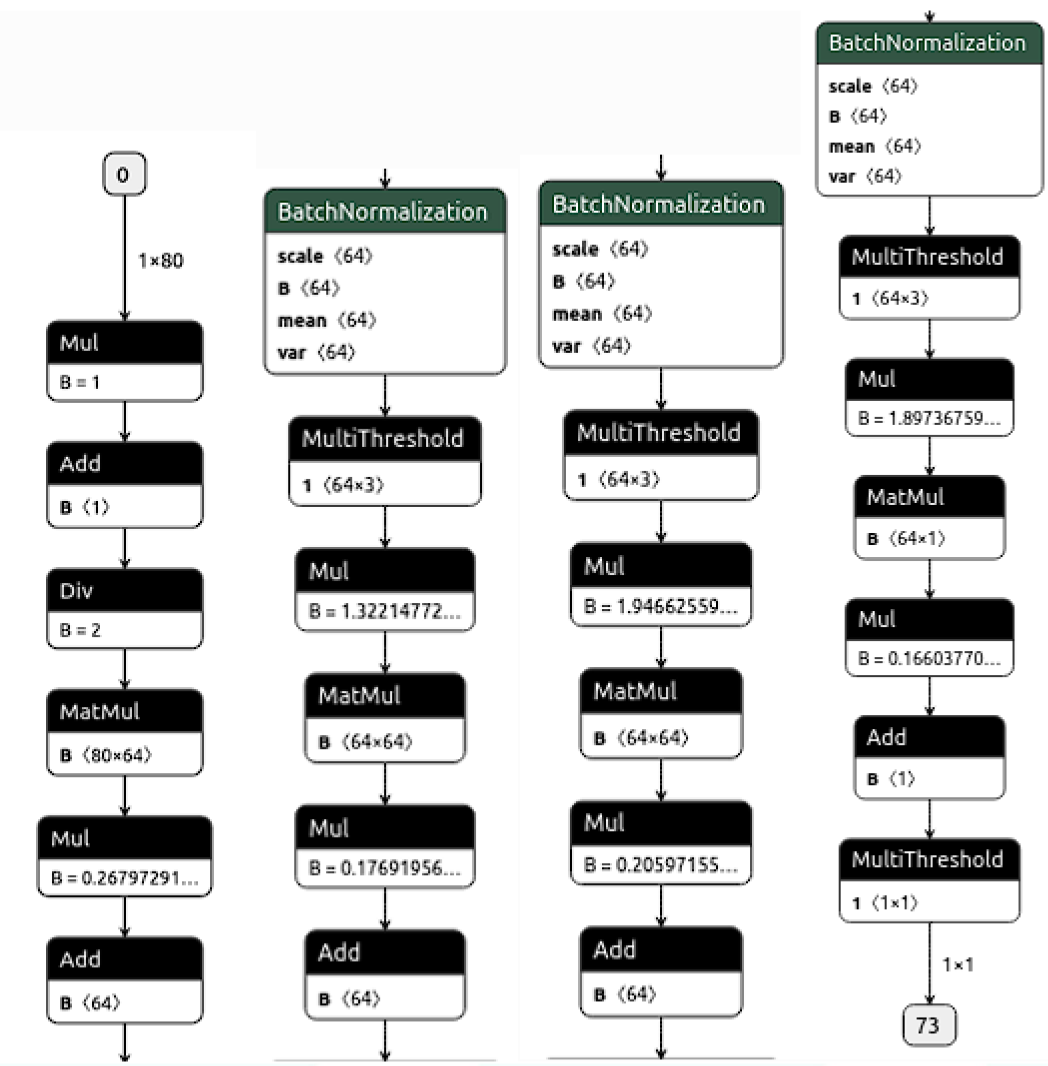

Figure 62 The exported ONNX model in Netron

5.3.4. *Import model into FINN and compare it with Brevitas execution*

To import the ONNX model into FINN, the wrapper around the ONNX model provides several helper functions so we can extract information about the structure and properties of the model. Before the comparison with the Brevitas execution, we need to prepare our FINN-ONNX model. With the Graph transformations in FINN we transform the model into a synthesizable hardware description. Finally, we can compare the two models by calling our inference helper functions for each input and comparing the outputs.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

5.3.5. *Synthesis of the accelerator and generation of the bitfile*

In this step we use the FINN compiler to generate an FPGA accelerator with a streaming dataflow architecture from our QNN. With the use of the Vivado HLS we map all the layers of the model into hardware description. Hence, we create a hardware architecture with parallel layers that are connected with FIFOs to form a full accelerator. Because the synthesis phase is time consuming, we always test our architecture through rtl simulation.

```
Final outputs will be generated in output_final
Build log is at output_final/build_dataflow.log
Running step: step_qonnx_to_finn [1/17]
Running step: step_tidy_up [2/17]
Running step: step_streamline [3/17]
Running step: step_convert_to_hls [4/17]
Running step: step_create_dataflow_partition [5/17]
Running step: step_target_fps_parallelization [6/17]
Running step: step_apply_folding_config [7/17]
Running step: step_generate_estimate_reports [8/17]
Running step: step_hls_codegen [9/17]
Running step: step_hls_ipgen [10/17]
Running step: step_set_fifo_depths [11/17]
Running step: step_create_stitched_ip [12/17]
Running step: step_measure_rtlsim_performance [13/17
Running step: step_out_of_context_synthesis [14/17]
Running step: step_synthesize_bitfile [15/17]
Running step: step_make_pynq_driver [16/17]
Running step: step_deployment_package [17/17]
Completed successfully
CPU times: user 1.64 s, sys: 283 ms, total: 1.92 s
Wall time: 26min 7s
```

Figure 63 The steps towards the bitfile generation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

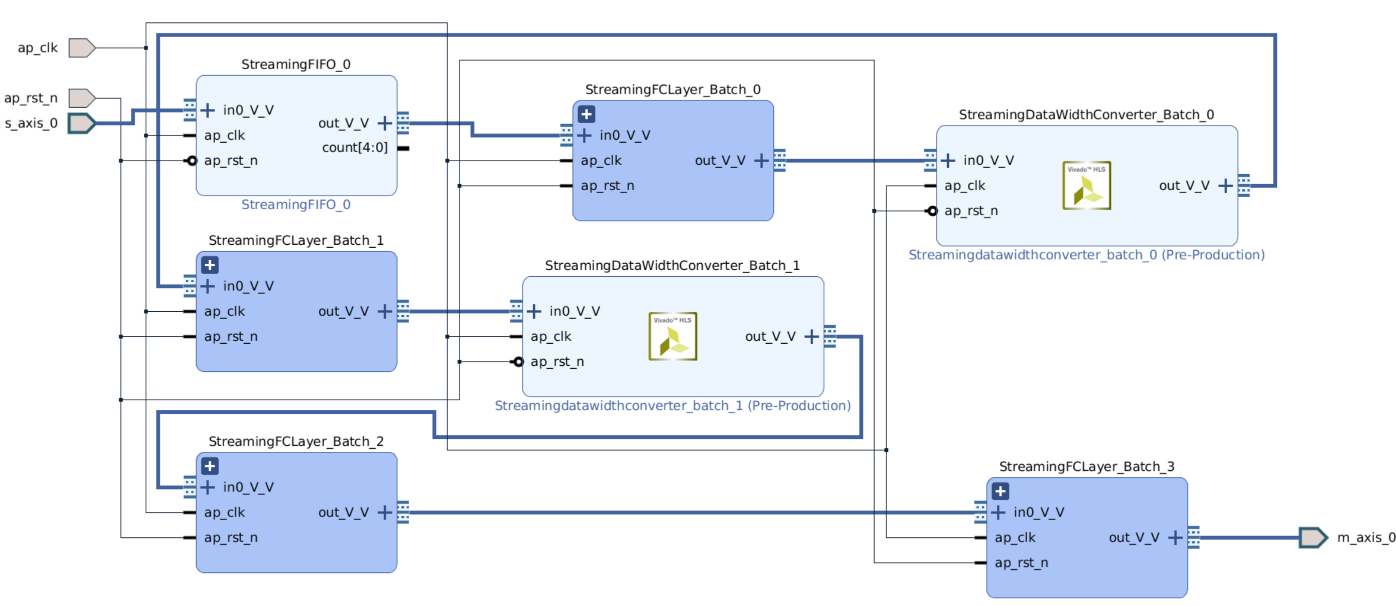

Figure 64 Block design architecture in Vivado

The final output of this process is the bitfile (and the accompanying .hwh file) that will be downloaded to the board (Figure 63). To test the accelerator on the board, we put a copy of the dataset and a premade Python script that validates the accuracy into the driver folder, then make a zip archive of the whole deployment folder. Finally, we send the zip folder to the board and run the commands below for testing our accelerator.

unzip deploy-on-pynq.zip -d finn-I2DS-demo

cd finn-I2DS-demo/driver

sudo python3.6 -m pip install bitstring

sudo python3.6 validate_TONIoT.py --batchsize 1000

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

```
Loading dataset...
Initializing driver, flashing bitfile...
Starting...
batch 1 / 10 : total OK 901 NOK 99
batch 2 / 10 : total OK 1790 NOK 210
batch 3 / 10 : total OK 2671 NOK 329
batch 4 / 10 : total OK 3556 NOK 444
batch 5 / 10 : total OK 4428 NOK 572
batch 6 / 10 : total OK 5303 NOK 697
batch 7 / 10 : total OK 6192 NOK 808
batch 8 / 10 : total OK 7084 NOK 916
batch 9 / 10 : total OK 7948 NOK 1052
batch 10 / 10 : total OK 8825 NOK 1175
Final accuracy: 88.250000
```

Figure 65 Terminal Output

Figure 65 Terminal Outputshows a typical example of the type of results that the execution of the AI model on hardware, generates. Hence, processing is applied on sets of data that are fed into the AI processing model in batches. In the figure above, the test data have been split and fed into the model in ten distinct batches. Subsequently, when the complete set of data has been processed, the process returns the effectiveness of the particular model given the specific set of data, i.e., accuracy. Note that the TSI I2DS solution can easily achieve an accuracy of 90%, which can be considered high level.

Finally, Table 2, lists the characteristics and specifications that are particular to the example demonstrated here. It pertains to the actual hardware resources occupied as well as the latencies that result from different stages of the process. For instance, fold and pack input relate to the preparation of the actual data for introduction into the model, whereas the copy_input_data_to_device and copy_output_data_from_device indicate the cost of moving the data between the CPU and accelerator memories.

Table 2 Metrcics of I2DS

LUTs	3208
FFs	4643
BRAM	7
DSPs	0
Throughput (im/s)	1177844
cycles	218
runtime (ms)	0.849
fold_input (ms)	0.067
pack_input (ms)	3.040

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

copy_input_data (ms)	0.980
unpack_output (ms)	253.8
copy_output_data (ms)	0.261
unfold_output (ms)	0.040

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

6. CONCLUSION

In this deliverable, the final research, design, and implementation activities of T2.1 and T2.4 as well as some implemented components of T2.2 have been presented. This complements the work presented in D2.1 from M6 to M14 by adding missing aspects that were available at M18 when the WP2 is concluded. The report of the overall work that has been performed in WP2 tasks 2.1 and 2.4 can be captured in its wholeness by considering both D2.1 and D2.2 deliverables. This includes the design and development of the EnerMan intelligent edge node execution environment and its various components, the design and implementation flow of intelligent operations to be executed in the edge node (using as proof of concept the energy consumption prediction case), the data harmonization mechanism and the security related operation on the edge and between the edge and the system layer.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**

REFERENCES

[1]. Yeqi Liu, Chuanyang Gong, Ling Yang, Yingyi Chen, DSTP-RNN: A dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction, Expert Systems with Applications, Volume 143, 2020

[2]. J. O. Ramsay and B. W. Silverman, Functional Data Analysis, Springer, 2005.

[3]. J. S. Morris, "Functional regression," Annual Review of Statistics and Its Application, vol. 2, p. 321–359, 2015.

[4]. Y. Xu, Y. Yang, T. Li, J. Ju and Q. Wang, "Review on cyber vulnerabilities of communication protocols in industrial control systems," 2017 IEEE Conference on Energy Internet and Energy System Integration (EI2), 2017, pp. 1-6, doi: 10.1109/EI2.2017.8245509.

[5]. https://github.com/mupq

[6]. DWORKIN, Morris J., et al. SHA-3 standard: Permutation-based hash and extendable-output functions. 2015.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement **No 958478**

Energy Efficient Manufacturing System Management

enerman-H2020.eu

enermanh2020

enermanh2020

enermanh2020

HORIZON 2020
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement **No 958478**