ICT-56-2020 - Next Generation Internet of Things

D 7.5
Final report on use case development, optimisation, benchmarking and evaluation

Document information	
Contract number	957197
Project website	www.vedliot.eu
Dissemination Level	PU
Nature	R
Contractual Deadline	31.01.2024
Author	Micha vor dem Berge (CHR)
Contributors	Oliver Brunnegard (Magna), Carina Marcus (Magna), Johan Thor (Magna), Nils Kucza (UNIBI), Franz Meierhöfer (Siemens), Roland Weiss (Siemens), Yufei Mao (Siemens), Andreas Ask (EmbeDL)
Reviewers	Mario Porrmann (UOS), Hans Salomonsson (EmbeDL)

The VEDLIoT project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957197.

Changelog		
v0.1	2022-01-20	Initial draft of D7.2
v0.2	2022-04-01	Initial input of all partners merged
v0.3	2022-04-05	Summary, Introduction, Conclusion added
v0.4	2022-04-08	Updating all chapters, ready for internal review
v0.5	2022-04-18	Internal review back
v0.6	2022-04-25	Updated all chapters after internal review
v1.0	2022-04-30	Final version (D7.2)
v1.1	2022-08-26	Initial version of D7.3, based on D7.2
v1.2	2022-10-07	Major content of all chapters updated
v1.3	2022-10-18	Ready for internal review
v1.4	2022-10-25	Updated all chapters after internal review
v1.5	2022-10-31	Finalization
v2.0	2022-10-31	Final version (D7.3)
v2.1	2023-12-05	Initial version of D7.5, based on D7.3
v2.2	2024-01-08	Major content updates of all chapters merged
v2.3	2024-01-09	Updated Summary, Introduction, Conclusion
v2.4	2024-01-10	Consolidated version ready for internal review
v2.6	2024-01-25	Updated version, based on internal review
v2.7	2024-01-29	Consolidated version, pre-final
v3.0	2024-01-30	Final version of D7.5

Table of contents

Executive Summary

This deliverable describes the use case developments and optimisations and the transition from traditional algorithms towards machine learning. It covers the work from M7 to M39 of the project and is based on the previous deliverables D2.3 [1], D7.2 [2] and D7.3 [3].

In the beginning of the project, for all four use cases, unified and formalised specifications were defined, covered in D2.3. Based on this, the four use cases started to develop a basic working prototype (in case of the home assistant use case, it was already available from a previous project) which acted as a platform to define the baselines for the selected metrics and KPIs, which is mainly covered in D7.2 and D7.3. Also, these prototypes provided a platform to gather lots of different measurements for ML learing training data. After this phase was finished roughly at the mid of the project, the neural networks were defined, trained and optimised in iterative loops. With these trained neural networks, the benchmarking and evaluation phase at the end of the project started. Various environments and situations were measured and compared to the baseline benchmarks from the beginning of the project. Also, for all use cases, visually attractive demonstrators were developed for fairs, conferences and the reviews.

The major outcome for the IIoT use case of Motor Condition Classification is the setup of a testbench, the capturing and labelling of training data and the design and development of a neural network and the development, manufacturing and integration of a use case optimised cognitive hardware platform where the neural network runs on. Also, an AR visualisation based on a tablet was developed and Christmann's Secure IoT Gateway integrated for high network security. Furthermore, a draft for the integration of the SIRE protocol and external MQTT for system security was developed. Overall, the targeted KPIs of e.g. a maximum power consumption of 6.6 mW was met, the system only needs 5.8 mW under certain conditions. Also, the total energy consumption of 30 Wh per year is reached and the NN model accuracy can reach 88.6% after training and 82.3% after quantization on the test dataset.

The IIoT use case for Arc Detection re-activated, improved several times during the project and secured an existing testbench for arc generation and measurements to generate training data for the NN development. With this training data, a NN was designed, trained, optimized and ported to run on different hardware architectures: CPU, embedded GPU AI accelerators and FPGAs. The NN optimization with the VEDLIoT toolchain resulted in over 80% reduction on inference time compared to 10 ms for one inference during the initial implementation. The real running arc testbench was also re-designed to make it portable for demonstrations on fairs, conferences etc. The achieved NN accuracy of up to 100% is a great success and an excellent base for future product developments.

The Automotive AI use case implemented the Pedestrian Automatic Emergency Breaking (AEB) as goal to be realized with the help of a NN. Therefore, vast amounts lots of training data was captured, labelled and a NN was designed and trained. It took several iterations and optimizations until the NN reached the desired quality for local processing, and it was optimized for embedded GPU compute units and ported to the also developed u.RECS. Further research fields of distributing the NN processing via different wireless networks (WiFi or 5G mmWave) in various ways to near edge and far edge computing units resulted in the knowledge that for a time-critical use case like the AEB, distribution is not well-suited due to the latency, but technically possible. The latency during the experiments overshot the KPI by four to five times. As most of this latency can be referred to the data transfer and not the NN operations, there is room for improvement. Regarding the accuracy of the results from the NN it was shown that it was able to detect the dummy representing a predestrian that was used and didn't mistake another test object, a trash can, for a dummy.

The Smart Home use case improved the pre-existing smart mirror demonstrator in the course of the VEDLIoT project massively, building several attractive prototypes for lots of fairs and conferences where it was shown with great success. Lots of developments have been accomplished like the change of the internal framework to ROS2, restructuring of the Face Recognition NNs were merged and optimised, porting and optimising towards heterogeneous hardware like different Nvidia embedded ARM/GPUs, dedicated AI accelerators like the Hailo.8 and FPGAs, the creation of a virtual mirror image from 3D point clouds and the development and integration of on offline voice assistant were added. Many of these developments led to improved user experience and functionality, but also enhanced the energy efficiency. The latest measurements, taken on an Nvidia AGX Orin and Hailo8 AI acceleratod resulted in 1.266 Watt / FPS, having 9.375 Watt / FPS as baseline which is a 7.4x improvement and further improvements towards 0.933 Watt / FPS are expected if the hardware was further optimised.

1 Introduction

This deliverable describes the developments and optimisations as well as the optimisation, benchmarking and evaluation of the four project use cases. They are classified in three sections:

UC1A Smart Industrial IoT: Motor Condition Classification use case
UC1B Smart Industrial IoT: Arc Detection use case
UC2 Automotive AI use case
UC3 Smart Home use case

All four use cases have been formally described and defined in the confidential Deliverable 2.3 [1]. For reasons of readability, however, all use cases are presented briefly so that this deliverable is self-contained. All use cases have been developed and optimised, from using traditional algorithmic methods to machine learning and energy-efficient hardware, using the VEDLIoT tool flow and methods as shown in Figure 1. These developments and optimisations are described in this deliverable. They are partially based on previous work, but some use cases have been developed from scratch.

It is important to mention that this deliverable is the third of three consecutive deliverables, so it is partially based on D7.2 [2] and D7.3 [3], but describes the final state of developments and optimisations and gives an overview of the impoved metrics and KPIs.

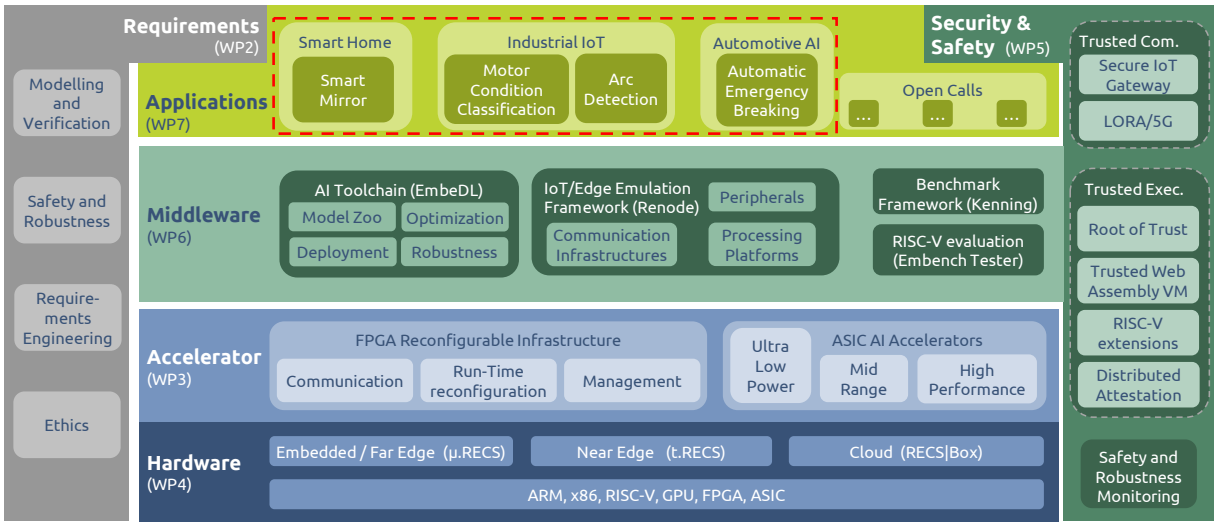


Figure 1: Global picture of the VEDLIoT project, the use cases are outlined in red

2 Smart Industrial IoT: Motor Condition Classification Use Case

This chapter describes the development and result of the industrial use case motor condition monitoring. This use case aims to demonstrate the utilisation of deep learning in industrial scenario by using ultra-low power cognitive Internet of Things (IoT) devices. The application area is the condition monitoring of middle size motors, especially the monitoring of the status of cooling systems. The chapter introduces the background of the use case, the development procedure, and the evaluation of the system.

2.1 Introduction

Condition monitoring is an important topic in an industrial environment. The Equipment is required to run 24 /7 and the condition monitoring system should be constantly available. The demand for system performance enhances the request for condition monitoring and predictive maintenance during runtime. The Current solution is based on IoT systems where end devices, which are typicaly based on microcontrollers, are mounted on equipment to monitor different features of those devices. Figure 2 shows an example of a battery powered smart field device (SFD) in real industrial application. The SFD is expected to last at least two years on battery power and should be easy to mount, maintain and replace. However, network congestion and limited bandwidth in large-scale IoT systems pose a significant hurdle. To address this, on-site data processing within SFDs using deep learning is explored to reduce the volume of data transmitted via radio, necessitating the integration of DL algorithms into resource constrained SFDs while maintaining hardware power efficiency.

Figure 2: Smart Field Device for Industrial IoT [source: Siemens]

The distributed computing- and DL-power provides improved adaptability and thus also improved flexibility for the whole system. The useful usage of SFDs for motor condition classification depends highly on the power class of equipped motors. For large drives, the implementation of special monitoring systems is more efficient than SFD. For small drives, even the costs of an SFD are too high compared to the costs of the motor. So, the expected range of target drives is within an axis height of 150 to 400 mm or a power range of 5kW to 500 kW. This is also the state-of-the-art solution in industrial operational environment [4].

However, the cost-effective adoption of ML-based monitoring systems utilizing Surface-mounted SFDs faces several key challenges, primarily in the realms of data generation, validation, and labeling, as well as energy efficiency. The diversity of states for training poses a substantial challenge. Recording training data for cooling condition classification demands comprehensive coverage of error types, such as loose or broken fan blades and obstructed air inlets. Similarly, for mechanical condition classification, accounting for error types like loose machine basements, broken bolts, misalignment, housing damage, and bearing issues is crucial, among others. Energy efficiency emerges as a critical concern, especially for

battery powered SFDs with a lifespan of around two years. Integrating additional DL capabilities amplifies the energy efficiency challenge. Extending battery life necessitates implementing a low-power operational state (sleep-mode) as the primary mode, supplemented by brief, infrequent algorithmic slots managed by a sophisticated power manager. This combination is essential to achieve prolonged battery life while ensuring optimal functionality.

2.2 Overview of Developments and Optimisations

This section provides a brief review of the use case background and the development progress compared to the previous deliverables the D7.2 first report on use cases [5] and the D7.3 second report on use cases [6].

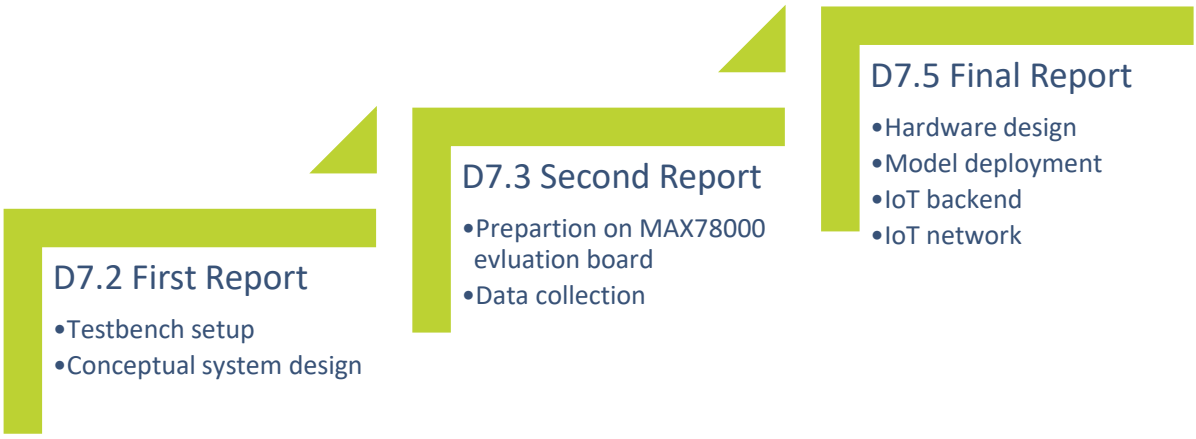


Figure 3: Development process presented in deliverables

The first stage of the development is described in D7.2 where a test-bench as field simulation was developed. This test-bench was built based on a small size motor of 750W, powered by a single-phase inverter, and the setup of the medium size motor of over 55 kW, which is the target system in real use scenario, is delayed due to delivery of missing components. At this stage, an initial concept for utilizing DL and IoT on the use case was also depicted in D7.2.

The result in between from the second stage is presented in D7.3. At this point, the hardware was selected based on the benchmarking on various AI accelerators. MAX78000 was chosen for the cognitive IoT device and experiment was conducted for the operation of MAX78000. At the meantime, data collection procedure started on the test-bench. The setup of the middle size motor is still delayed.

This deliverable presents results from this use case and the major development and improvement since D7.3. During the final stage, a model is trained based on the given data and the IoT hardware is designed, manufactured and assembled. Both components are implemented on the test-bench. Besides, an IoT system around the testbench has evolved with the help of other project partners for the improvement in aspects of security and integrity. Along the development process, results in between are evaluated and further challenges are addressed. In order to address these challenges, problem analysis techniques are deployed with the help from project partner University Gothenburg for better design of AI-based solutions.

2.3 System Design

The basic requirement for this use case is to monitor three independent conditions with DL models, as already introduced in D2.3 [7]:

- Operational state (ON / OFF)
- Cooling system state
- Mechanical state

Due to the complexity of the problem, the development of the demo focuses on the classification of the state of the motor cooling system. The solution for this can be simple if a large and expensive sensor system with multiple temperature sensors or air flow sensors is deployed. But such a large and expensive complex system is not feasible fro mid-sized motors due to economic reasons. An attachable one-piece sensor device without external temperature, airflow or pressure sensors is a more realistic option.

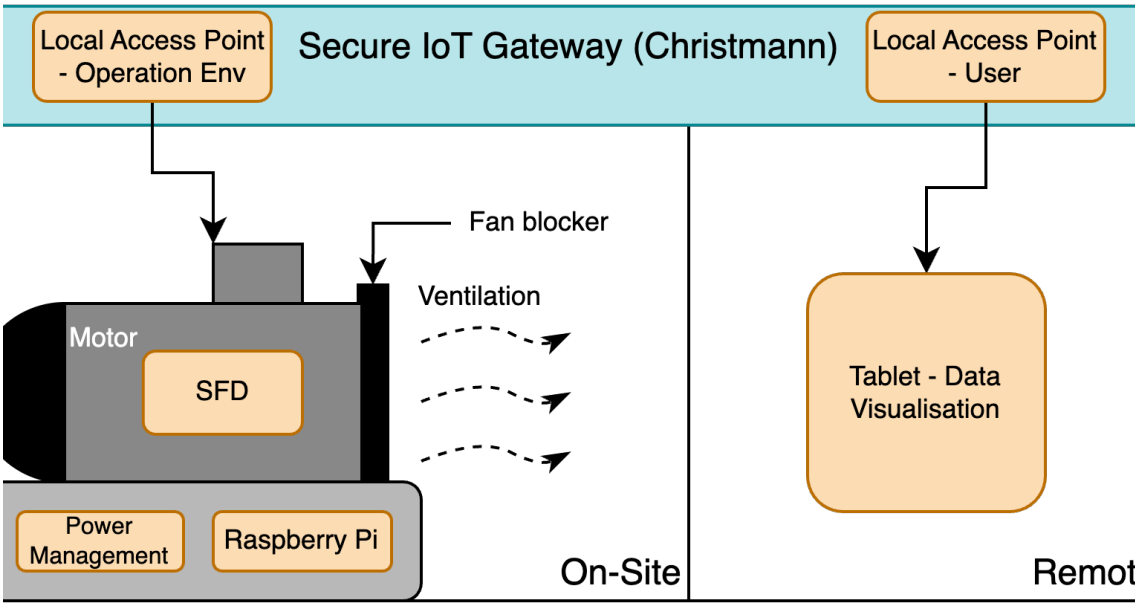


Figure 4: Motor monitoring system overview

The demonstration system is depicted in Figure 4. The on-site side of the figure is the testbench we built for the simulation of the operational environment. This part consists of an SFD for sensing and data processing, a power management device to control the motor operational status, and a raspberry pi as a local server that exchanges information between the on-site devices and the remote users through the secure IoT gateway. The remote user side is designed as applications on tablets to control the motor, to fetch data from the system, and to visualize the data in real time.

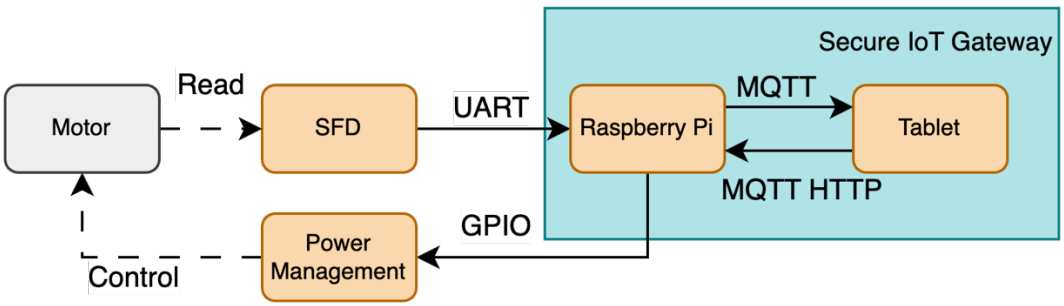


Figure 5: Data flow in the motor monitoring system

The communication between these two parts in the system is shown in Figure 5. Bilateral communication is demonstrated there. The sensor data is sent from the SFD to the tablet. The control signal are sent from the tablet and are forwarded to the power management system to control the operation of the motor. The SFD records sensor data from the environment and processes it with embedded deep learning model. All the data are forwarded to the raspberry pi 3B+ via a UART interface. The raspberry pi further publishes the data to a MQTT server. On the other side, users that subscribe to the motor data can access the information by using the MQTT protocol. In this way, the data is forwarded to the user side and further visualised. On the other hand, the raspberry pi also runs an HTTP server, which port listens to the control signal sent from the connected user. Users can control the speed of the motor by designed interface.

2.4 Deep learning on smart field device

This section presents the development of the essential components of the use case demonstrator -- the deep learning model and the hardware design. The description of the test bench setup and the data collection as preparation is presented separately in D7.2 [5] and D7.3 [6]. In this deliverable, the efforts for the design of the algorithm around the deep learning model and the related results are presented. In the end, the hardware design and the model implementation is summarized. Details about hardware design and its workflow were described in detail in D4.7 [8].

2.4.1 AI-powered Smart Field Device on the test bench

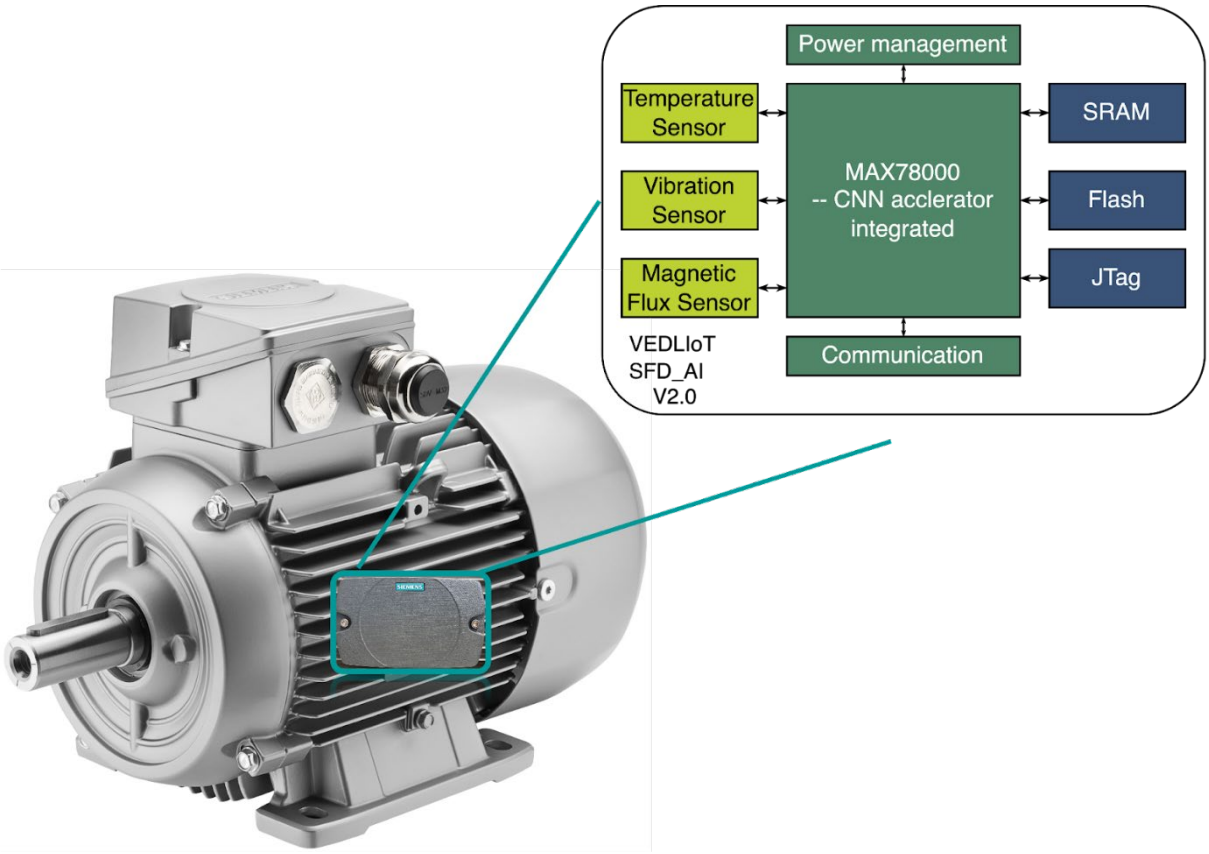


Figure 6: SFD with dedicated AI-Accelerator as a "mount-on" solution

The testbench setup is depicted in D7.2 where the motor with a shutter for the simulation of environmental effects on the cooling system is included. An SFD without AI power was first attached to the motor and has been utilized for data collection. Along the development of the use case, the AI powered SFD is designed, manufactured, tested, and further evaluated on the test-bench. The new SFD installation is depicted in Figure 6.

2.4.2 Data collection

Data collection strategy is described in D7.3, where sensor data under different situations are collected. The feature dimensions of the data are rotation per minute (rpm) and shutter position. They represent the motor speed and the condition of the cooling system condition representatively. A data sample from the small motor testbench with unblocked cooling system is depicted in Figure 7. To demonstrate the operational environment where motors are switched on and off alternately, the motor on the testbench is turned on and off periodically (every ten minutes, 120 data points for one time interval). The figure denotes the motor status with vertical dashed line. At the first phase in the figure, the motor is turned on and the temperature decreases because the cooling system is working. When the motor is off on the second phase, the temperature increases because the cooling system stops to work when the motor is switched off. The residual heat generated by the motor during operation can gradually disperse into its surroundings. Compared to the data from

operational environment depicted in Figure 13, similar temperature change tendency can be observed.

On the other hand, the rate of temperature change exhibits a direct correlation with the motor's rpm as shown in Figure 7. Elevated motor speeds result in a proportionately increased rate of temperature change within the surrounding environment.

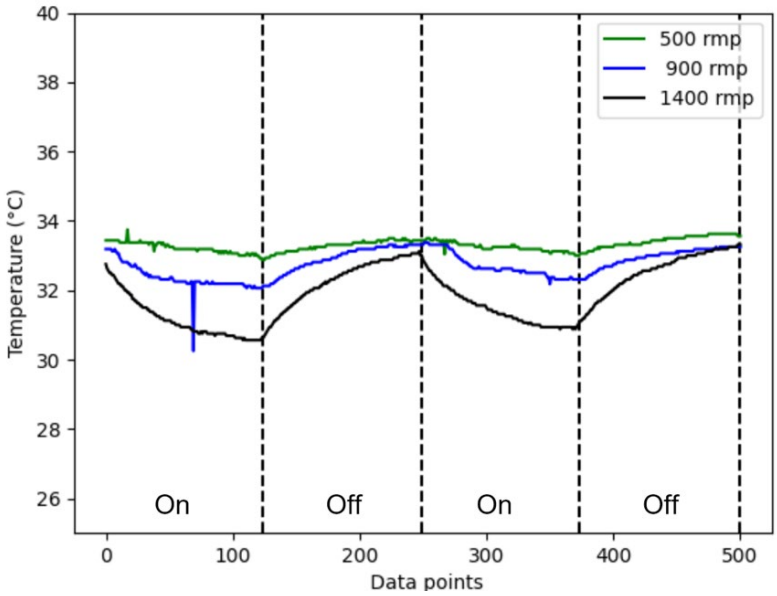


Figure 7: Data example from small motor testbench, cooling system unblocked

For better understanding of the data and the problem, Figure 8 depicts the collected temperature data under different cooling system conditions – blocked air intake and free air intake. It is obvious that when the cooling system is blocked, the temperature is likely to stay stable and do not change even when the fan is working along the motor operation.

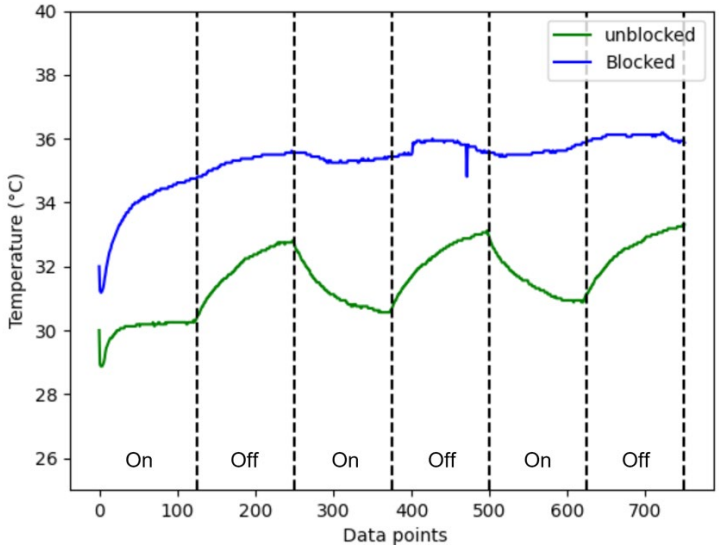


Figure 8: Temperature change when cooling system is blocked and unblocked

The data collection is conducted based on the matrix shown in Deliverable 7.3. The two dimensions mentioned above are covered: rpm and shutter position (air intake). The number of data used for model training is approximately 24000 datapoints, which sums up to around 33 operation hours of operation.

2.4.3 Algorithm

The model used for the cooling system condition classification is trained according to the workflow as described in Deliverable 4.7, where the hardware, which is based on the microcontroller MAX78000 with integrated DL-acclerator, and thecorresponding workflow is described. For the model training and algorithm implementation, the already mentioned standard procedure is used: data pre-processing, model training and post-processing.

2.4.3.1 Pre-processing

For the model training, only data from temperature sensor are used, because the data from the magnetic flux sensor and the vibration sensor are stable. The reason for that is the lack of variability of the rotation speed during the operation of the motor. For the data pre-processing stage, a suitable input size from the time series data points should be selected. After hyperparameter tuning, the window size is settled to 128 data points that represent the temperature change within 5 minutes. After that, input data frame is quantized and normalized as preparation for a quantization-aware training procedure. Figure 9 depicts an example of for a normalized and quantized data input.

To improve the model performance, data augmentation is also implemented to enlarge the limited dataset and increase the variety of the datasets. Three techniques for data augmentation are implemented: jittering, time wrapping and slicing, as well as averaging and interpolation. Jittering adds random noise to existing data points, time wrapping and slicing stretch existing time series data by different factors; averaging and interpolation combines two time series data with same window size and create average series data. The combination of three augmentation techniques shows slight improvement of the performance of the model. The accuracy of model has improved slightly – around 1% increment with an extra augmented dataset.

2.4.3.2 Model training

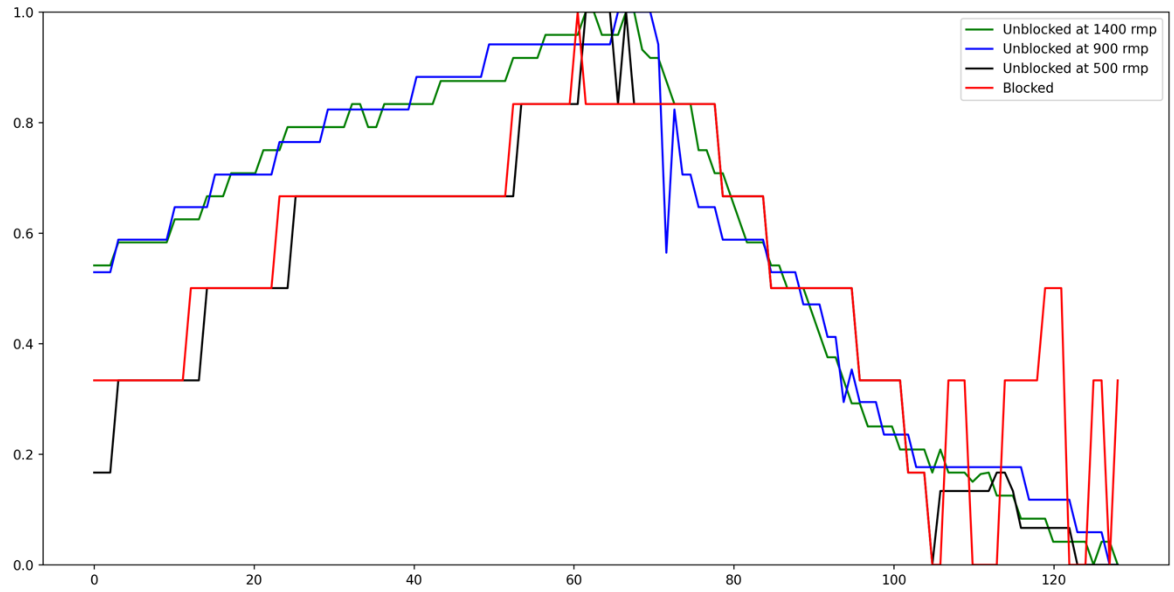


Figure 9: Quantized input data after normalization

The model training is similar to a conventional ML model training with pytorch library, but it needs to be designed with consideration for the way in which processors and instance memory device are operating to ensure the optimal memory and processor usage. Details for the training workflow are described in Deliverable 4.7.

The model implemented for the cooling system condition classification is a 1D convolutional neural network (CNN) with three convolutional layers and one dense layer. Two layers are fused with max pool layer and all three convolutional layers come with ReLU as activation functions. Details of the model structure are shown in the code snippet in Figure 10. The model class is comprised of both model architecture and model parameters. The filter size specified here with "128" and "64" in each layer is compatible with the CNN engine processor unit for optimal DL acceleration on MAX78000. The number of filters for one-dimensional time series in the model reflects the number of channels connected to specific processors, which should also be divisible by 4 to facilitate easier activation for 4 processors at a time, as each of the 4 processor is connected to a specific instance memory. In case the number of the inputs channels or filters exceeds the limit of the 64 activated processors, a multi-pass technique is employed. For technical details regarding processor allocation and multi-pass refer to [9]. The architecture is then followed by flatten layers that connect the convolutional part to softmax for classification. To optimize processor usage. The number of flatten layer's nodes is also designed to be divisible by 4, depending on the channel number of the last convolutional layer and the length of of the time series, which may vary after each layer depending on the number of paddings strides. Additional layers, such as dropout, can be incorporated for regularization.

```
class AI85motorNet2_on_off_off_on(nn.Module):
  def __init__(
      self,
      num_classes=3,
      num_channels=1,
      dimensions=(128, 1),  # pylint: disable=unused-argument
      bias=False,
      **kwargs
  ):
    super().__init__()
    self.drop = nn.Dropout(p=0.2)
    # Time: 128 Feature : 13
    self.current_conv1 = ai8x.FusedConv1dReLU(num_channels, 64, 3, stride=1, padding=0, bias=bias, **kwargs)
    # T: 126 F: 64
    self.motor_conv1 = ai8x.FusedMaxPoolConv1dReLU(64, 32, 2, stride=1, padding=1, bias=bias, **kwargs)
    # pool-stride is 2 and pool-padding is 0
    # T : 64  F: 16
    self.current_conv2 = ai8x.FusedConv1dReLU(32, 16, 3, stride=1, padding=0,
                                bias=bias, **kwargs)
    # T: 62 F: 16
    self.motor_conv2 = ai8x.FusedMaxPoolConv1dReLU(16, 16, 2, stride=1, padding=1,
                                   bias=bias, **kwargs)
    # pool-stride is 2 and pool-padding is 0
    # T : 32  F: 16
    self.fc = ai8x.Linear(512, num_classes, bias=bias, wide=True, **kwargs)

  def forward(self, x):  # pylint: disable=arguments-differ
    """Forward prop"""
    # Run CNN
    x = self.current_conv1(x)
    x = self.drop(x)
    x = self.motor_conv1(x)
    x = self.current_conv2(x)
    x = self.drop(x)
    x = self.motor_conv2(x)
    x = x.view(x.size(0), -1)
    x = self.fc(x)
    return x
```

Figure 10: Code for CNN model structure

The CNN accelerator of the MAX78000 employs signed integers for weights storage and calculation. However, in the training phase, floating-point values are commonly applied to both data and weights, constrained within a specific range. Quantization is then required to save the memory and to reduce the energy consumption of the calculation. This reduction in memory size is accompanied by a compromise in model accuracy. To evaluate the impact of weight quantization on model precision, an assessment of the quantized model and the original model in terms of accuracy and performance metrics was made.

The evaluation was carried out using identical hyperparameters and augmentation values that were used during training the normal weights. These values were then applied to train the quantized weights. Finally, the evaluation was performed on the same test set to assess the performance of the model with the new quantized weights. When it comes to quantization, all metric values experience a decline. This occurs because the values are no longer free to take any arbitrary value. Instead, they need to be quantized within a specific range determined by the weights. While this drop in performance cannot be avoided, it serves the purpose of reducing memory requirements. By quantizing the weights, the memory usage can be reduced by a factor of four. However, this reduction in memory comes at the cost of a decrease in accuracy, typically around 6%.

Table 1: Evaluation on quantization

Data Type of Model Weights	Model Precision	Model Recall	Model F1-score	Model Accuracy	Negative Accuracy
Float32	0.9138	0.9268	0.9203	88.64%	75.86%
Int8	0.8835	0.8718	0.8766	82.78%	72.19%

2.4.3.3 Post-processing

Post-processing constitutes a pivotal phase of the algorithm, contributing significantly to system enhancement and reliability. The strategy voting system implemented in this use case involves executing model inferences repeatedly and collating the outcomes. Increasing the number of inferences provides the model with more opportunities to detect mistakes, ultimately improving the accuracy of the classification. In the case of the motor monitoring system, which is not a time-constrained problem, a time delay caused by multiple inferences is not considered critical.

During the experiment, four system designs are evaluated: one inference time system, a triple voting system, a 5-time voting system, and a 7-time voting system. The final result is based on the classification results of all inferences through an additional voting process. For instance, if majority inferences indicate a blocked state and one inference indicates an unblocked state, the final decision will be a blocked state. Figure 11 depicts the inference trend for three inference times system designs. In the case of a triple voting system, the system maintains the storage of 75 data points. With each new data point, the oldest point is replaced by the newest one. This process continues until there is a switch in the operational state from on to off. At that point, the system collects the remaining 53 data points, concatenate them with the 75 data points, and performs the CNN model inference. Following this, it acquires 11 new data points, replaces the oldest 11 points, and conducts the second inference where the operational state change is in the middle point at data point 64. This process is repeated for the third inference where the change of the operational state occurs at data point 53.

For the 5-time voting system, the same procedure is followed, but the model starts by saving 85 data points instead of 75. The inference in five votes, with the change of the operational state positioned in the middle of the data points or shifted by 11 and 22 data points to the left and right. Similarly, For the 7-time voting system, the same procedure is followed, but the model starts by saving 96 data points instead of 75. The inference in five votes, with the change of the operational state positioned in the middle of the data points or shifted by 11, 22, and 33 data points to the left and right.

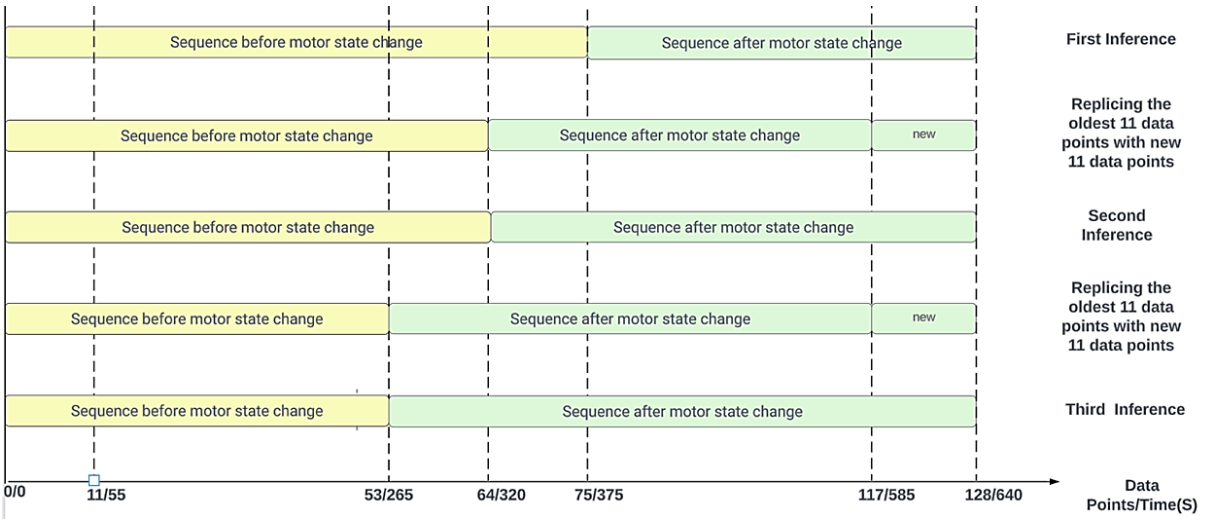


Figure 11: Triple inference voting system

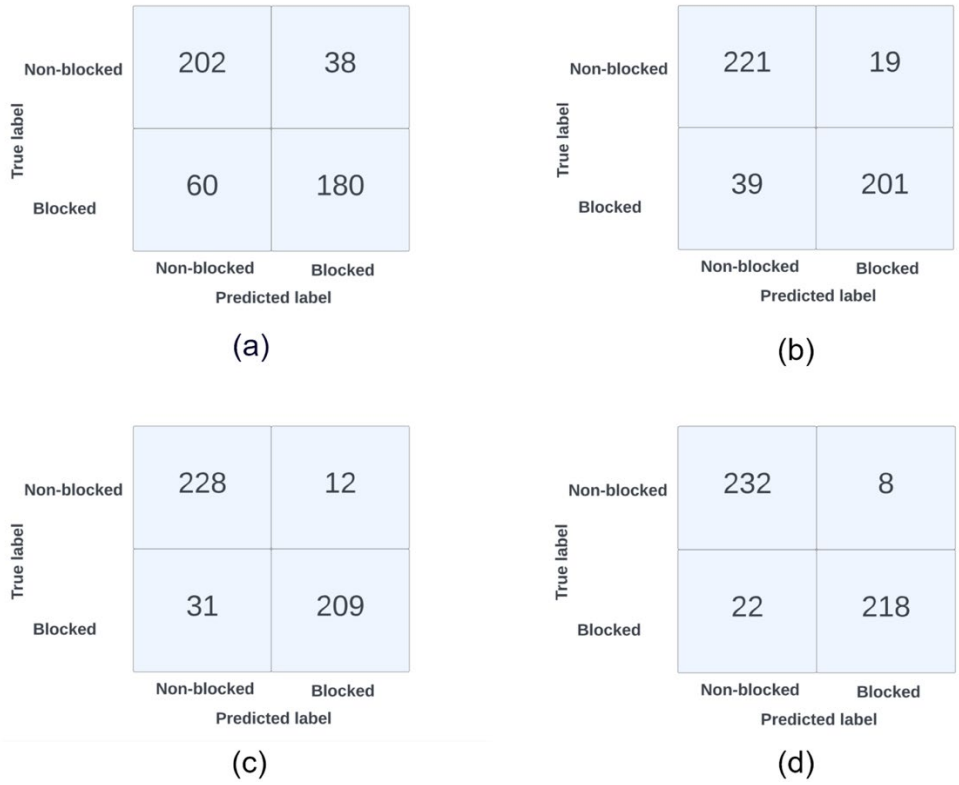


Figure 12: (a) One-time inference system, (b) Triple voting system, (c) 5-times voting system, and (d) 7-time voting system

The evaluation of these different sequences is conducted by the assessment of accuracy across the various cases described above. The results shown in Figure 12 indicate that the inference accuracy on one same test dataset improves as the voting system incorporates more votes. The accuracy of the algorithm with the voting systems for 7-time, 5-time, 3-time, and single inference are 93.75%, 91.04%, 87.91%, and 79.85% respectively. This trend is expected, as the higher the number of votes, the more room there is for mistakes to occur. Worth to mention is that the accuracy presented in Table 1 is based on shuffled dataset, however the voting system requires unshuffled dataset for the evaluation, thus leads to different model accuracy.

It is notable in Figure 12 that the improvement is skewed towards the unblocked class, as its accuracy is 84% is higher than that of the blocked class (accuracy 75%). This discrepancy in accuracy originates from the higher probability of the better class being selected when two or three votes are considered in the system. Therefore, the improvement for the blocked class is more pronounced compared to the unblocked class.

With an inference time of 202 μs, the CNN model in our case demonstrates suitability for time-constrained prediction problems. This indicates that the accelerator can deliver efficient and timely predictions within the required time constraints for other applications.

2.4.4 Limitation

The hardware implementation has confirmed the practicality of integrating AI methods into SFDs. It also demonstrates the viability of the workflow in creating AI-driven solutions within Cyber-Physical-System (CPS). Nonetheless, in pursuing further advancements and potential product development, it's crucial to address existing limitations of the above described result.

2.4.4.1 Limitations of testbench setup and data collection

With resoect to the testbench setup, which was used for data collection and demonstration can hardly simulate or even represent the real scenario. First of all, the motor type due to its small size cannot simulate the temperature characteristic of target medium size motors, whose temperature change characteristic is depicted in Figure 13. Besides, the rotation speed of the motor in the testbench setup has low variation. This leads to highly constrained datasets with limited hidden features and information.

The inverter of the testbench for the medium size motors was damaged during tests for another project in the first phase of the VEDLIoT Project. The replacement components had a long lead time and were delivered late in Q3 2023. At this time the data needed for the DL based evaluation within the project was already recorded and prepared.

Furthermore, a strategic reorganization within the SIEMENS AG impeded access to other similar sized testbeds and to their datasets.

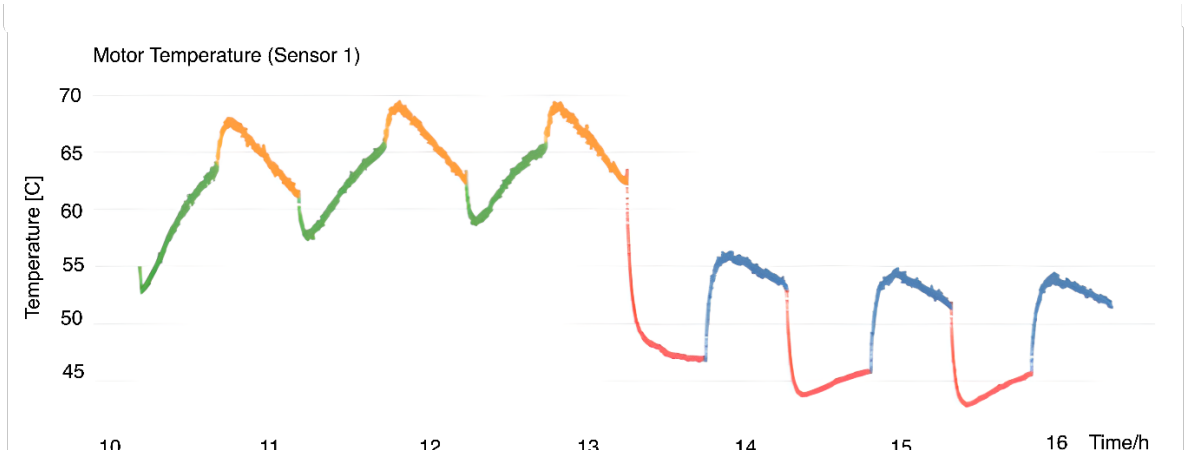


Figure 13: Example of temperature data for the middle size motor, from D7.2

Furthermore, time for data collection is a bottleneck for the development of industrial AI based use cases, because compared to e.g. image processing there are no large pre-processed data sets for typical industrial applications available.

As described before, the model is trained with a data set that represents 33 hours of experimental recording time. This does not include time between every experiment, and this is necessary to have motor cool down to room temperature. For each factor combination e.g. rpm of 1400 when air intake is blocked, data collection of 1440 data points (2 hours) is minimal for one scenario. This also requires turning motor on and off every 10 minutes.

2.4.4.2 Limitation on model

The size and quality of the dataset affects the performance of the trained model. For this binary classification used for the cooling condition monitoring, a thorough evaluation on the model is conducted, and its performance on different data classes are compared.

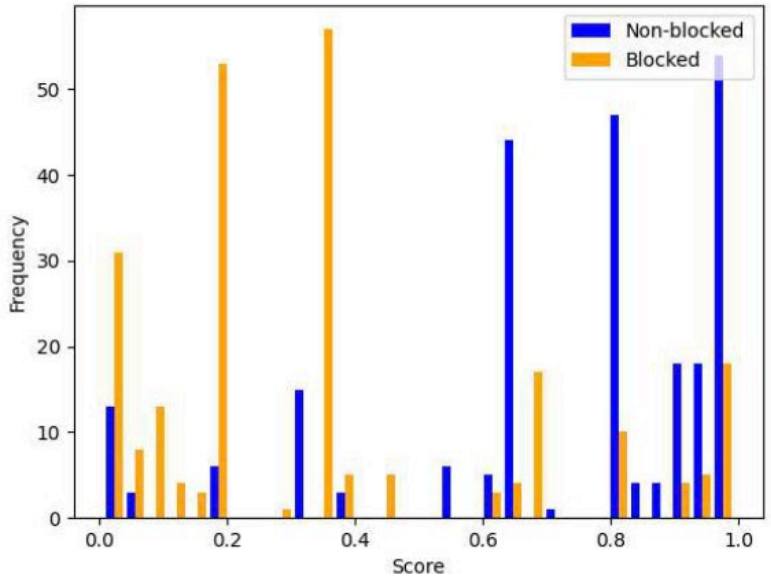


Figure 14: Histogram of model inference result on the test dataset

The histogram shown in Figure 14 provides insights into the condition results. The score represents the probability for unblocked condition. In the case of the unblocked condition, there are three dominant scores: 0.9930, 0.6633, and 0.8133. The highest score of 0.9930 and 0.8133 is associated with the 1400-rpm and 900-rpm conditions respectively, while the

score of 0.6633 is more commonly observed for the 500-rpm condition and rarely for the other two conditions.

For blocked conditions, the same three dominant scores are present: 0.007, 0.3367, and 0.1867. The most dominant score is 0.3367, which corresponds to a score of 0.6633 for the blocked case. This suggests that the model struggles to differentiate between the blocked case and the unblocked case at 500 rpm. Misclassification between the 500-rpm low speed and blocked cases occurs due to the relatively small temperature changes observed for both conditions.

This pattern strongly correlates with the constraints of the testbench, particularly in scenarios where temperature variations remain indiscernible at low rotation speeds, especially in smaller motors. This circumstance can be improved by applying a load to the motor, albeit this adjustment may amplify power consumption, consequently resulting in more pronounced temperature alterations.

2.4.4.3 Limitation on hardware

The MAX78000 hardware imposes constraints on the network configurations, affecting parameters and capacity [9]:

- One-dimensional convolutional layers are limited to kernel lengths of 1 to 9, with specific padding options and stride values. Dilation is constrained based on kernel length and limited to 1 for longer kernels.
- Input and output channel capacities are capped at 1024, with bias support restricted to 512 output channels.
- Layer count is limited to 32, excluding pooling and element-wise operations preceding a convolution, while data dimensions are restricted to 1023 rows or columns.
- Weight memory capacity varies based on kernel size and bit usage, diminishing with increased channel counts.
- Data normalization involves mapping values to 256 specific levels rather than scaling between 0 and 1. This method is critical for optimal hardware CNN engine operation.
- Deviations beyond a 1/256 value range post-augmentation and normalization might not significantly alter data, potentially leading to overfitting risks. Careful adjustment of hyperparameters is crucial to avoid this and to achieve the desired effects.

2.4.4.4 Limitation on power consumption

Another challenge for the deployment of the DL model in real application scenario is the power consumption. The model is trained with data collected every 5 seconds. However, in an operation environment, the SFDs is designed to collect data for certain time interval every five to ten minutes. The SFD in an industrial environment usually in sleep mode most of the time to save energy. Otherwise the battery will be empty after several weeks and not after several years. In this case, it is difficult to implement a model that requires data for a duration of 5 minutes or more.

2.4.5 Problem Analysis

For this use case only the result from the second stage is presented, because the validation of the causality graph on it is intractable.. Besides, conditions simulation based on the causality graph also requires more discussion on feasibility and safety issues due to the complexity of the problem. Furthermore, massive data collection is also time-consuming due to the nature of the problem.

Based on the given requirements for the motor condition monitoring use case and the limitations of the testbench of the use case, workshops for a thorough problem analysis were conducted with a novel approach for an optimized design of an AI-based software solution.

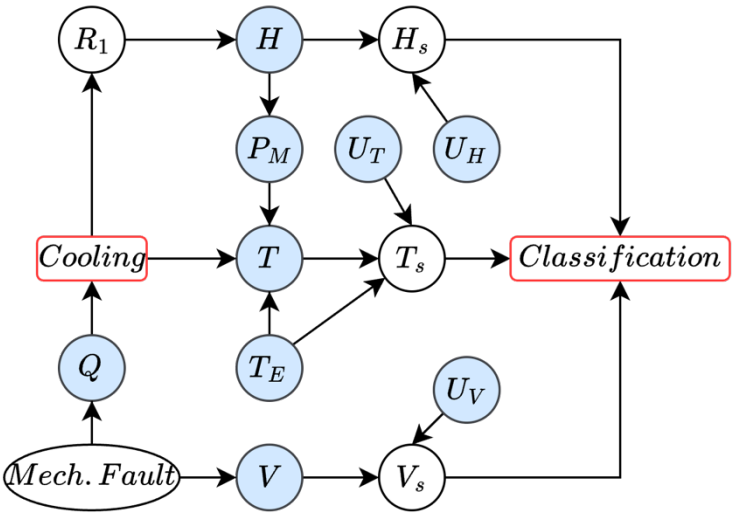


Figure 15: Causal model for motor monitoring

This approach is proposed and largely supported by the University Gothenburg. Engineers with experience in related field are involved for the discussion and derivation of the causal graph shown in Figure 15. The graph starts with the classification object "cooling" and the classification result "Classification". Factors that are involved in the causality derivation between them are deduced. The white circles indicate observable parameters and grey circles indicate unobservable factors. Table 2 lists all the variables in the causal graph. Based on the analysis, requirements for further developments can be derived in aspect of data collection and model training:

- Sensed temperature should be conditioned on environmental temperature.
- The final classification of the status of cooling system should take temperature criteria and vibration criteria into consideration.
- All input measurements in dataset should be augmented by characteristic sensor noises.

The input layer must take data from the measurement of the temperature, the magnetic flux, and the vibration.

Table 2: Variables for motor condition monitoring causal graph

Variable	Definition
$Cooling$	Fan system status
Q	Airflow
Mech. Fault	Mechanical fault of motor
P_M	Mechanical power
R_1	Electrical (inner) losses
T_E	Environmental temperature
$U_H\ U_T\ U_V$	Unmeasured noises

$T\ (T_s)$	Surface temperature (measured)
$H\ (H_s)$	Magnetic Flux (measured)
$V\ (V_s)$	Vibration (measured)

This causality modeling completes the development procedure and improves problem understanding for all stackholders. This approach is especially beneficial for the design of AI-based solution for Cyber Physical Systems (CPSs). It does not only provide a systematical framework for system analysis and derivation of data and model specification, but also a proper standard for the documentation of the generation of datasets.

2.5 Implementation on IoT system

Other than the integration of deep learning in the IoT end device, the development of IoT infrastructure is also an important part in the development of this use case. The infrastructure consists of a secure IoT network provided by Christmann, raspberry Pi backend service setup and an AR applications for an improved human machine interface (HMI).

2.5.1 Secure IoT Network

The infrastructure for the secure data transmission is provided by the project partner Christmann. The system consists of two access points in different locations, one for user and the other one for the on-site system as depicted in Figure 4. A central monitoring interface is provided as shown in Figure 16. The interface can be used to monitor the status of the access points and connected devices, and to configure the communication protocols.

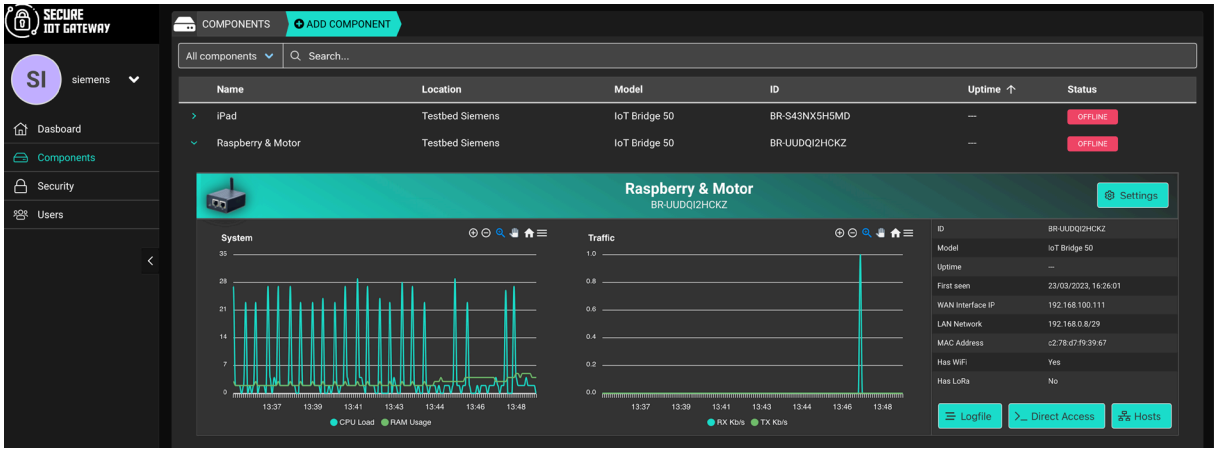


Figure 16: Central monitoring interface for the Secure IoT Gateway

2.5.2 Raspberry Pi Backend Server

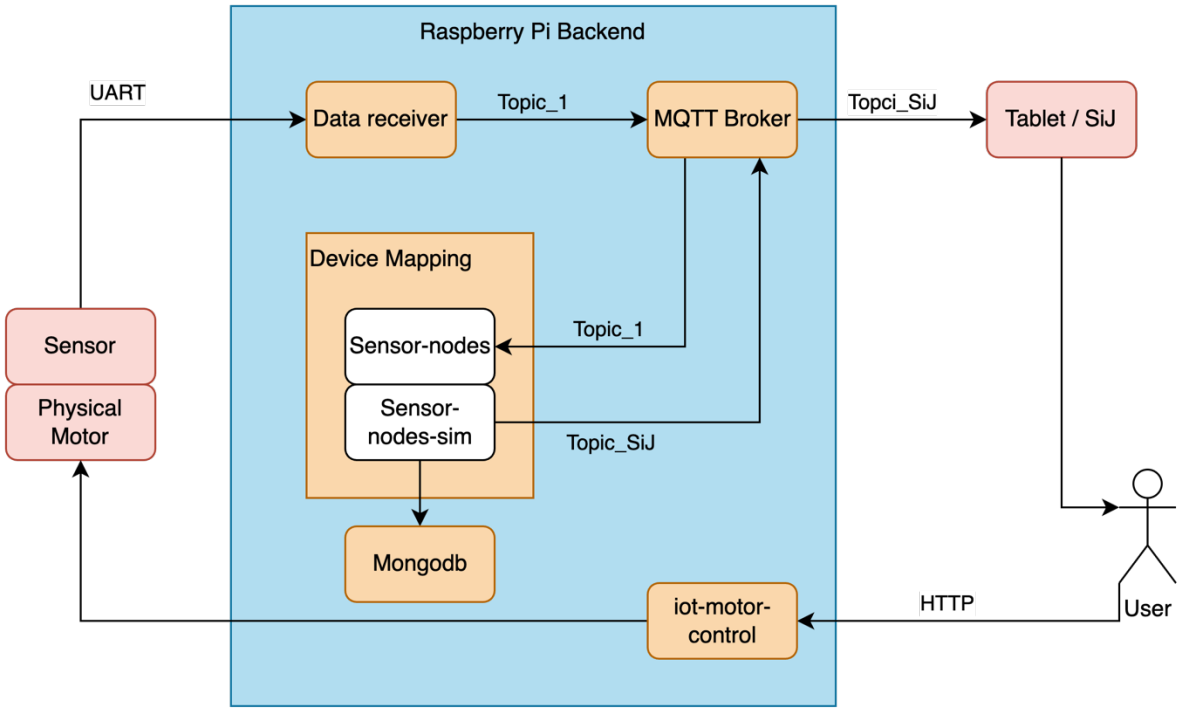


Figure 17: Raspberry Pi as backend server with multi functions integrated

The raspberry pi serves as the central backend server and is the core of the IoT infrastructure. The services installed on the raspberry pi are shown in Figure 17. The sensor sends data to the raspberry pi through a UART interface. A customized data receiver script collects the data and forwards the data to a MQTT broker. Currently, the MQTT broker is deployed on the raspberry pi. In the backend, a device mapping server is also present and subscribes to all messages from the sensor. This function is for the AR interface, which requires a coaty [10] protocol for the data transmission. The Device Mapping creates a virtual sensor node in the server and converts all the sensor data to the coaty standard. The message is then published to the MQTT server and subscribed by the AR application on the tablet, a copy of the data is also stored in the database, here the mongodb [11] is deployed.

Besides, the user can control the motor remotely with a HTTP requests. A web server is running on the raspberry pi server and the requests are converted to a signal to the power management unite of the demonstrator. Thus e.g. the speed of the motor can be changed.

Also, the integration of the SIRE protocol for device authentication in cooperation with University Lisboa is also in progress. The SIRE attestes of the connected end devices and make the authenticity of them available for the user. And the MQTT broker is also about to be replaced by an external MQTT server from the University Neuchatel to enhance the data security of the motor condition monitoring system.

2.5.3 Human interface

AR applications for Android and iOS systems are developed as an easy to use HMI. As the operation systems are different, applications are developed with different libraries and approaches. Since the development of such applications require large time investment and aesthetic design is not the focus, the applications demonstrate the data flow to the user end and work as prototype for data visualization.

The iOS application is designed with the coaty library. The application could subscribe to the MQTT topic channel. The data is transmitted and received in JSON Format. An example of interface is shown in Figure 18. In this figure, temperature is displayed as a bar chart and motor startus is shown as an emoji.

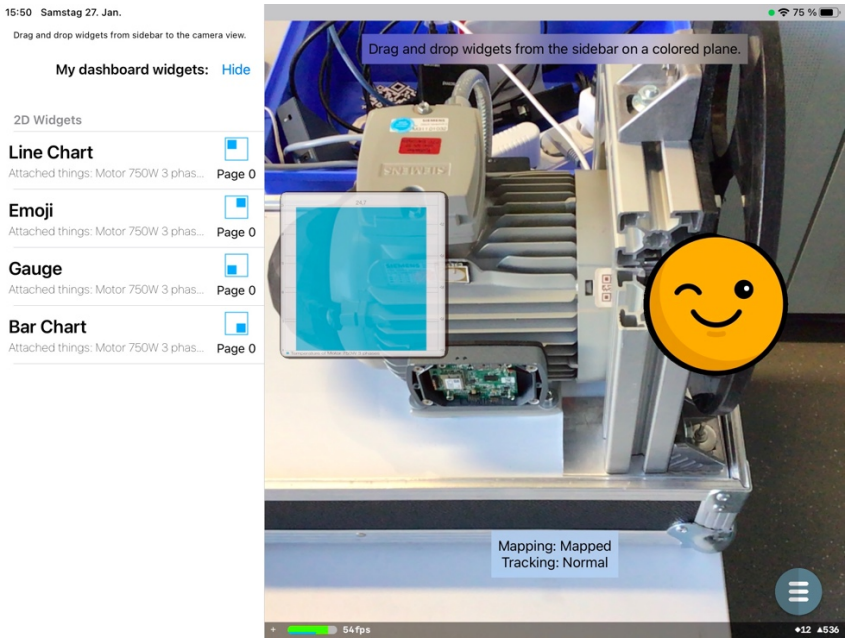


Figure 18: iOS AR interface under development

2.6 Conclusion

The motor condition monitoring use case demonstrates the possibility of integrating a deep learning model in on-site SFDs. The following achievements have been made during the development process:

- The cognitive hardware was designed, soldered, tested, and iterated to a second version.
- The first model was trained with a dataset and further quantized.
- The first monitoring algorithm was developed and deployed on the hardware.
- The cognitive IoT hardware was tested on the test bench.
- Problem analysis was undertaken as the first step for the second development iteration in cooperation with the University of Gothenburg.
- The first IoT system with a backend (Raspberry Pi 3B+) and a frontend (AR on a tablet) was designed and prototyped.
- The Secure IoT Gateway was integrated in cooperation with Christmann.
- An initial system evaluation was conducted.
- The first draft for the integration of the SIRE protocol and an external MQTT server for system security was developed in cooperation with the University of Neuchâtel and the University of Lisbon.

For benmarking, since signal processing was not implemented on SFDs, the goal set in Delivearble 7.2 is used to evaluate the final result. The achieved improvements regarding the KPIs are listed below:

- Memory: 128kB SRAM and 512kB flash integrated in the controller meets the set requirement. Besides, 1Mb external SRAM and 128 Mb external flash memory are also integrated in the PCB, which exceeds the set goal and provides flexibility in furture application.

- Cost: The MAX78000 as the micro-controller costs 15€ per piece. The price can reach 13€ when order with big amount. The cost of the system is around 60€ for all the main components. Therefore, the set price requirement 14 € [controller] / 70 € [System] is met.
- Power: the KPI was set to 6.6mW. With the system settled on MAX78000, the power consumption of the controller is 5.8mW under certain conditions.
- Energy: the system can reach set goal on power consumption of 30 Wh per year given duty cycle of two inference per hour and 7 minutes of data measurement per inference. (Refer to the energy estimation in Deliverable 7.3)
- Accuracy: The model accuracy can reach 88.6% after training and 82.3% after quantization on the test dataset as mentioned in Table 1. The accuracy depends highly on the quality of data. Various limitations are listed in section 2.4.4.

The power consumption of the controller can vary based on multiple factors. It can be optimized by changing the operation mode of the controller and the peripheral devices, it can also change based on the clock frequency set for the processor and the scheduling for the model inference. Thus, the power consumption given here is a realistic calculation result based on the system design. A detailed discussion on the factors for energy calculation can be found in Deliverable 4.7.

3 Smart Industrial IoT: Arc Detection Use Case

This chapter is a detailed report on the solution for integrating deep learning into the industrial IoT use case – direct current (DC) series arc fault detection. This solution is developed along the course of the VEDLIoT project and has received support from many project partners. The research on this use case focuses not only on the technical implementation, but also on improvements in the development procedure for the general implementation of deep learning methods on far edge devices in industrial IoT systems.

3.1 Introduction

This section emphasizes the motivation behind the use case, reviews the work on this use case along the build-up of our deliverables. It also provides a detailed description on our contributions and cooperation during the development process of the use case.

3.1.1 Motivation

In recent years, DC distribution systems have gained prominence over AC (alternating current) distribution systems due to the improved efficiency and the cost-effective integration of energy storage devices [12]. This transition holds significance in renewable energy production, the ICT-Industry (Information and Communication Technology), and the electro-mobility market, marking a shift toward DC grids for energy production, consumption, and distribution [13].

Arc fault detection in DC distribution systems poses significant challenges compared to AC systems. While abnormal behaviors in AC systems, such as the recognizable "flat shoulder" at zero-crossing, facilitate easier detection, the complexity arises in detecting series arc faults in DC systems [12]. Unlike parallel arc faults (short circuits) or ground arc faults that trigger circuit protection due to a high current flow, series arc faults can be easily overlooked as their current fluctuations might blend with system noise.

Conventional methods for detecting these faults often prove low adaptability in real scenarios and requires more work for data analysis ahead. On the other hand, the implementation of AI-based methods in arc fault detection presents a promising advancement. The integration of AI algorithm posts another challenge: deep learning algorithms requires usually high computational power and thus needs longer runtime.

The VEDLIoT project's methodology offers valuable tools for deploying deep learning algorithms in industrial scenarios, meeting high precision, time, and adaptability requirements. Our arc fault detection system aims to provide an AI-based (artificial intelligence) solution for series arc detection in low voltage direct current (LVDC) systems with high accuracy in real-time, addressing the unique challenges within this domain where mature solutions or products in the market are currently lacking: to ensure the detection accuracy and the critical requirements on the runtime at the same time.

3.1.2 Development milestones

Figure 19: Development Milestones for arc fault detection

Figure 19 shows milestones reached within related deliverables. On the first stage Deliverable 7.2, the first model based on historical data was built. The models were fully connected neural networks and can reach 97% accuracy on the historical AC dataset and 95% on historical DC dataset respectively. Besides, for the improvement of the project, a testbench is designed and built based on the standard UL1699B [14], the testbench is capable for arc generation and current data sampling with our own designed ADC board (refer to D7.2). The runtime was not evaluated at this stage because the availability and delivery of planed AI accelerator for the use case was delayed.

On the second stage, whose achievements were summarised in D7.3, the testbench for arc generation was put into operation and the first datasets were collected. Besides, the model was updated on this stage with the new dataset and the performance can reach 97% on this dataset. However, the dataset lacked of variety and thus the testbench was expended with extra components to increase the variety of the dataset. At the meantime, the accelerator was integrated and the initial evaluation of the system regarding runtime was conducted.

The final stage presented in this chapter further improves the testbench setup, especially in the respect of its safety and integrity. Besides, the goal of a real-time detection of DC arc fault was reached. An initial test on the runtime reports an average delay time of 13 ms for the detection algorithm, in which 11ms is contributed by the model inference. This is already close to the requirements. With further iteration on the project (procedure optimised with University Gothenburg) and compression of the deep learning model (in cooperation with Embedl [15]).

3.1.3 System design

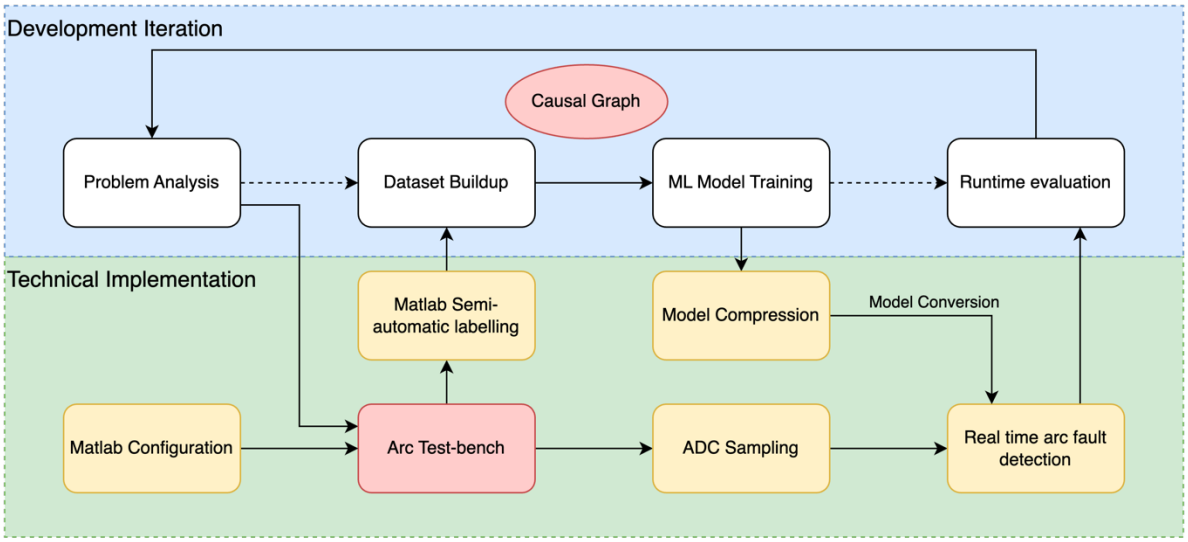


Figure 20: Development and system overview for AI-based DC series arc fault detection

The development on the use case focuses not only the improvement of hardware and software, but also the optimisation of the development procedure. Figure 20 shows the development iteration we took and technical implementation.

- Development iteration
  - **Problem analysis:** Problem analysis is the step where the problem is inspected, and the requirements are reviewed for a more standardized and efficient development procedure. It is conducted from aspects of machine learning models, data requirements and hazard analysis of the application scenario.
  - **Dataset buildup:** Dataset buildup this is the step where we collect data, based on the analysis result.
  - **ML model training:** the models with different hyperparameters are trained and evaluated.
  - **Runtime evaluation:** The model is converted to onnx and implemented for evaluation of its accuracy and runtime.
- Technical implementation
  - **Arc testbench:** The arc testbench is a prototype that we built for DC series arc generation. It also serves as the scenarios simulation for the test of real time detection system.
  - **Matlab configuration:** The generation of arc fault on the testbench is controlled by a Matlab application. The programmable components in the arc testbench can be configured to work in different modes.
  - **Matlab Semi-automatic labelling:** Matlab application that can import, visualize and label data according to user's input.
  - **ADC sampling:** the ADC board is developed, improved, and implemented for high quality data collection.
  - **Model Compression:** the model is compressed for size reduction and furthermore runtime optimisation on the hardware.
  - **Real time arc fault detection:** the inference of deep learning model

3.2 Development Procedure

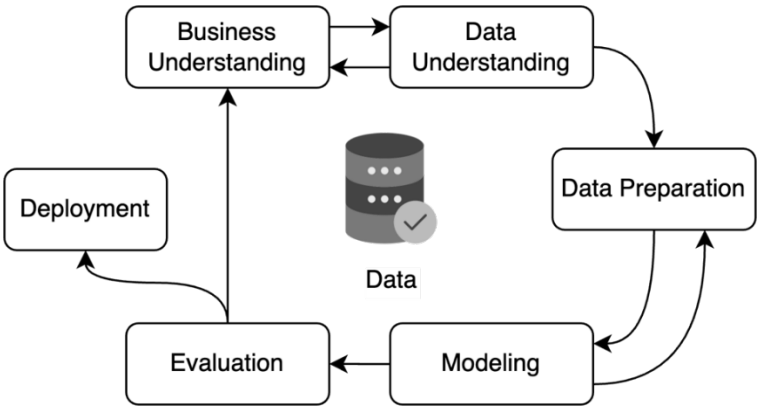


Figure 21: CRISP-DM model for data science process (Based on [16])

A systematic development procedure is important for product development and software iteration. It provides a standardized method for communication, documentation, and system design. The development of the arc fault detection system follows the standard process CRISP-DM (Cross industry standard process for data mining) for data science process, as depicted in Figure 21. This is a development step that was underestimated at the beginning of the development process of our use case.

The requirements were vagueand the goal was not discussed thoroughly with respect to feasibility. With the help from the University of Gothenburg, we reviewed each step from the procedure in our development and improved iteratively, which is similar to the procedure describe in the other use case in section 2.4.5. Besides, during the cooperation, drawbacks of convention method for software development is spotted and a method as extension for the development of AI-based software is proposed.

- **Business understanding**: we examined again related studies and products, compared them with our project requirements and the development status in between. The development direction is aligned, and the iteration is boosted.
- **Data understanding**: we conducted data analysis before feeding into model training. The problem itself is better understood and improvement on the detection algorithm is proposed for better results.
- **Data preparation**: a massive data collection is conducted with various configurations of the arc generation circuit.
- **Modelling**: the results from the step data understanding are implemented in the algorithm design and the model training procedure. The trained model is further validated on new data and optimized with pruning techniques.
- **Evaluation**: the algorithm is first evaluated on a test dataset, and then tested on the detection system prototype on Nvidia Jetson Xavier NX [17] for real time arc fault detection. Furthermore, the algorithm is also implemented on the FPGA board Ultra96-V2 and evaluation of its performance is conducted based on the test dataset.
- **Deployment**: the final deployment in real scenarios still requires further iterations of procedure above.

3.2.1 Problem analysis

This step is conducted together with the University of Gothenburg where multiple workshops are held under the given framework. The workshop first focuses on the use case definition, requirement definition and hazard analysis.

An agreement on the definition of use cases regarding various criteria on the product performance does not exist as a universal standard. A suggested performance criteria is the response time for DC system of different level is suggested in UL1699B [14]. The most critical situation is the limit of 0.8 s response time: a DC system with 900 W power, max current of 14 A and an arc distance of 6.4 mm. Since the response time consists of the time for data transmission from the sensor to the edge device, the detection time, the actuator signal transmission time and the time for actuator reaction. The system is optimised to keep the detection time as short as possible, but at least below the set goal in Deliverable 2.1 [7] of 10 ms.

Another prerequisite is that the current signal should serve as sole data source for arc classification in the testbench. This is also the most used feature in studies on DC series arc fault detection [12] [18]. There are several reasons for this decision First, the installation of current sensor is often mandatory for safety and energy efficiency reasons. Compared to voltage monitoring, the mounting of current transducers does not require extra circuit intervention. In addition, current signal sampling is less susceptible to environmental disturbances compared to other sensors such as temperature, optical or acoustic sensors. Other than this, the current change on one point can reflect the status of the whole circuit.

3.2.2 Hazard analysis

This step aims to determine the evaluation metric for machine learning model. Through analysis of the application requirements and the expected system behavior, risks are listed in Table 3 and mapped to three categories shown in Figure 22. The detection procedure is decomposed to small parts and risks. In every step the harzard is evaluated based on the likelihood of the hazards and their severity in results. Green indicates acceptable risks, medium risks are marked with yellow, and red refers to high risks.

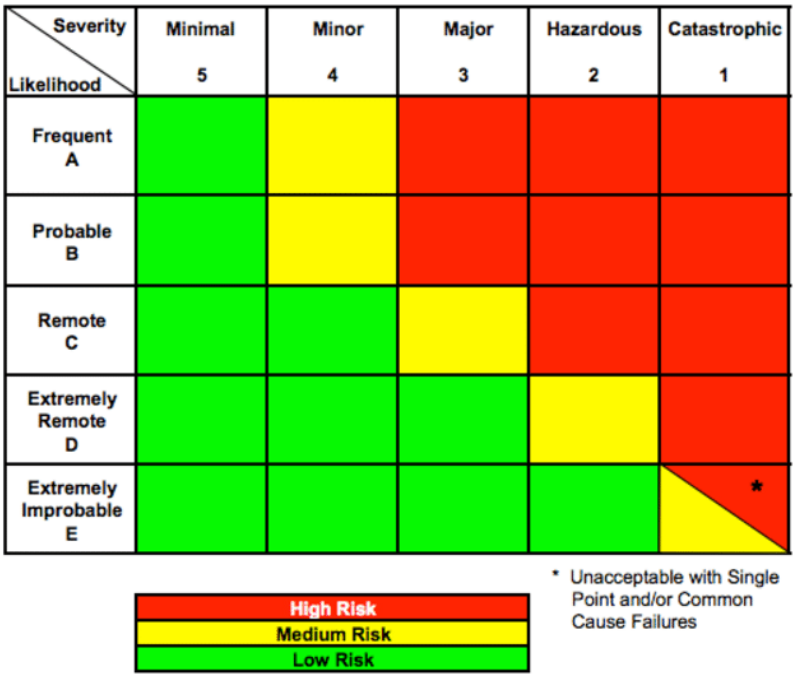


Figure 22: Matrix for risk evaluation [19]

Table 3: Hazard analysis of arc fault detection use case

Task	Subtask	Hazard	Current severity	Current likelihood	Current risk level	Recommended controls	Future severity	Future likelihood	Future risk level
DC arc fault detection algorithm based on current signal (Algorithm)	Data preprocessing	Abnormal results from preprocessing due to unexpected current data as input	4	D	Low	Improvement on normalisation			
	Detect current with high frequency in the circuit	false positive -- Noise from load such as inverter recognized as arc	3	C	Medium	Import different kinds of data with all situations covered Combine different methods – FFT + DL, ensemble DL Consecutive inference to increase trustability	3	D	Low
		false negative -- Noise from arc not recognized because of noise from inverters	2	C	High		2	D	Medium
	Detect sudden current drop in the circuit	False positive – due to load drop caused by load disconnection	3	D	Low		4	D	Low
		False negative – arc not recognized due to not enough current drop	2	D	Medium		3	D	Low
	Distinguish pattern from noise	False positive	3	D	Low		4	D	Low
		False negative	2	D	Medium		3	D	Low
	Model inference within time	Model inference (including data transmission) over 10ms, data overlapping due to delay on inference	4	B	Medium	Hardware and software optimisation	4	D	Low
	Verify correct processing (ML inference)	Cannot detect incorrect inference output	3	E	Low				
Data Collection on ADC	Collect precise current value	Data incorrect due to single-bit error, out of sync or radiation effects	3 (4)	E	Low				
Data communication	Send data to accelerator through ethernet cable	Prediction in certain time window skipped due to data overlapping or missing data	5	C	Low				
Actuator	Actuators switch off in time	Actuators operate too late – over 2 seconds	3	D	Low				
		Actuator not operating – does not switch off	2	E	Low				
	Control signal	Signal arrives too late	3	E	Low				
		Signal fails to control	2	E	Low				

The analysis shown in Table 3 aims at the algorithm because it is the focus of the project. Therefore, only recommendations for the detection algorithm is listed out. They are effective measures that help improve the detection system.

3.2.3 Causality Model

Several workshops with the University of Gothenburg for the analysis of the causality is of the DC series arc fault were conducted .

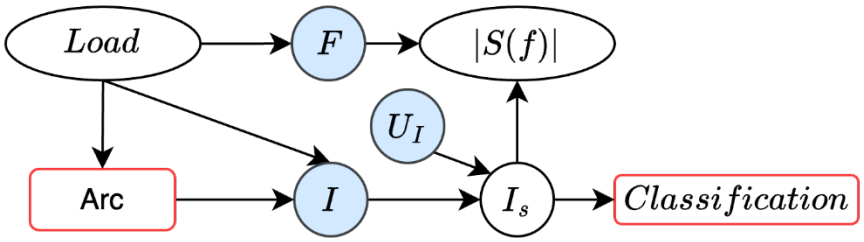


Figure 23: Causal model for arc fault detection

The result of the analysis is shown in Figure 23. From the result of the causality analysis can be seen that some key parameters have a major influence on the classification result: I is the current in the circuit, I_s is the measured current from the sensor, U_I means the characteristic noise; Load represents the system dynamic, F is the frequency component that can partially reflect system dynamic, and $|S(f)|$ is the frequency domain information that can be conducted from the measurement data of the current I_s. The requirements and specifications derived from the graph are:

- The current measurement should be augmented with characteristic sensor noise.
- The testbench setup should be able to include different components for various circuit dynamics.
- The testbench circuit should contain active switching components to generate high frequency pattern on the current.
- The frequency information should be used as additional input feature in ML model.

3.3 Implementation

This section provides technical details in implementation of the detection system, including the testbench setup, all software components, the results from system improvement and software optimisation steps.

3.3.1 Testbench

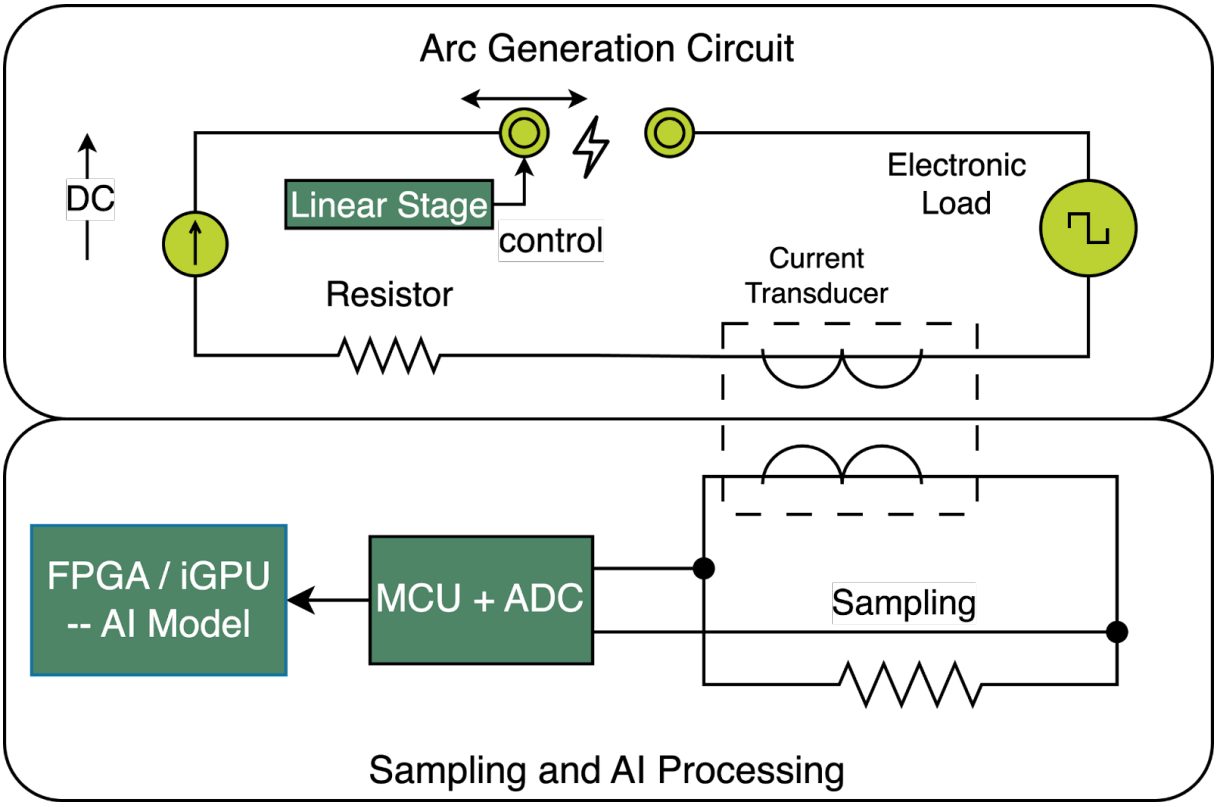


Figure 24: Final testbench setup for arc fault detection

The whole system depicted in Figure 24 consists of two parts, the arc generation circuit part and the detection system part. The arc generation circuit has a DC power supply, an electronic load, a passive load, and a pair of electrodes that are connected during normal operation. One of the electrodes is controlled by a linear stage and can create an air gap between the copper electrodes for arc generation. This is a good simulation for a DC serial arc, because those are often caused by a crack in a copper wire. In the detection system, the output of current transducer, which reduces the current proportionally is sampled by the ADC. The data is then transmitted to an edge device for signal processing with machine learning algorithms.

Component description:

- Power supply provides a stable DC voltage up to 100 V.
- Programmable electronic load can simulate different load behavior by changing its impedance. The load can operate in different operational modes.
- Programmable linear stage allows the adjustment of the air gap between the electrodes, where the arc is generated.
- Passive loads e.g. resistor and/or conductor.
- Current transducer transforms the current signal by a fixed ratio. The representing voltage signal can be measured with a shunt resistor.
- Anti-aliasing low pass filter with a cut-off frequency of 160 kHz.
- Analogue-Digital-Converter (ADC) converts data at a sampling rate of 16 kS/s (the sigma delta ADC ADS131A02 in use has a maximum sampling rate of 128 kS/s and maximum resolution of 24bit).
- AI accelerator is utilised for data processing and execution of machine learning algorithms.

The testbench has been updated since Deliverable 7.3. We rebuilt the entire setup by placing all components in a 19-inch rack and splitting it up into different, self-contained modules. The safety of the testbench was improved by grounding every component, securing critical components such as the blank electrodes from unintentional touching and moving sensitive components such as the ADC-Board into a hard case. Another benefit of the new setup is the improved mobility and thus capability to demonstrate the experiment to an audience outside the lab.

3.3.2 Arc generation and data collection

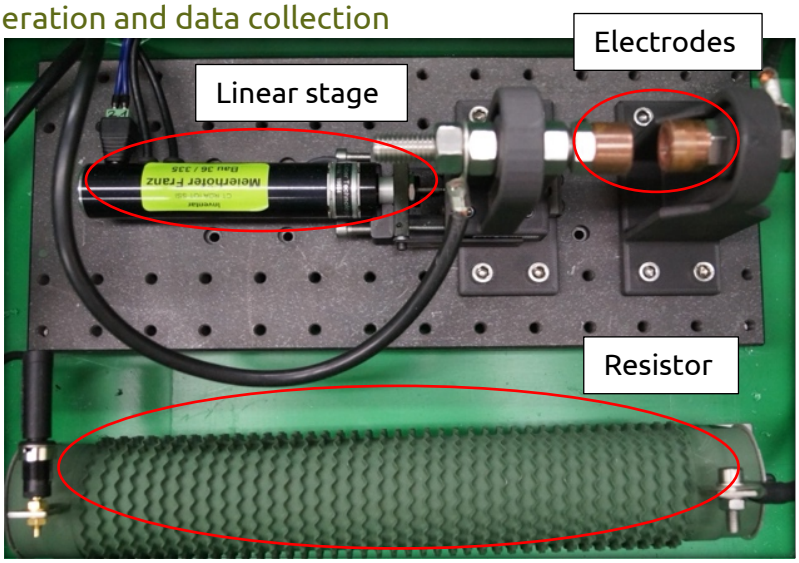


Figure 25: Electrodes for arc generation

For arc generation, the circuit topology has been updated. More recently we removed the parallel 1 Ohm resistors to achieve a higher influence of the variable load on the measured data and additionally collected data with the remaining three variable load modes: “CV – Constant Voltage”, “CC – Constant Current” and “CP – Constant Power”. The variable load is introduced in Deliverable 7.3. A load pattern extracted from a lab application is recorded and repeated in the arc generation. Along the development, more experiments and different modes of electronic load are implemented. In D7.3 we added an electronic variable load to the setup, which we have been using since. We added it in parallel to two 1 Ohm Resistors and ran it in CR mode, with a repeating 100-Value sequence ranging from 0.05 to 30 Ohm, simulating the Load Pattern of one of our lab-power-supplies.

For further development, the load pattern is simulated under CC mode, where current in the circuit is regulated to be on a stable level as shown in Figure 26.

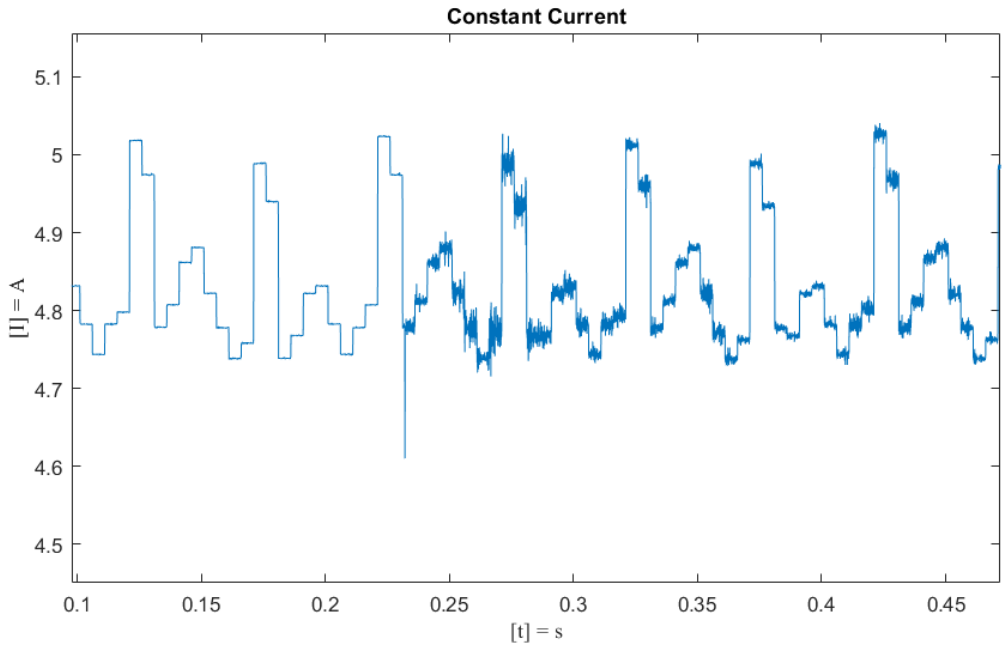


Figure 26: Arc occurrence from 0.23s on with load pattern simulated under CC mode of electronic load

3.3.3 Analogue Digital Converter (ADC) for data collection

The sensor for data collection in the detection is crucial as it determines the data quality, for this use case, an embedded system for sensing is developed in aspect of hardware and software. The initial version of ADC board is shown in Deliverable 7.2, where the ADC component behaves as an add-on function on the micro-controller. The board is then further improved and the ADC component is integrated with the microcontroller on one board as depicted. The hardware design overview is shown in Figure 27.

The integration accelerates the data transmission between the micro-controller and the ADC, thus allows a more precise data sampling and a higher sampling rate.

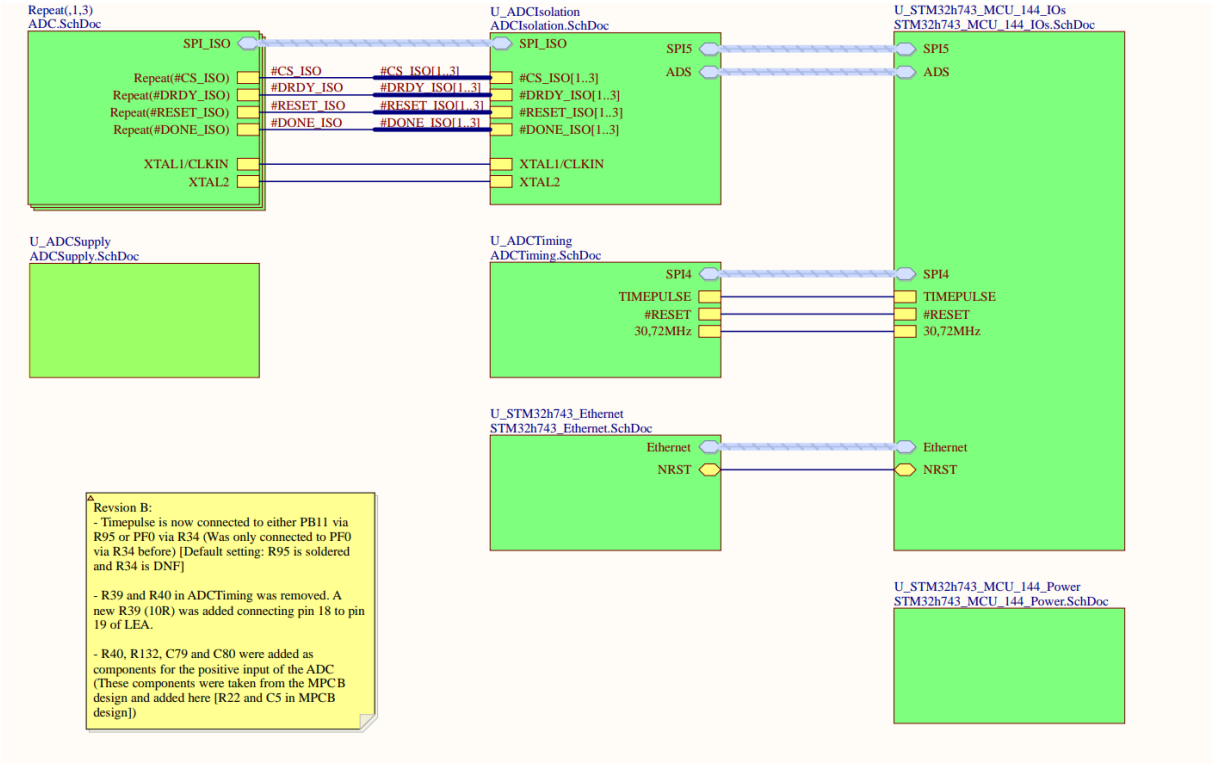


Figure 27: Circuit design with ADC board integrated with the micro-controller

3.3.4 Data labelling

With the testbench for arc generation and the sensor system for data collection, the next step is the data labelling. The amount of data is large and manual labelling consumes too much time and could lead with a certain probability to mislabeled data. For conveniences, a semi-auto labelling application is designed for an fast and accurate labelling of the collected data. This application is written in Matlab and its workflow is shown in Figure 28.

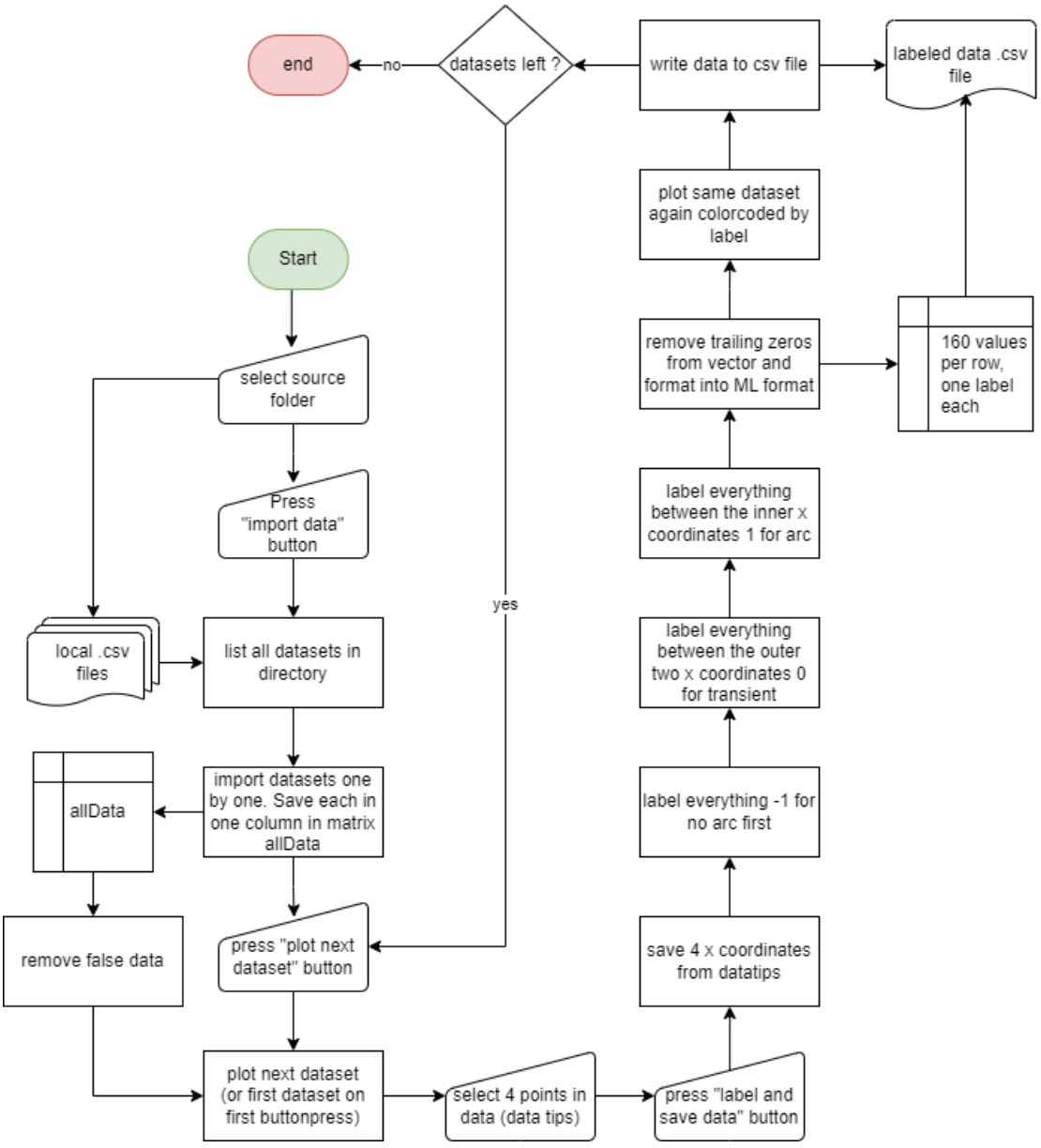


Figure 28: Semi-Auto labelling application workflow

1. The user imports data to the application.
2. The data is filtered with certain conditions.
3. The datasets are plotted in an interactive window as shown in Figure 29.
4. The users can select key turning points on the plot those four marked coordinates in Figure 29.
5. The corresponding data sequences are labelled, and the result is shown in Figure 30.

The labels have a high level of quality and reliability, as the user can always verify and, if necessary, modify the labelling of the dataset.

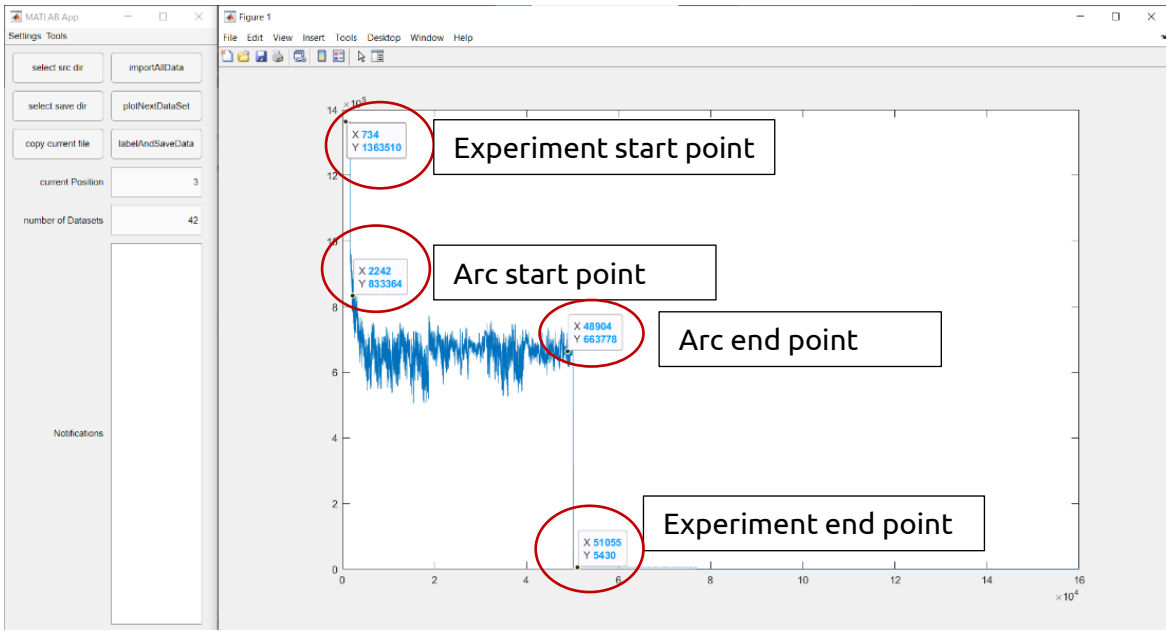


Figure 29: Selected datapoints in labelling software

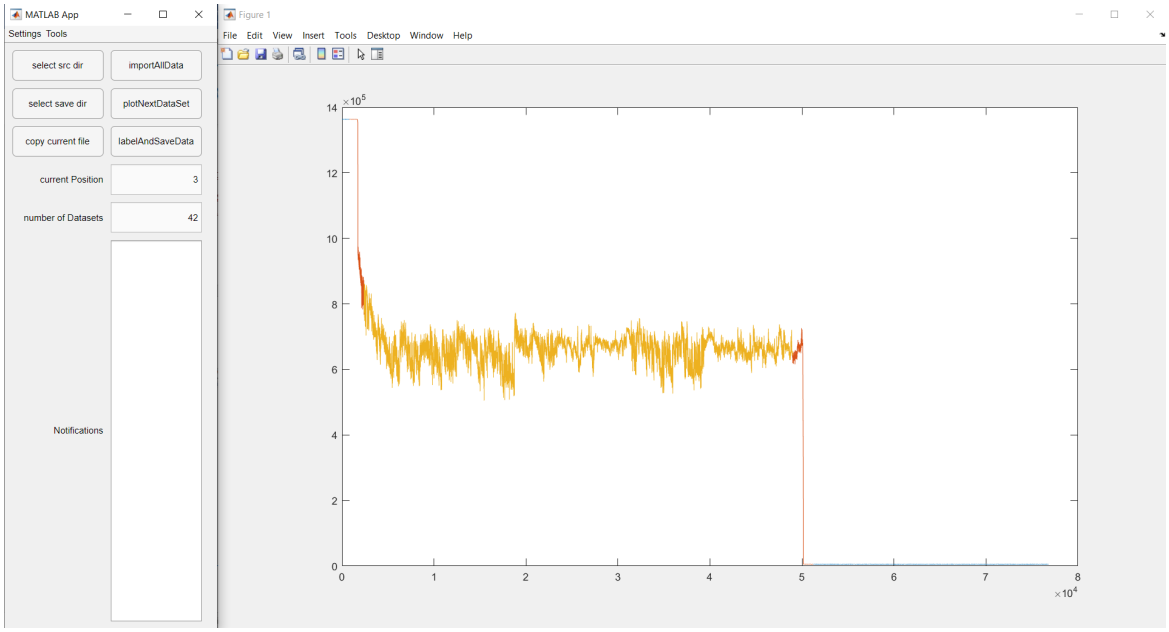


Figure 30: Datapoints after removing trailing zeros and labeling dataset. Yellow marks "arc", red marks "transient" and the rest are "no arc"

3.3.5 Model Training

The Model updated when new data were available. The initial model at the beginning of the project was trained with data that had already been available due to prior research activities. This models reached an accuracy of 95% on the test data (refer to Deliverable 7.2). The sampling rate of this dataset is 250kHz and the test data has similar data distribution as the training data. The first model a five-layers fully connected neural network (FCNN) was not tested in a real time implementation. During the further development, the model was updated with the dataset collected from the testbench. This model in D7.3 inherited the structure of previous model, thus had initially a large size and leads to longer runtime. Based on the research and the problem analysis, frequency feature extracted from Fast Fourier Transform (FFT) is introduced in as an extra input for the model. In this way, the model is reduced to a 3-layer network while keeping the model accuracy. In the latest iteration, a CNN

is also trained to a high accuracy. The advantage of CNNs compared to FCNNs is the weight sharing, which requires less memory and saves inference time. Figure 31 and Figure 32 shows the training history of FCNN (or FNN) and 1D-CNN respectively.

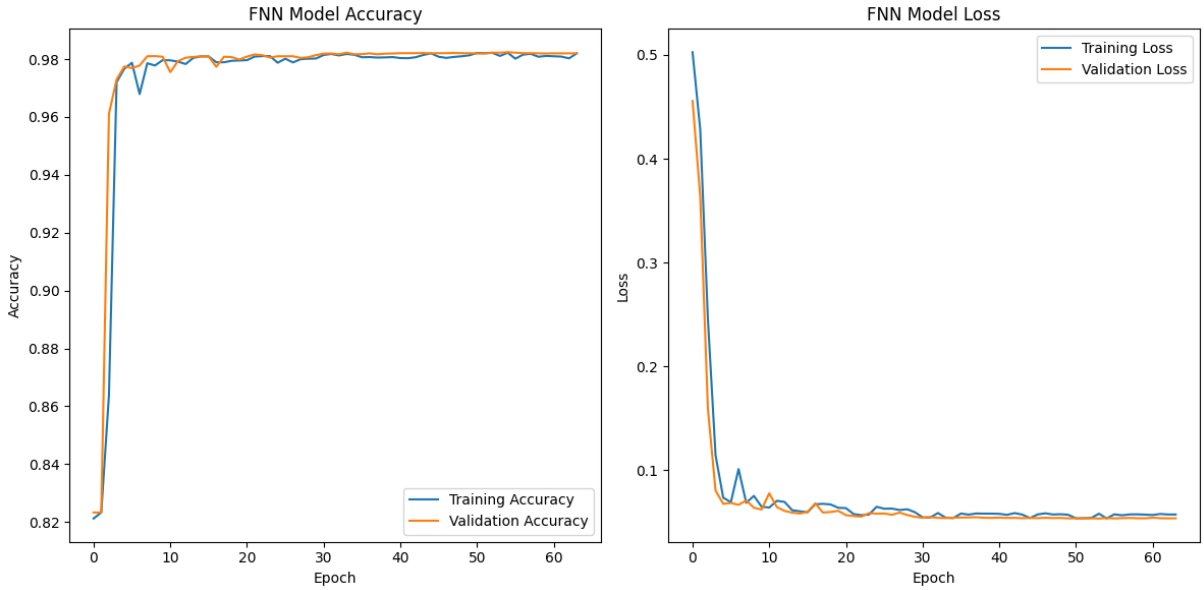


Figure 31: Fully connected neural network after tuning – training history

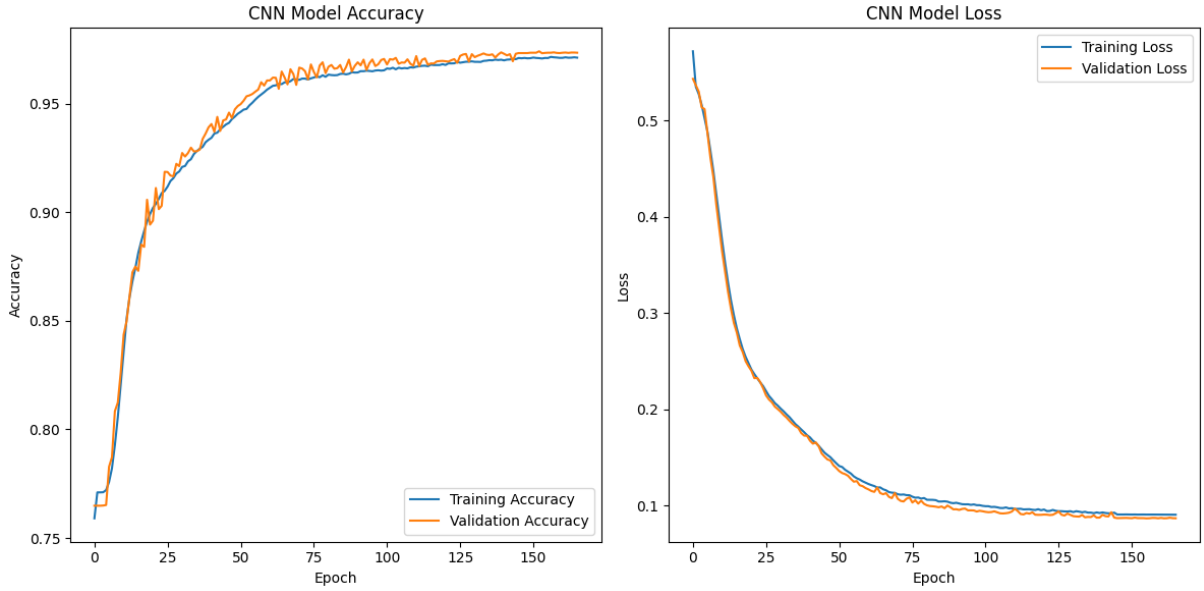


Figure 32: 1-D Convolutional neural network - training history

Table 4 summarises the evolution process of the deep learning model for arc fault detection. The model has been improved in size and runtime along the optimization process. Even though the accuracy of the current model is not largely improved compared to the initial model, they have better robustness compared to previous model as more data with extra circuit dynamics are included.

Besides, the latest model utilizes a lower sampling rate, thus the algorithm is more friendly for data transmission in real application compared to the former models. It is also notable that after the large size model is reduced to smaller structure due to FFT, there is still room for pruning. This proves that the pruning tool from Embedl can efficiently reduce the size even for a relatively small model. Detailed pruning evaluation is presented in section 3.3.7.

Table 4: Evolution of deep learning model for arc fault detection

	Structure	Dataset	Dataset size[1]	Inference time	Accuracy
D7.2	FNN 5 layer	Historical data	5500 (AC) 3200 (DC)	/	97% (AC) 95% (DC)
D7.3	FNN 5 layer without FFT	Simple data without circuit variation	32200 (DC)	11 ms[2]	96%
D7.5	FNN 3 layer – with FFT	Mixed[3]	60000+(DC)	724 us	98%
	FNN 3 layer (tuned and pruned)	Mixed	60000+(DC)	200 us	98%
	CNN2D	Mixed	60000+(DC)	209 us	97-98%
	CNN2D-pruned	Mixed	60000+(DC)	198 us	97-98%

3.3.6 Hardware acceleration

The hardware implementation on the Nvidia Jetson Xavier NX [17] was already introduced in Deliverable 7.3. The library used for model inference on the Nvidia board is TensorRT [20]. The simplified flow chart of the software implementation is depicted in Figure 33. Two options are provided, real time detection and evaluation of the model. The real time detection get data from the sensor through an ethernet connection, and reading from csv file provides a good tool for runtime and accuracy evaluation. The following process is the same as training with data pre-processing, including data normalisation, frequency information extraction with FFT.

Besides, the arc detection algorithm is also implemented on an FPGA board Ultra96-V2 with library STANN, in cooperation with the project partnet University of Osnabrück. The runtime and power consumption are evaluated on board. Details for the hardware deployment and evaluation results can be found in the Deliverable 3.4.

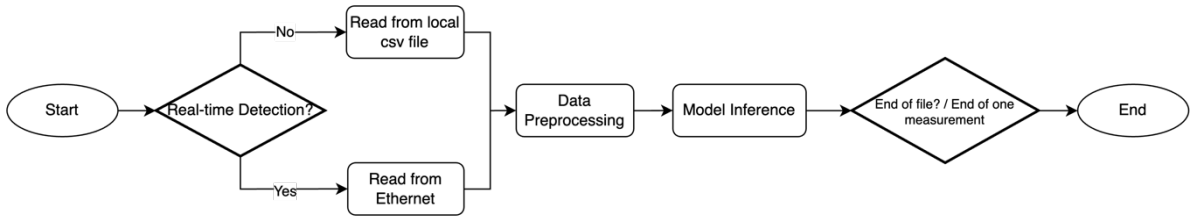


Figure 33: Simplified software flow chart on hardware

[1] One data entry represents a time series data with 160 data points, which represents current signal sampled within 10ms.
[2] This result is evaluated after the submission of D7.3.
[3] Mixed means dataset contains data from simple circuit setup without any circuit configuration and data of arc occurrence with load pattern simulation.

3.3.7 Model optimization

After the optimization of the model structure, we implemented the software package from Embedl for further model compression. This section presents our approach for the model optimisation and provides the evaluation of the software package. Table 5 and Table 6 show the optimisation results for two kinds of models, FNN and CNN2D respectively. And Figure 34 and Figure 35 show the accuracy to target curve for the pruning of both models. The target indicates the proportion of saved model parameters during pruning. The smaller the target, the smaller the pruned mode is. Uniform pruning is used here as it shows consistent results compared to other pruning methods. Uniform pruning refers to the removal of specific branches or nodes from the tree based on their importance ranking.

Table 5: Improvement of FNN model after pruning

FNN	**Nr. Neurons**	**Nr. Parameters**	**FLOPs[4]**	**Size**	**Accuracy**	**Runtime(us)[5]**
Original	500	79881	5080320	1	98.37%	231.6
Pruned	227 (-54.6%)	34115 (-57.3%)	2163904 (-57.4%)	0.43 (57%)	98.38% (-0.01%)	200.83 (-13.3%)

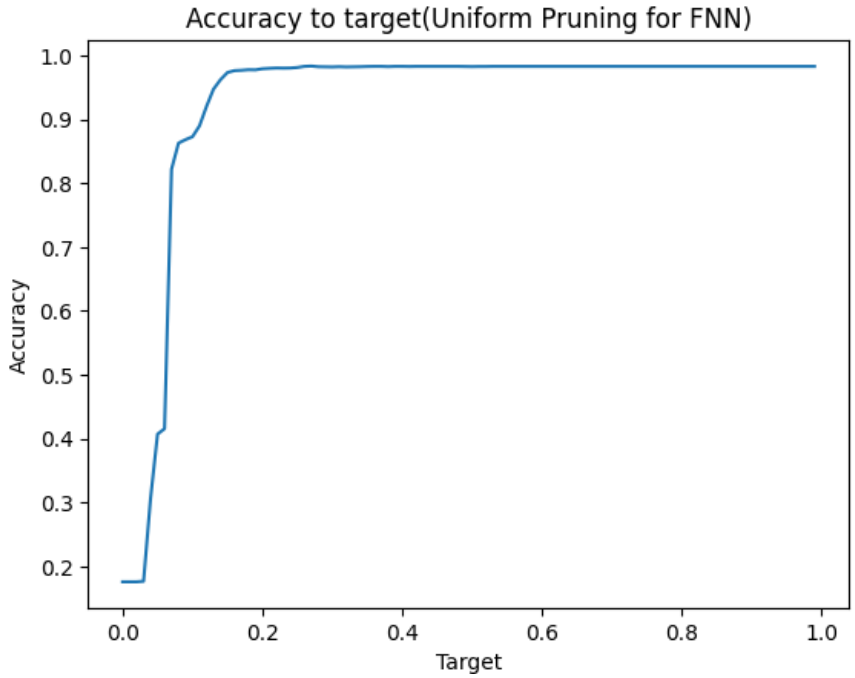


Figure 34: Accuracy to target curve on FNN model

In the context of uniform pruning for both types of modles (FNN and CNN2D), the results show a clear trend. The graphs indicates a positive correlation between the targeted retention ratio, which represents the proportion of weights and the retained biases compared to the original model's parameters, and the accuracy of the pruned model. As the retention ratio increases, the accuracy of the pruned model improves. However, once the retention ratio reaches around 0.2 for FNN and 0.5 for CNN, the accuracy stabilizes and approaches its maximum value.

Additionally, it is interesting to note that the relationship between the targeted retention ratio and accuracy seems to follow a logarithmic function. The accuracy curve gradually

[4] Floating Point Operations of the neural network, indicates the computational cost of the model.
[5] Model inference time, excluding pre-processing and memeory access

increases, reaches a plateau, and then shows minimal fluctuations around the maximum accuracy achieved by the non-pruned model. This logarithmic trend highlights the diminishing returns of preserving additional parameters beyond a certain threshold. This emphasizing the importance of carefully selecting the retention ratio to strike the right balance between model size and performance.

Table 6: Improvement of CNN2D model after pruning

CNN2D	**Nr. Neurons**	**Nr. Parameters**	**FLOPs[6]**	**Size**	**Accuracy**	**Runtime(us)[7]**
Original	128	25418	101140	1	97.35%	209.0
Pruned	60 (-53.12%)	11920 (-53.10%)	47420 (-53.72%)	0.47 (53%)	97.39% (+0.04%)	197.8 (-5.36)

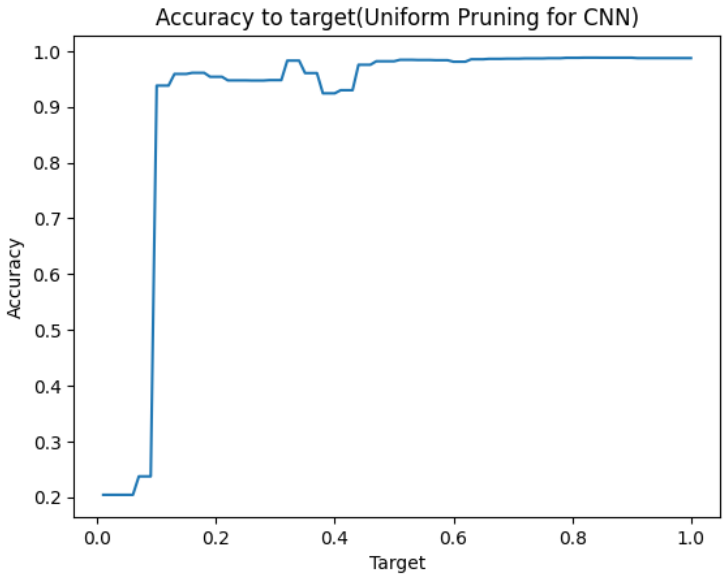


Figure 35: Accuracy to target curve on CNN2D model

3.3.8 Real-time Detection

The result for the real-time arc fault detection system is shown in Figure 36. The blue plot at the top refers to the current signal during an arc occurrence simulation. The black one in the middle is the probability output of the algorithm. And the red plot in the bottom is the final label with after-processing. To increase the reliability of the algorithm, after-processing is implemented. The idea is to raise warning only when arc is detected after three consecutive inference. In this case, the accuracy can reach almost 100%. However, the detection time is delayed to three times sampling interval, which is 30ms in our case.

[6] Floating Point Operations of the neural network, indicates the computational cost of the model.
[7] Model inference time, excluding pre-processing and memeory access

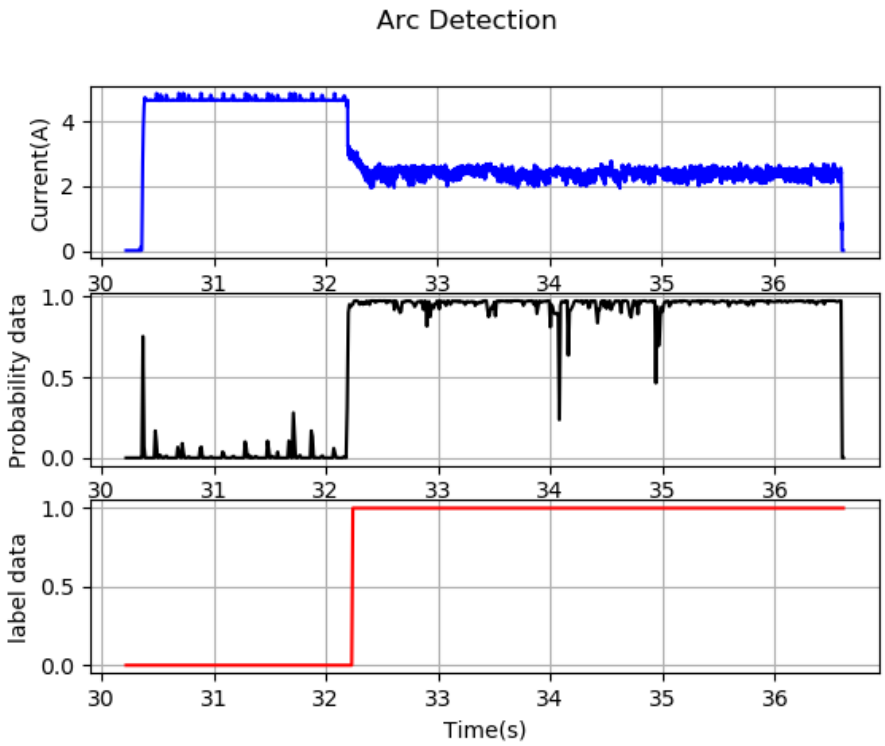


Figure 36: Result visulisation of the real-time DC serues arc fault detection system on the Nvidia board

3.4 Conclusion

The arc fault detection use case focuses on the application of DC power grids due to the growing relevance of DC power grids and lack of an cost effective DC arc detection device. For AC AFDDs, products can be found in the market, and DC AFDD is still a research field and has many open questions. The following metrics as described in Deliverable 7.2 are the benchmarking:

- Accuracy: the accuracy can reach 98% in real time evaluation amd for dataset with higher variety. With post-processing, the accuracy can reach 100%. This result achieves the set goal compared to other studies of DC series arc fault detection. The result is not compared to DC AFDDs since there is no such product in the market specializes on DC series arc fault detection.
- Execution time: around 200 us inference time on Nvidia Jetson Xavier NX after algorithm and model optimisation (refer to Table 4). Over 80% reduction on algorithm execution time (including data transmission in real time) compared to 13 ms in the initial implementation. This result meets the set requirement of 11 ms execution.
- Model size: optimized to minimal size, see section 3.3.7. Goal is reached while keep the accuracy.
- Costs: over 700 € for Nvidia Jetson Xavier NX, which is overqualified, for development purpose. This exceeds the 500 € buget set in D7.2. However, with optimised algorithm and compressed model, cheaper hardwares can be considered for this application.

3.4.1 Achievements

The demonstration of the DC series arc fault detection as a show case for the implementation of deep learning on a far edge AI accelerator for IoT systems. Throughout the development process, following achievements have been accomplished:

- Real-time DC series arc fault detection on an embedded GPU based AI accelerator.
- Collaboration with the University of Osnabrück to design and evaluate the first FPGA-based hardware accelerator for the use case.
- Evaluation of the system performance and the runtime using a Nvidia Jetson Xavier NX as the AI accelerator.
- Developing a systematic causality graph based problem analysis in cooperation with the University of Gothenburg.
- Improvement of the test bench in terms of integrity and safety.
- Expansion of the testbench to include a wider variety of circuit topologies and components.
- Improvement of the ADC board by integrating the MCU and enhancing the quality of the data (such as more accurate sampling rate and faster data transmission speed).
- Expansion of the dataset to incorporate a greater variety and higher quality of the data.
- Iterative improvement of the detection algorithm, including the optimization of the algorithm by introducing FFT in pre-processing, the optimisation of the deep learning model structure with hyperparameter tuning, and the compression of the model through pruning (in collaboration with Embedl).
- Reduction of the deep learning model (FNN) size by 96.39% in terms of number of parameters compared to the initial implementation of FNN from D7.2.
- 80% reduction of the total detection algorithm runtime including data transmission, and 98% reduction of the inference time on the hardware in total.

3.4.2 Limitations

It is also noteworthy that limitations need to be addressed for the further development:

Regarding the use case itself, various circuit dynamics, including different load profiles, voltage levels, and unexpected disturbances, should be taken into consideration to ensure a trustworthy AI-powered detection system. This poses a challenge for setting up and expanding the testbench for the experimental evaluation.

In the context of the development of the software algorithm , robustness needs to be further enhanced by implementing ensemble machine learning for post-processing. However, using the ensemble machine learning algorithm to execute DL models simultaneously on the same time series data can increase runtime. To reduce the real detection time delay caused by the post-processing implemented, reseach should also be conducted to shorten the time window for model input.

Regarding the hardware acceleration, although the model compression and algorithm optimisation have significantly reduced the runtime of the model while maintaining a high accuracy, there is still room for improvement in future development. This includes integrating more features with computationally intensive functions such as Discrete Wavelet Transformation, as well as using the ensemble of machine learning technique mentioned above.

4 Automotive AI Use Case

Light vehicles can contain a large number of Electronic Control Units, ECUs, which are performing the data processing associated with a particular function, such as a front-looking camera detecting a pedestrian. The recent trend in the premium segment is toward central processing units with significantly higher processing power that handle multiple functions in the vehicle, as opposed to satellite ECUs that solely handle the function related to the connected sensor. The centralized compute bears a high cost to the vehicle which in turn makes the associated safety benefits inaccessible to the lower-end segments.

The aim of the automotive AI use case has been to explore how computational load can be distributed between local, e.g., hardware physically located within the vehicle, and remote computational resources, which resides in external infrastructures such as base stations. The use case relates to an existing feature for vehicle safety, Pedestrian Automatic Emergency braking (Pedestrian AEB). It is described by, among others, the Euro NCAP organisation in [21]. However, in this implementation, the intent has not been to design a function that can be used in a safety-critical setting, but to explore the computational properties.

This section describes the developments and tests that were performed at the Magna test site in Vårgårda within the project's lifetime and includes results for the KPIs latency and power consumption when using the ML model used with computational power onboard a vehicle and at a base station. The results are analyzed, discussed and conclusions are presented.

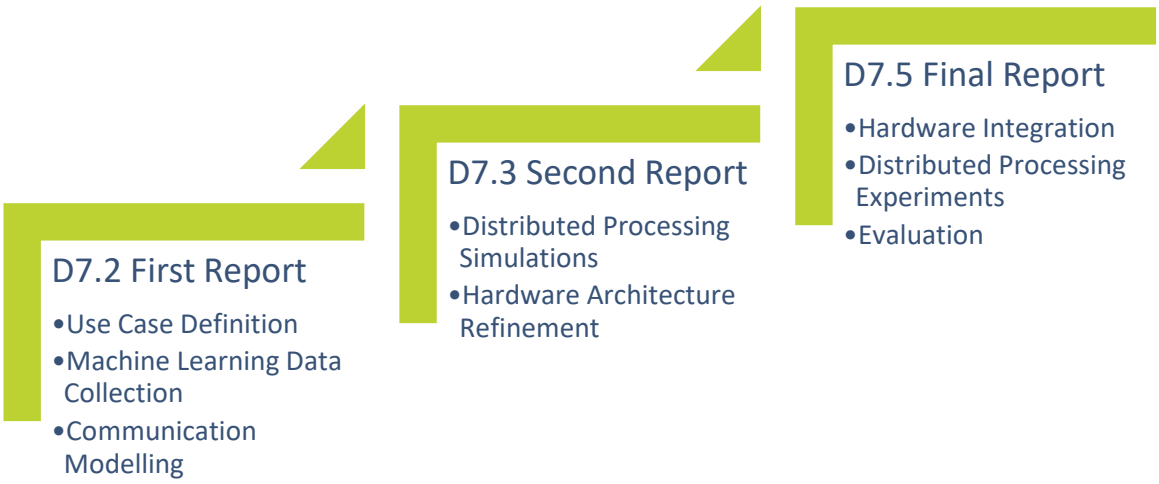


Figure 37. Development process of automotive use case presented in deliverables

4.1 Development of automotive AI model

This use case has been defined to explore how a machine learning inference can be distributed over multiple processing nodes.

4.1.1 Machine learning training data

Scenario data for the use-case was collected at the Magna test site the airfield in Vårgårda, see Figure 38. The use case assumes an open road with a pedestrian or possibly another object on the scene.

Figure 38: Bird's eye view of the data collection environment at the Magna test site in Vårgårda.

The following scenario and data variations were identified and collected:

- Target type
  - Pedestrian
  - Other object (trash bin)
  - No Target
- Target longitudinal distance
  - Every 1 m over 1 to 100 m (handled by continuous sampling of pictures during each drive)
- Target lateral distance
  - 1, 1.5, 2, 2.5, 3 m
- Target attitude
  - Moving towards
  - Moving away
  - Crossing
- Background type
  - Tarmac
  - Grass
- Illumination
  - Sunlight
  - Cloudy
  - Rain

The data has been collected and labelled for all the combinations as described above. Data for ten runs per combination were collected, resulting in a total of 290 data sets with continuous images. These images make up the training, test and validation data. The labelling strategy used to create labels for the data sets was to classify one of two conditions:

1. Pedestrian on the road
0. No pedestrian on the road

The data collection was performed with the dedicated data collection system depicted in Figure 39. It includes a camera for capturing images of the scene ahead of the vehicle, an RTK GPS to capture a very accurate position of the vehicle at the time of the image capture, and a PC used for managing the incoming data and saving to an external hard drive. This system was designed to collect and store the images used to learn the ML. It is different from the system to be used for validation. The validation system is described in Chapter 4.2.

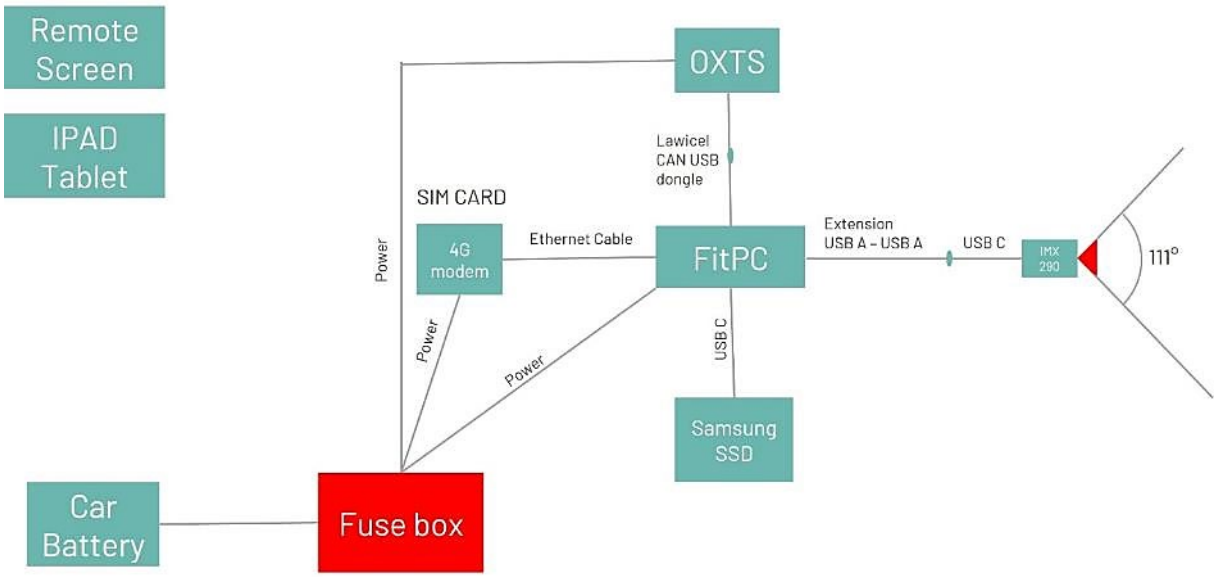


Figure 39: Overview of data collection system

Figures demonstrating each scenario can be found in the following pages, where Figure 40 demonstrates the scenario of a pedestrian moving in the same direction as the vehicle on the side of the road. The DL model will classify this scenario as “No pedestrian on road”.

Figure 41 demonstrates the scenario of a pedestrian moving towards the vehicle. The DL will classify this as “No pedestrian on road”.

Figure 42 demonstrates a pedestrian crossing the road. The DL will classify the object as “No pedestrian on road” while the pedestrian is outside of the lane markings and “Pedestrian on road” while the pedestrian is inside the lane markings.

Figure 43 demonstrates the scenario of no objects present in the scene. This will always be classified as “No pedestrian on road”.

Figure 44 demonstrates another type of object, in this case a trash bin placed at 3 different positions, present on and off the road. This will be classified as “No pedestrian on road”.

Figure 45 demonstrates the pedestrian moving away from the vehicle. This scenario is much like the one demonstrated in Figure 40 with the change in the environment around the pedestrian as she is walking on grass instead of concrete. The labelling strategy for both scenarios was the same.

The same repetition and change in the environment were made with the scenario demonstrated in Figure 41 and Figure 42.

Figure 40: Pedestrian moving away from ego vehicle

Figure 41: Pedestrian moving towards the ego vehicle

Figure 42: Pedestrian crossing the path of the ego vehicle

Figure 43: Empty scene

Figure 44: Other objects on and off the path of the ego vehicle

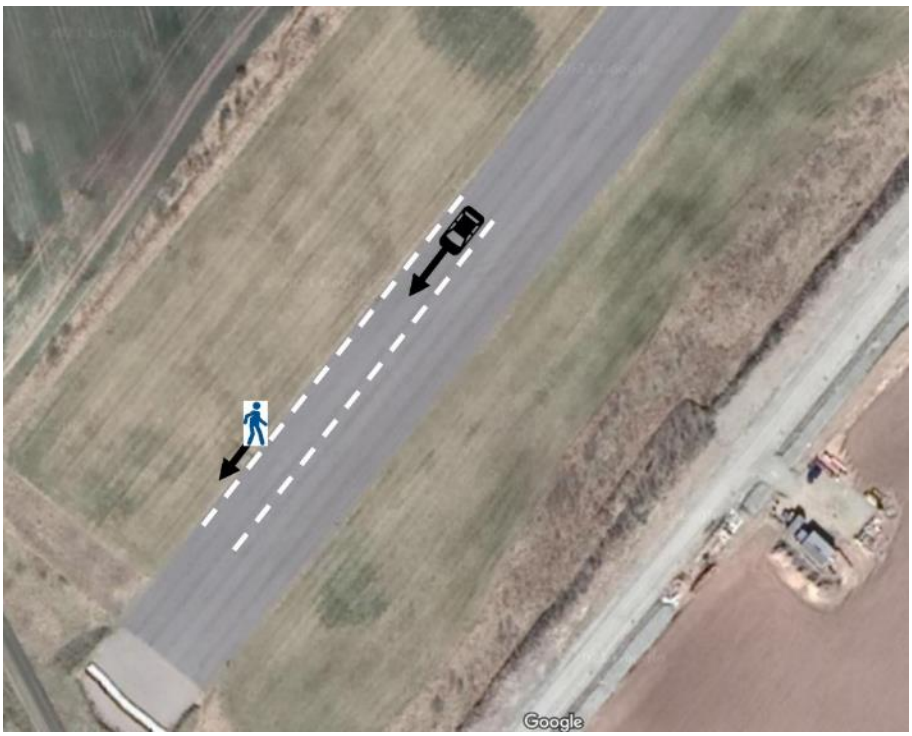


Figure 45: Pedestrian moving away from the ego vehicle on the grass

4.1.2 Machine learning model

The Convolutional Neural Network (CNN) architecture EfficientNet was used to design and train a model that can determine if there is a pedestrian on the road. Due to the serialized design of EfficientNet, it is possible to pass the execution on to a different set of nodes at a split point as the execution of an operation, or block of operations, feeds into the next block of operations. The data size varies at each split point, due to the neural network's inherent compression and decompression of the data during inference. This leads to three

dimensions that can vary: the data size, the latency when transferring data between different nodes and the computational power of the nodes. It is thus possible to perform an optimization of the entire process of detecting a pedestrian, given boundary conditions in the form of constraints in the available vehicles and infrastructure. An in-depth documentation of the development process is described in D6.5 [21].

4.1.3 Communication modelling

When distributing computations, the latency and stability of the communications become an important factor. A model, containing factors such as limited bandwidth and distance between the units, was developed and included in the ML model optimization.

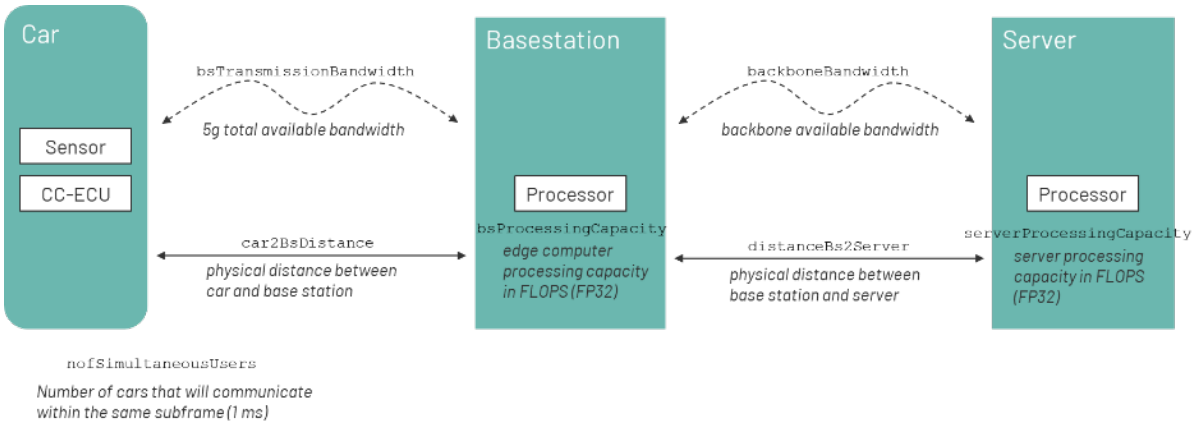


Figure 46: Model of communication in a 5G system with distributed systems

The purpose of the model is to provide reasonable effects, e.g., latency and packet loss, to the distributed DL system. The model induces bandwidth occupancy limitations due to an overload of user's connection to the base station. Other effects as for instance those induced on the signal due to distance to the base station can be managed in the model.

4.1.4 Distributed Processing Simulations

There are multiple scenarios where distributed inference is beneficial. What those benefits are, depends on one hand on the limitations of the distributed compute nodes, but also on external constraints such as data transfer speed or the number of users of a compute node. In general, the idea is to exploit differences in computational power between compute nodes by leveraging the cost of transmitting data against the performance boost of a powerful remote compute node. The objective is often to reduce the overall latency of inference, but other motivations can be driving the need for distribution, such as memory or energy restrictions of a compute node. We have started to investigate the potential of reducing the overall latency of inference.

The execution of CNNs, such as EfficientNet, is highly serialized. The execution of an operation, or block of operations, feeds into the next block of operations. The serialized execution means that the network can be "split" or divided at some point along the network. It is thus possible to perform part of the computation on one compute node and then distribute the rest of the computation to another node, which sends back the prediction. If all compute nodes have equal computational power, it would never be beneficial to distribute, as the extra latency of transmitting the output data of the first network would only increase the total inference latency. In this use case, however, the compute node at the car's sensor, the central compute node, and the compute node in the remote base station, all have different computational power. Additionally, the data size varies at each split point, as a neural network naturally compresses and decompresses the data during inference. If the data size at any potential split point in the network is smaller than the input image, it is

possible to reduce the overall latency. The potential latency reduction, however, depends on the speed at which data can be transferred between the nodes.

As a first set of experiments, we have investigated distributed inference in a setup mimicking a weak compute unit at the car's sensor, and a powerful compute unit at a remote base station. Specifically, we have considered a Raspberry Pi 4, and an NVIDIA Xavier. Ultimately, a similar analysis will be performed using the Congatec i.MX8+ device, described above, instead of the Raspberry Pi.

In Figure 47, we have calculated the total latency of EfficientNet for a number of different split points. The first part of the computation is performed on the weak Raspberry Pi, the output of that computation is fed as input to the second part of the split network. The extra latency due to data transfer and resource allocation is estimated using the MATLAB model described in section 4.4. For realistic parameters, we conclude that distributed inference has the potential to be highly beneficial. When comparing the overall latency between performing the full computation locally and choosing the optimal split, the latency is reduced from 120 ms to 77 and 66 ms in Figure 47 (a) and (b), respectively. The two parameter regimes we consider are: (a) a car 500 m from the base station, with 1 Gbit/s basestation bandwidth, no other users (cars), and (b) a car 500 m from the base station, with 5 Gbit/s base station bandwidth and 3 other users (cars).

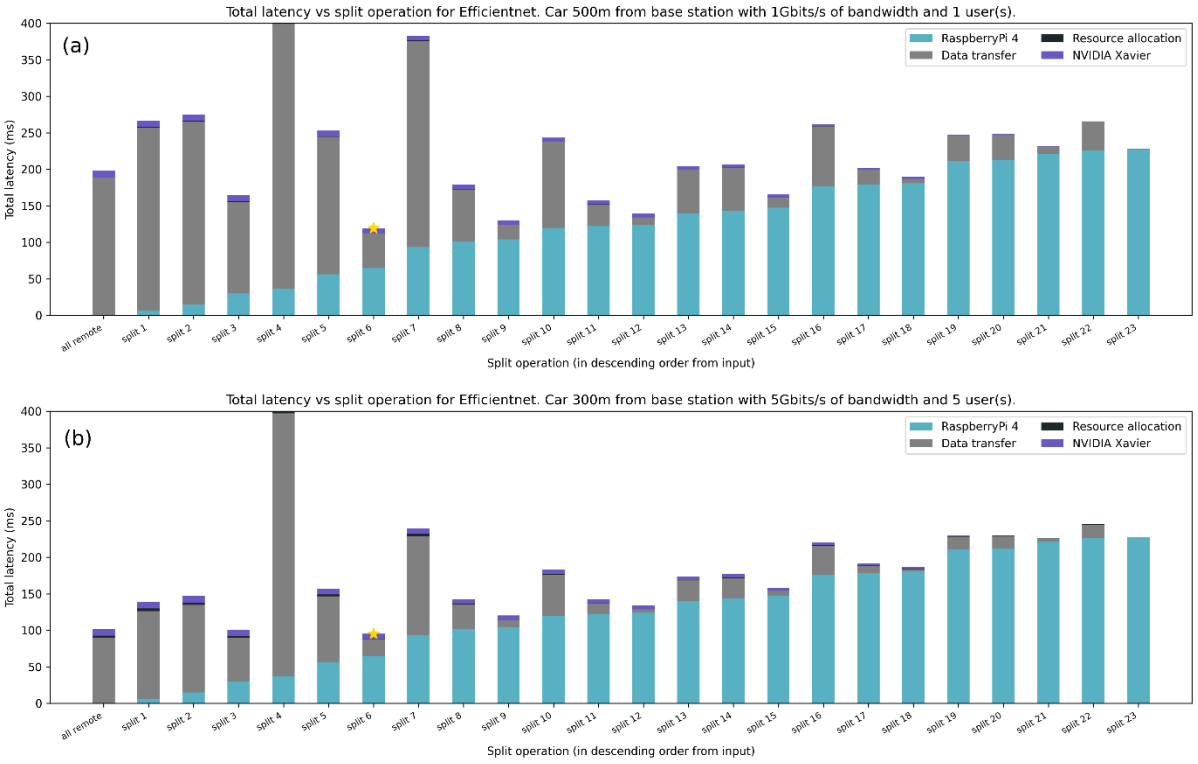


Figure 47: Simulation of total latency when splitting the neural network

4.1.5 Distributed Model Demonstrator

A distributed machine learning demonstrator was developed Embedl. The purpose of the demonstrator was to introduce the core concept of distributed processing to visitors to various faires where VEDLIoT were present. Through offline processing of selected data sets, described in Chapter 4.1.1, the resulting latency of the system was shown. The demonstrator can first be initialized with the mode of processing. The options of processing mode include local, remote or distributed. Parameters related to the data transfer are also available in the cases of remote or distributed, specifically bandwidth, occupancy and

distance. The visualization of the demonstrator is shown in Figure 48. The upper left box contains an area to display the recorded image form the camera sensor. The lower left box shows the computation latency over time, as a result of the parameters inputed in the lower right box. The upper right box illustarates the hardware involved in the computation as per the given input.

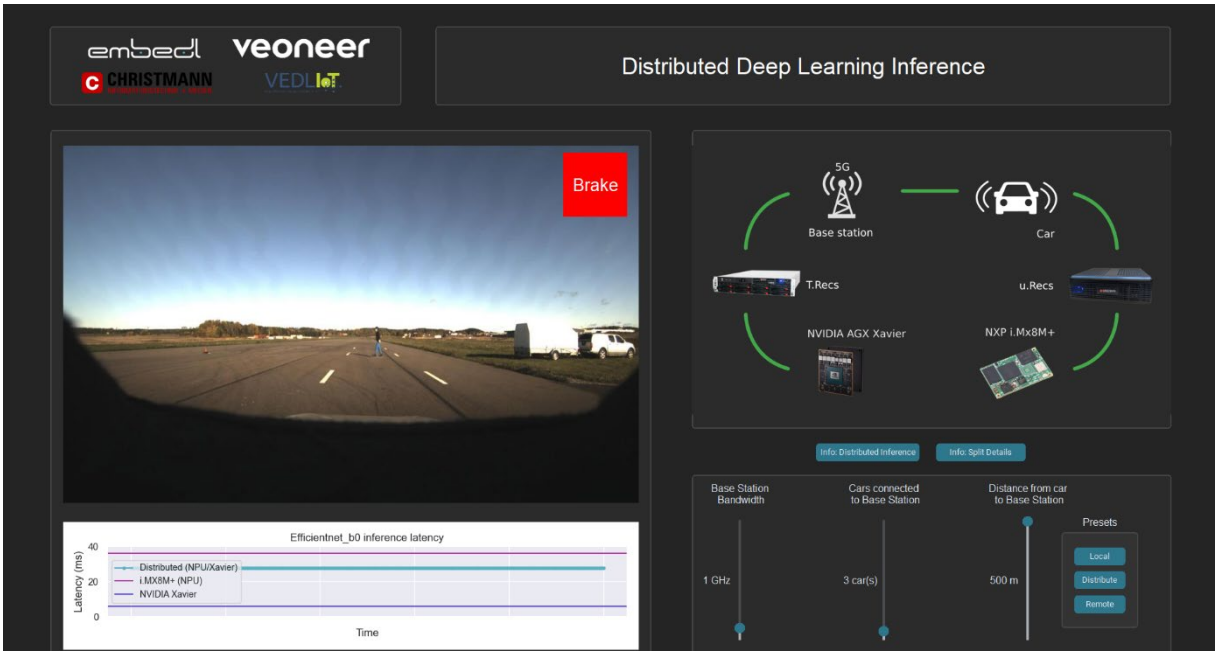


Figure 48: Distributed Processing fair demonstrator

4.2 Test Setup

In the following section, the details of the hardware in the vehicle, of the base station and the design of the experiment will be discussed. As shown in Figure 49, the vehicle contains a camera and processing hardware. It communicates with the far edge device using either a WiFi or 5G mmWave connection. No cloud instance is used in this setup, but is included in the description for completeness as a cloud instance could be a useful resource in a real-life implementation of the use case.

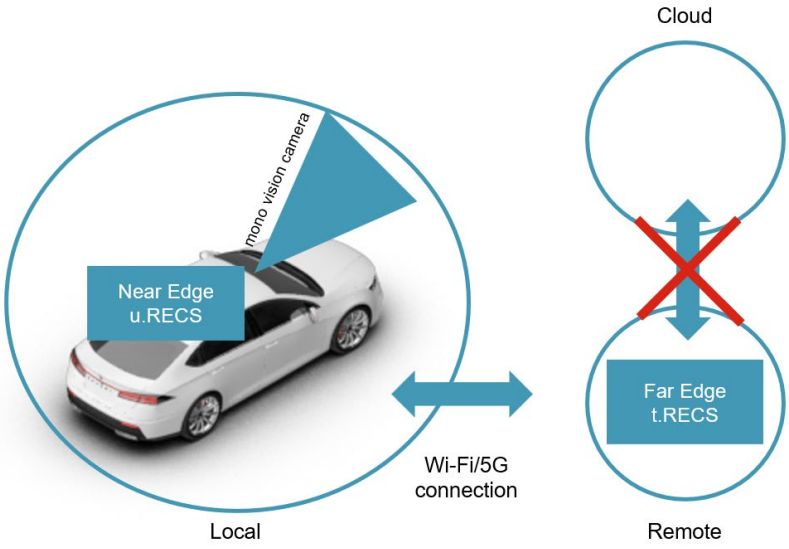


Figure 49: High level overview of a distributed AI system. The Automotive Use-Case utilizes the Local device in the vehicle and the Remote device located in the basestation. No cloud instance was used in this project.

4.2.1 Equipment, vehicle and edge

The hardware design for the automotive use case assumes four possible processing nodes. These are depicted in Figure 49 and below:

- The mono-vision camera processing device
- The vehicle central processing unit (ECU)
- The 5G base station edge processor
- The cloud server

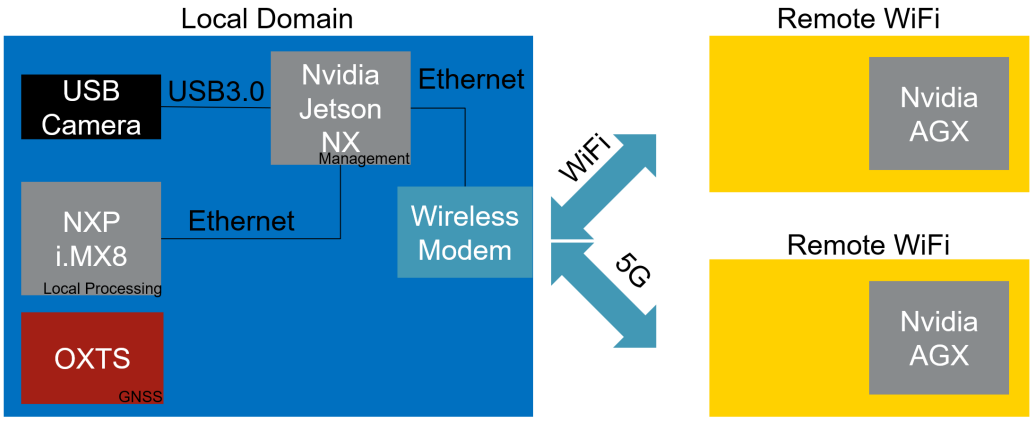


Figure 50: Hardware design used for the experiment. The blue box, the local domain, describes the hardware in the vehicle. The two yellow boxes, the remote domain, describes the same hardware placed in connection with either the 5G or WiFi infrastructure.

In the VEDLIoT project the RECS hardware platform was selected [22] to provide the hardware base for the vehicle tests: a u.RECS solution placed in the vehicle and a t.RECS placed in the base station. The u.RECS was modified to suit the needs of the automotive use case and is a product of WP4. Figure 50 illustrates the hardware components in either the local domain, e.g. in the vehicle, or in the remote domain, e.g. in the base station. The u.RECS contains both a Congatec NXP i.MX8 module, compatible with the existing camera processing device, and an Nvidia Jetson NX module which equals the central computer (ECU) available in vehicles today. The base station used a t.RECS server installed as processing unit. It contains an Nvidia Xavier AGX as the processing device.

The test vehicle was equipped with a u.RECS unit, as described above, a Wireless Modem to manage the transferring of the data from the vehicle to the remote domain, as well as an OXTS GNSS unit [23] used for logging the absolute position of the vehicle during the measurements. Figure 51 shows the installation of the hardware in the trunk of the test vehicle.

Figure 51: In-vehicle installation of u.recs, OXTS and Wireless Modem

4.2.2 EfficientNet distribution

The experiments conducted in this report describe the inference executed on the different domains, see Figure 52, e.g. the local domain or the remote domain. The third alternative is the distributed domain, where part of the inference is processed within the local domain and part of the inference is processed within the remote domain. The local and the remote versions constitutes the normal way of feeding an image to the model and thus resulting in a prediction. The distributed version instead processes a few layers of the inference within the local domain, then transfers the intermediate buffer from the local domain to the remote domain, where it's finally producing a prediction of the presence of a pedestrian on the road. The total EfficientNet model contains 20 layers, and based on previous analysis, it was decided to perform the split between local and remote processing at the 13th layer for the best performance. At the 13th layer, the intermediate representation of the input data is highly compressed, thus reducing the transmission latency at this point and minimizing the overall inference latency in most parameter regimes of interest in this study.

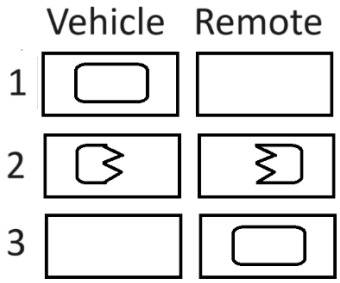


Figure 52: Local, Distributed and Remote processing. 1: The entire inference is processed in the vehicle. 2: The model resides partly in the vehicle and partly in the base station. 3: The entire inference is processed in the base station.

4.2.3 Wireless communication

To be able to test and evaluate the distributed and remote variants of the automotive use case, a dedicated wireless high throughput communication method was needed. The test facility had two technologies available; a WiFi technology at 802.11ac and 5G mmWave that was being set up during the testing period. We chose to conduct the test cases on the 802.11ac network, and as the 5G equipment became available during the test period, 5G communication was added to relevant scenes.

The WiFi technology used was standard equipment for outdoor use. The selected WiFi Access Point had a theoretical maximum range of ~300 meters. The vehicle PC network was connected to an industrial network router with a 1 Gbit/s upper limit of transfer speed was used. The uplink was configured in WiFi client mode to allow connection to the secured WiFi network established by the Access Point.

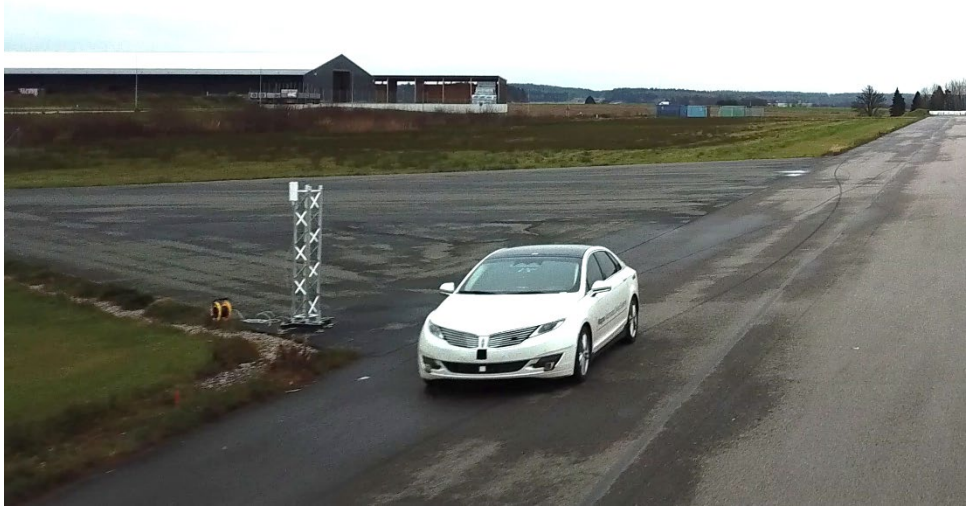


Figure 53: WiFi Access Point at pole near test location

For the use of the 5G network [24], a 5G mmWave modem was connected to the vehicle equipment in a similar fashion to the WiFi transfer configuration. 5G network was isolated and dedicated for the execution of our test cases.

While no cloud instance was used in this data collection, the 5G network on a mmWave band, supplied by Telia using Ericsson technology, allowed us to set up the far edge close to the Radio Access Network, by using a Local Breakout, allowing for a very short route for traffic between the radio transmission and thereby provide a way to access a far edge. Both technologies for data transmission between the near and the far edge provide throughput well over that which would be used for communication between the far edgeand the cloud instance, avoiding a potential bottleneck situation.

4.2.4 Design of the Experiment

The site chosen for the tests was the airfield at Vårgårda, see Figure 54. It is a private, fenced-in location where tests can be carried out in a controlled environment.

Figure 54: Overview of the positions of relevance during the experiment. Yellow boxes describes positions for the wireless medium and blue boxes describes the positions for the vehicle. A=WiFi Near, B=5G Near, C=Dynamic Start, D=WiFi Medium, E=Dynamic Stop, F=WiFi Far, W=WiFi Antenna, 5=5G Antenna

There are several possible dimensions to explore when evaluating the performance of the system. Five parameters were chosen:

1. Distance between the vehicle and the communication device
2. Where the processing was made: Locally, Remotely or Distributed between the two
3. The connection mode: WiFi or mmWave
4. Observed scene: Pedestrian, Empty, or Other object (i.e. Trash Can)
5. Vehicle velocity for 5GDynamic: 30 or 50 kph

The distances between the vehicle, communications equipment and dummy/object, depending on communication mode and vehicle dynamics: For the WiFi tests, the distances between vehicle and WiFi access point was 5, 12 and 200 m for WiFi Near, WiFi Medium and WiFi Far, respectively. For the 5G tests 5G near is at 80 m and 5G Dynamic 80 m–180 m from the base station, drivig toward the dummy. In all stationary tests, the distance to the dummy is 10 m.

Note that the antenna for the base station and the WiFi access point are not co-located and that the distances for the respective communication means are different as the WiFi network does not have the same power output or data transfer ability as the 5G network. The WiFi was positioned on the test track at position W as seen in Figure 54 in as opposed to the mounting of the 5G antennas on the roof of the control room, position 5.

A full factorial investigation of these parameters would amount to 27 scenes in total. In this report, data from 14 WiFi-related and 12 5G scenes is presented.

Table 7: Automotive AI integration scene overview

Scenario	Vehicle Position	Inference Distribution	Object in scene	Ego Vehicle Speed [kph]	Cellular Connection Type
S-1	WiFi Near	Local	Pedestrian	0	WiFi
S-2	WiFi Near	Remote	Pedestrian	0	WiFi
S-3	WiFi Near	Distributed	Pedestrian	0	WiFi
S-4	WiFi Far	Local	Pedestrian	0	WiFi
S-5	WiFi Far	Remote	Pedestrian	0	WiFi
S-6	WiFi Far	Distributed	Pedestrian	0	WiFi
S-7	WiFi Near	Local	Empty	0	WiFi
S-8	WiFi Near	Remote	Empty	0	WiFi
S-9	WiFi Near	Distributed	Empty	0	WiFi
S-10	WiFi Near	Local	Other Object	0	WiFi
S-11	WiFi Near	Remote	Other Object	0	WiFi
S-12	WiFi Near	Distributed	Other Object	0	WiFi
S-13	5G Near	Remote	Pedestrian	0	5G
S-14	5G Near	Distributed	Pedestrian	0	5G
S-15	5G Near	Remote	Empty	0	5G
S-16	5G Near	Distributed	Empty	0	5G
S-17	5G Near	Remote	Other Object	0	5G
S-18	5G Near	Distributed	Other Object	0	5G
S-19	WiFi Medium	Remote	Pedestrian	0	WiFi
S-20	WiFi Medium	Remote	Pedestrian	0	WiFi
S-21	5G Dynamic	Local	Pedestrian	30	5G
S-22	5G Dynamic	Remote	Pedestrian	30	5G
S-23	5G Dynamic	Distributed	Pedestrian	30	5G
S-24	5G Dynamic	Local	Pedestrian	50	5G
S-25	5G Dynamic	Remote	Pedestrian	50	5G
S-26	5G Dynamic	Distributed	Pedestrian	50	5G

These scenes represent a sample of parameters that can be chosen. However, the experiment also contains environmental conditions that cannot be chosen, at least not to any extent as the tests are booked well in advance.

4.2.5 Test conditions

The tests were performed at the Vårgårda airfield between October 30th and November 3rd 2023 with fair lighting conditions under daytime. In Figure 55, an overview of the test site taken at the end of a test day, is shown. The weather was overcast but not raining and the road surface was dry.

Figure 55: Overview of the Vårgårda test site during the field measurements. The conditions were cloudy with a dry to slightly wet surface during the measurements.

Although the time of the year was roughly the same during the testing and the collection of the data set used to train the machine learning model, there are some differences in the environment. The grass was greener and the color of the trees more diverse colors of yellow and red, as is typical of the Swedish autumn, during the data collection of the training data.

The positions of the test scenarios are also slightly different between the two data sets as adjustments had to be made to allow for parallel projects doing data collections during the same period of time.

Data was collected during 60 s for each static scene. For the dynamic scenes, the data collection lasted around 20 s for the 30 kph scene and 16 s when finishing at 50 kph. This is due to the fact that the vehicle started from standstill, then accelerated to the designated velocity and finally braked in time to avoid collision with the pedestrian dummy, all within a fixed distance.

4.3 Evaluation Parameters

During the data collection, four parameters were logged to be analysed afterward. These were:

- The **latency** was monitored and logged on a round-trip basis as the final measurement to be evaluated. The measurement included the time from when an image was captured until a classification of the image was made.
- The **power consumption** was measured for each used accelerator, two in the u.RECS inside the vehicle and one in the t.RECS in the base station.
- The **accuracy** (or **detection rate)** is the ratio of how often the ML model accurately classifies the scene with "Pedestrian on road" or "No pedestrian on road".
- The **robustness** is a measure of the number of packets that are lost in the communication to and back from the base station.

The KPIs for this use case are

- pedestrian detection accuracy > 95%
- detection latency (per image) < 20 ms

As the use case is designed to test whether it is possible to distribute the calculations between different nodes, no pass/fail limits with respect to power consumption or robustness are given.

4.4 Results

In this section, an overview of the results will be shown, and then more detailed results relating to different clusters of measurement cases will be presented where appropriate. The latency and energy consumption are shown for each scene, then follows the accuracy of the model and finally the robustness in terms of the ratio of packets delivered and lost in each scene.

4.4.1 Energy Consumption

Data for the power consumption and latency is shown using box-and-whiskers plots, where the box covers the 25th to 75th percentiles, the orange line represents the median value, the whiskers end at the farthest data point lying within 1.5 times the inter-quartile range, i.e., the height of the box, from the box. The circles represent outliers that have values which fall outside of the whisker ends.

In Figure 56, the total power consumption is shown. The locally performed computations in the (blue) require the least amount of power, the remote (red) a higher amount and the distributed (orange) some linear combination of the two. However, the values presented concerns the entire computational capacity of the far edge, and the power actually consumed due to the inference is likely significantly smaller.

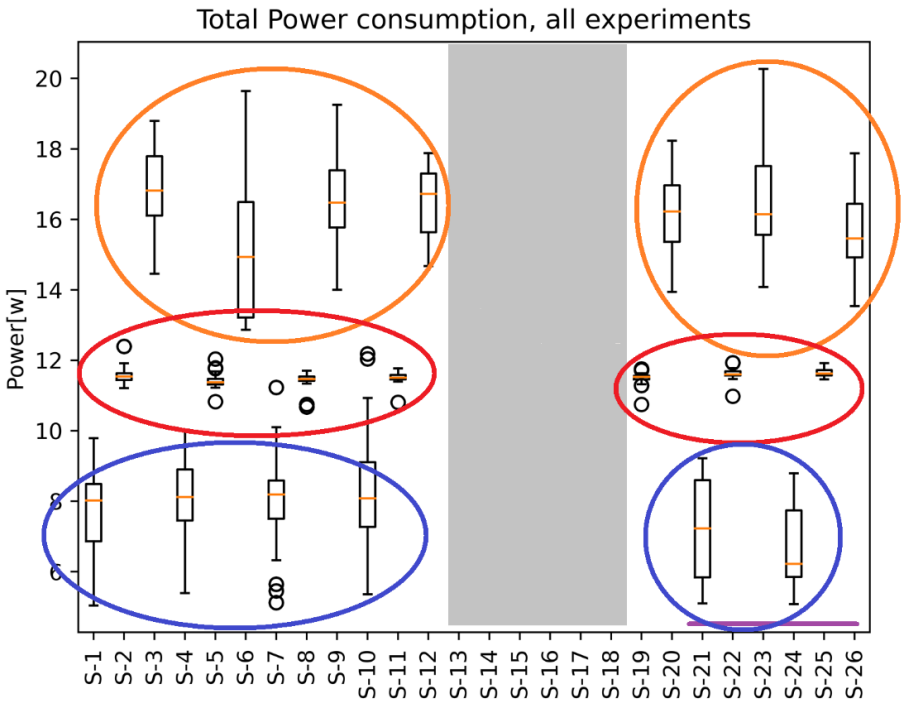


Figure 56: Total power consumption for all experiments. Orange: distributed calculations, red: remote, blue: local. The purple line shows where mmWave communication was used, WiFi for all other scenes. The grey rectangle represents scenes where the power log could not be used.

The contents of the scenes have no visible impact on the power consumption. This holds true for all scenes, regardless of communication technology and whether the vehicle is static or dynamic.

4.4.2 Latency

Considering the total time for all experiments, see Figure 57, the most obvious outliers are encircled in red and are due to the measurement being made far away from the communications device with either entirely remote or distributed processing. Due to a slow data transfer, the total time increased drastically.

It is also clear that the scenes using mmWave communication, underlined in Figure 57, have higher maximum latencies than those that use WiFi-connection. Inspecting the boxes, a latency around 100 ms is found to be common when using both WiFi and mmWave, which is five times higher than the KPI.

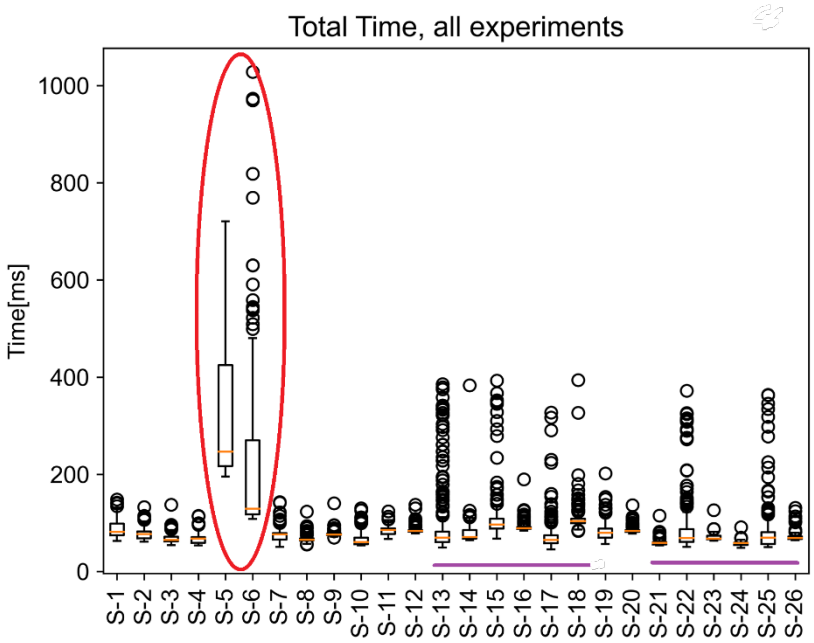


Figure 57: Total latency for all experiments with outliers due to intermittent communications encircled in red and 5G mmWave-communication marked with purple lines, S-13 through S-18 and S-21 through S-26.

Studying the bitrate over a 20-second interval, as has been done in Figure 58, reveals that the 5G network has a bitrate that is initially slow, then overshoots before stabilizing and being quite steady. The WiFi connection on the other hand fluctuates more over time, but still has an overall high throughput.

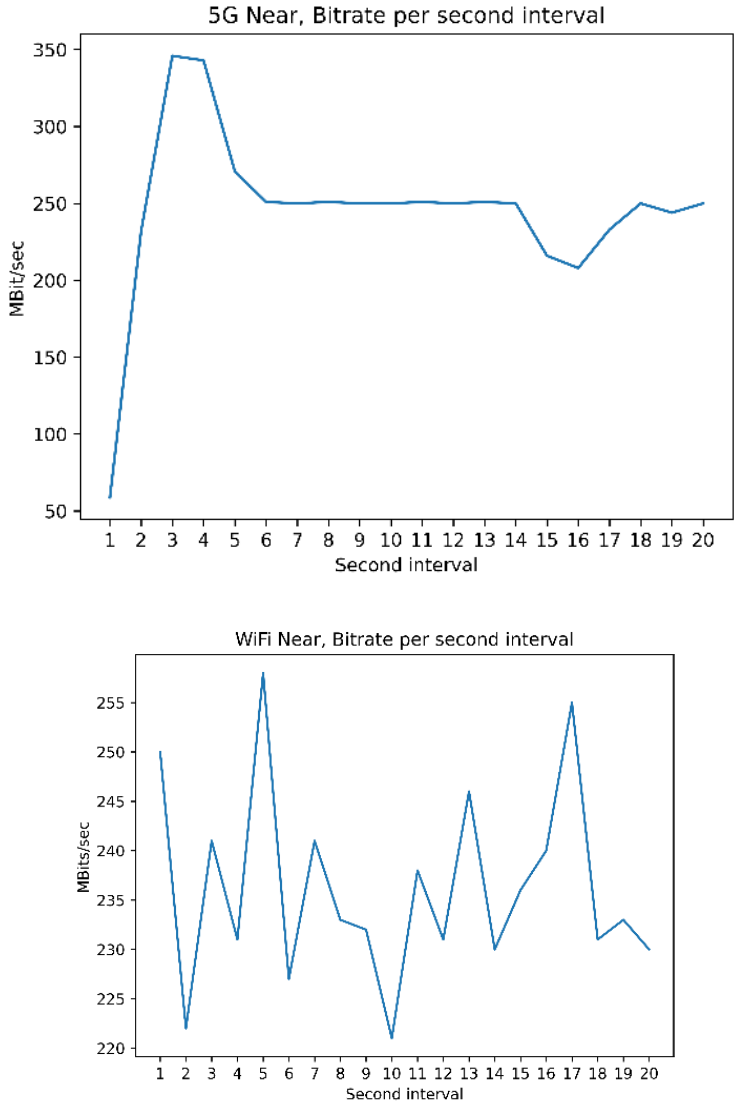


Figure 58: Comparison between 5G and WiFi bitrates as a function of time.

Note that “Near” means 12 m for the WiFi and 80 m for the 5G network.

4.4.3 Detection Rate/Accuarcy

The goal of the ML inference was to determine whether or not there was a pedestrian (dummy) in the scene. In Figure 59, the accuracy of the model is shown. The orange bars and adjoining boxes represent scenes where a pedestrian dummy was present. The blue bars represent the other object and finally, the green bars denote an empty scene.

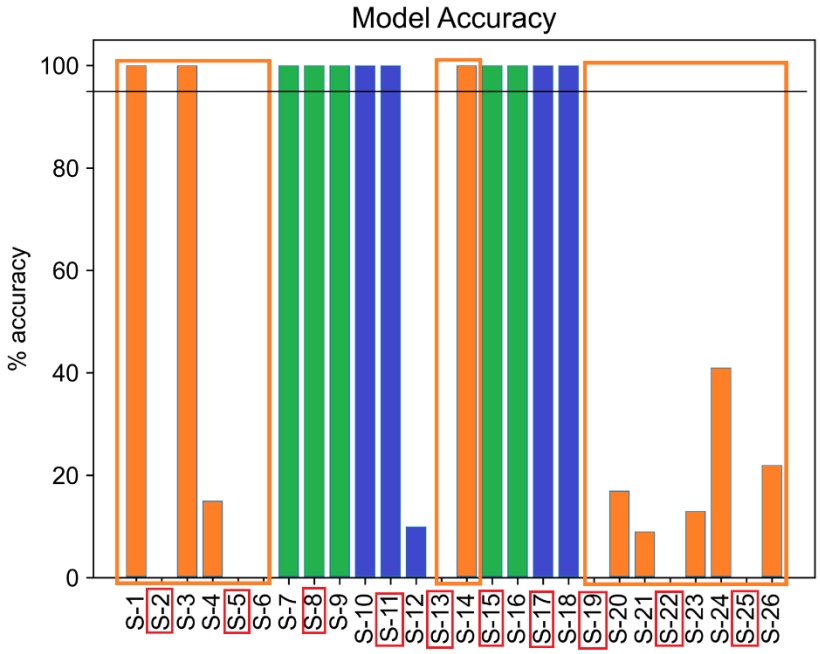


Figure 59: Accuracy of the ML inference: orange = pedestrian dummy, green = empty scene, blue = other object. Blue boxes around the scene numbers on the x-axis indicate all remote processing. The horizontal black line represents the 95% accuracy KPI of the use case.

These detection rates are the aggregated detection rate for the entire data collection sequence of 60 seconds for the first 20 scenes. The dynamics scenes with the moving vehicle S-21 to S-23 have a 20 s collection time and S-24 to S-26 only 16 s.

Starting with the scenes containing the pedestrian, it is clear that only two of the ten static scenes have a high accuracy. Remarkably, the inference performed entirely on the far edge never results in a correct classification. This also holds true for the dynamic scenes with pedestrian, indicating that there is something wrong with that mode of computation. The distribution of the calculations, as described in Section 4.2.1, is made by first performing local calculations on the near edge, then transferring data to the remote far edge for finalization. This type of inference generates both possible results in a logical manner. The entirely remote calculation is made by transferring the unprocessed image to the far edge where the process continues and the result “no dummy” is always delivered in this setup. It does not, however, indicate the inference process is defective, just that no correct answer is delivered.

Therefore, the empty scenes and those containing another object where all remote processing was used need to be removed from considerations of accuracy which leaves the identification of the empty scene which is correct in all three cases, while the accuracy of the identification of the other object as a non-dummy is higher than 95% in two of three scenes.

In Figure 60, instantaneous inference results as a function of driven distance is shown with the maximum velocity at 30 kph. The pedestrian is placed at 100 m, and the car breaks in time to avoid a collision with the dummy. Here it can be seen that at large distances from the dummy, it is not properly identified. At a threshold range value, the distributed

calculation gives the correct result and the all local follows very closely thereafter. For some unknown reason, no identification is made with the all remote calculation.

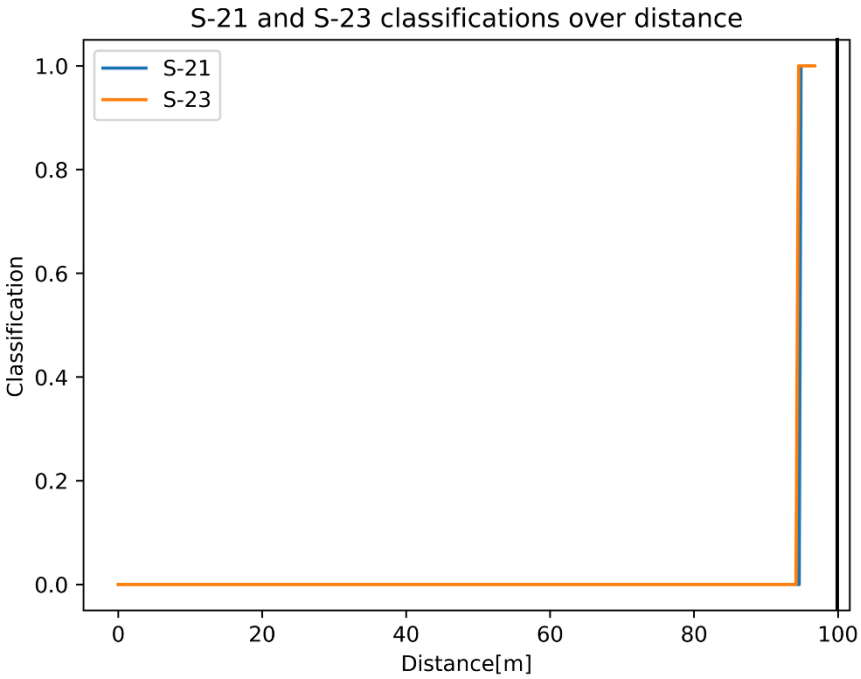


Figure 60: Detection as function of distance, 30 kph, S-21 blue, local, S-23 orange distributed, pedestrian at 100 m.

The results with a maximum velocity of 50 kph are shown in Figure 61. Here, the several single and separate detections occur for the local processing, S-24, until the steady detection state is entered. It can be noted that, see Figure 59, S-24 has a higher overall success rate, but most of that is accounted for by the unsteady detection state. The distributed and the local calculations enters a state of steady detection very close to each other in distance. The remote calculation does not deliver any detection at this velocity either.

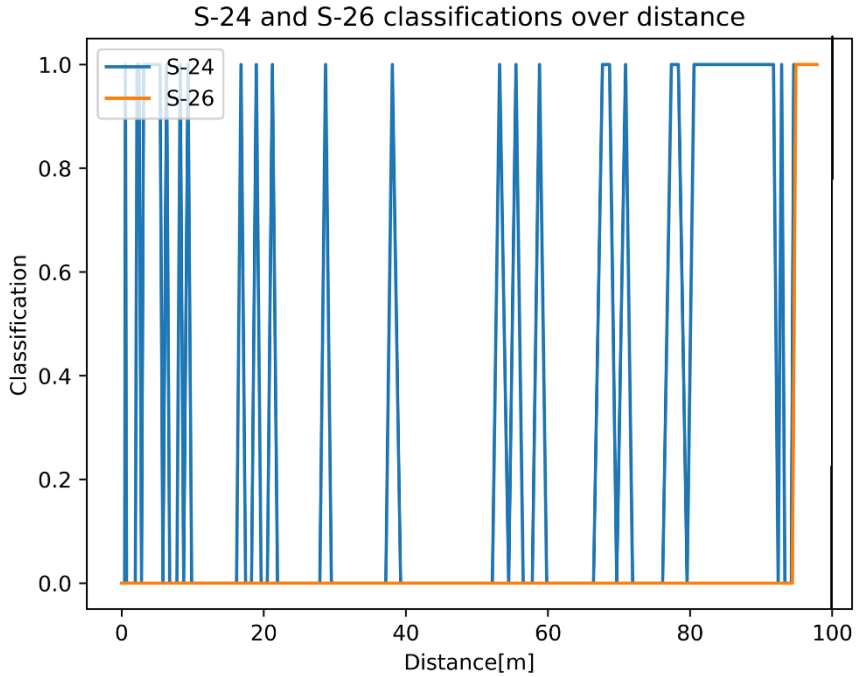


Figure 61: Detection as function of distance, 50 kph, S-24 blue, local, S-26 orange distributed, pedestrian at 100 m.

Inspecting what the images from the onboard camera actually looks like can to some degree explain the results from the detection process. The first image captured in a dynamic scenario is found in Figure 62.

Figure 62: First image, dynamic test S-21.

Here, the pedestrian can barely be seen in its position in the middle of the lane 100 m from the vehicle. As the car closes in on the dummy, it gets increasingly visible in the image, see Figure 63.

Figure 63: Final image, dynamic test S-21.

At the end position, the dummy is clearly visible. As can be seen in Figure 60, the state of detection changes from “no dummy” to “dummy” rather close to the car and the corresponding image is found in Figure 64.

Figure 64: The image when detection occurs in the the dynamic scene, S-21.

4.4.4 Robustness

The robustness is said to be the ability to transfer data at a useful rate without having to resend a large amount of packages. There is no particular KPI for this property, but it relates to the latency as a low data rate and a large fraction of resent packages means a higher latency and consequently a lower frame rate.

The robustness of the wireless connection is presented in terms of bitrate and how many transmission retries occurred prior to the testing. A network test using iperf3 [25] was performed during a 20 second window. The same procedure was repeated at the same geographic positions as the experiments. The result is presented below in Figure 65 and Figure 67.

The bitrates for the four static positions are shown in Figure 65, and as can be expected, an increased distance from the WiFi access point means a lower data throughput. Note the significant drop in bitrate between the WiFi Near, 2 m and WiFi Medium, 12 m. The 5G network has a high bitrate, but also a larger variation, see the circles that represent the outliers.

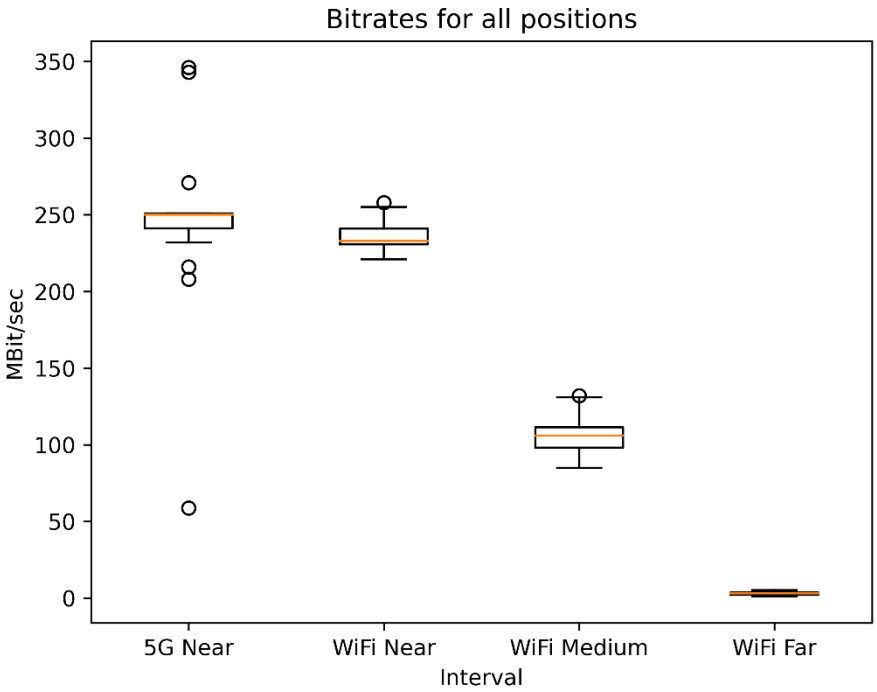


Figure 65: Bitrates for the four static positions.

The network tests were done at the same position, and close in time, as the experiments, however they were collected prior to the experiments. The results are therefore an indication of the properties of the wireless data link. We can see that the bitrate changes over time, but it will most likely not change in the same way over time during the experiment. Figure 66 illustrates the bitrates over 20 seconds for the static position WiFi Near.

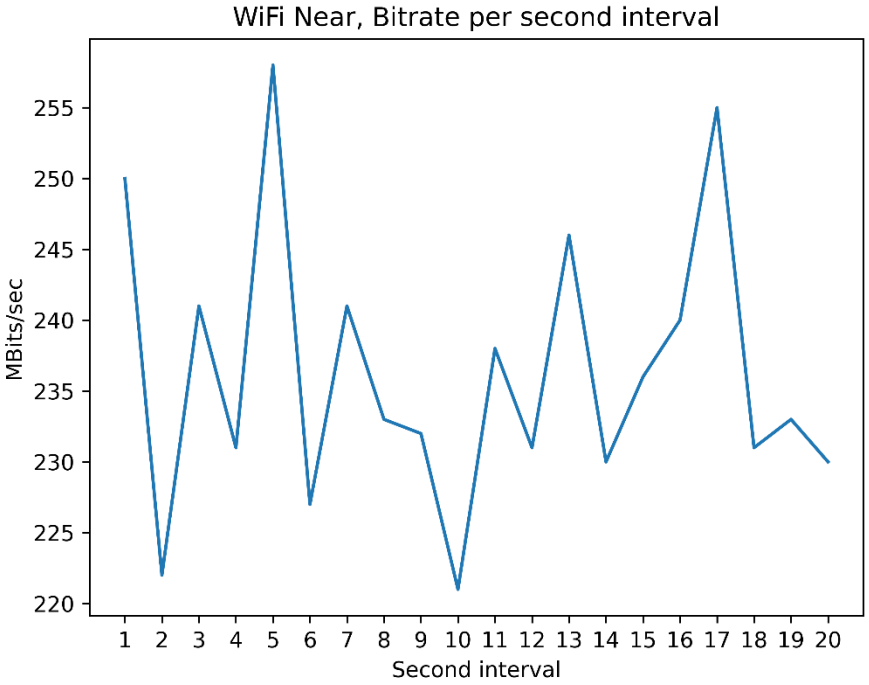


Figure 66: Bitrate as a function of time at the static position WiFi Near.

In Figure 67, the retries for each static position, that is the number of times the communications protocol deemed it necessary to resend a package, are shown. For the 5G Near, WiFi Near and WiFi Medium, the retries are rare, but does occur. The 5G exhibits more significant outliers. Not surprisingly, the WiFi Far option, which has demonstrated a low capacity to transfer data also requires the largest number of retries.

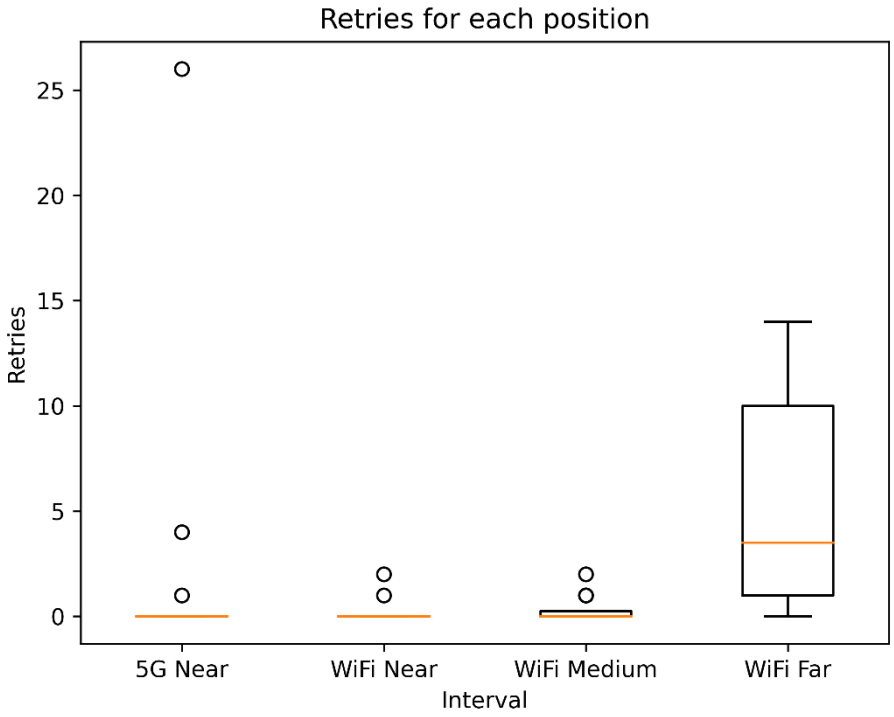


Figure 67: Retries for the four static positions.

4.5 Discussion and Conclusion

The results presented regarding the automotive use case represent an effort to determine whether it is possible to divide the computational effort of a ML inference between different physically separated nodes. It may seem illogical to try and distribute a safety-critical function, but the intent is only to experiment with the distribution of computations in a situation that is relevant. That being said, considering vehicles that lack substantial computational power, being able to supplement the existing capacity could be a major advantage from a safety perspective.

As this data set is the first of its kind and the test time was limited, there are a number of phenomena that remain unexplained at this time and consequently need to be further investigated: The results from the inference that was performed entirely using the remote compute resource were not properly delivered. Furthermore, the connection using the 5G network has inherent properties that affect the latency in ways that remain to be understood. As the project is ending, this will take place in another context. Most parameters, with the logical exception of the accuracy, are not directly influenced by the contents of the scene.

There are two KPIs for the use-case and they relate to the latency and the accuracy. Considering the latency, it can be seen in Figure 57 that the boxes in the plot, representing the majority of the data, indicate latencies around 100 ms which is five time higher than the stipulated KPI which is less than 20 ms. The results regarding accuracy show that either the

inference is successful to at least 99.8% or significantly below 95%, the KPI limit, c.f. Figure 59.

The power consumption is interesting both from the perspective of a vehicle with limited energy storage, e.g., an electric car and the environment considering how much energy that is used to execute a function. The results show that the local computation requires the least amount of power, the distributed the highest and the remote less than distributed but more than local. This probably does not give a complete picture of the situation, as the remote far edge consumes a basic amount of power just being active, and the added need from the computation is very small compared to that. In these measurements there is only one user while the far edge is capable of processing data from a significantly larger number of users.

Considering the latency, which is a significant factor from an automotive safety perspective, the results are presented as a compound value including both the inference and the data transfer. Because the actual data transfer rate achieved was 0.3 GB/s as opposed to the initially estimated 10 GB/s has surely influenced the result. There is room for improvement to shorten the latencies introduced by the communication and reach the KPI level. The results are still interesting. When there is a large distance between the near and far edge, the data transfer rate is reduced, causing latencies as large as around one second. The scenes where WiFi is used for the data transfer have lower maximum latencies than with the 5G, which requires some investigation.

Regarding the detection rate or accuracy of the model, it is evident that the ability to detect a human dummy is not sufficient to use in a safety function. As the choice of use case was not made with that intention in mind, but to be able to show the distribution of the processing in a realistic setting, that is, this could be useful in the future.

The cases with the empty scene were satisfactorily classified and those containing the trash can were to a large extent accurate, even when disregarding all scenes with remote processing as those results are erroneous. That is, the system does not mistake a trash can for a human. However, the ability to correctly determine that a human dummy was present needs further improvements.

Considering the scenes with the moving vehicle, the low accuracy is partially due to the fact that the resolution of the images limits the ability to detect the dummy. When inspecting the classification result as a function of distance to the dummy, it could be seen that for all cases except those with remote processing, proper classification does occur, albeit at a distance around 5 m from the dummy.

The robustness of the data connection is of extreme importance for a distributed calculation as it directly influences the latency of the system. The WiFi connection exhibited a stable performance and behaved as expected, the longer the distance of the communication, the lower the bitrate. The 5G mmWave option had, at the time of the data collection, not been thoroughly investigated, but is expected to have a large data transfer capacity with low latency.

In conclusion, it has been demonstrated that the computational processing power used to execute a machine learning model designed to detect the presence or absence of a human dummy can be divided between two computational nodes, physically separated and connected by wireless data transmission network, WiFi or 5G mmWave. The demonstration has been done in a realistic setting for tests of a distributed function.

4.6 Challenges and Future Work

When distributing an ML inference in a traffic scenario, a major challenge is ensuring that correct and adequate remote resources are available along with the necessary communication capacity. Security and privacy aspects of the communication must be considered.

It was expected and indicated by results from theoretical analysis early in the project that it would be possible to get a more efficient, with respect to power consumption and latency, inference by remote or distributed inference. To understand why this was not the case in practice, further data collection and analysis is needed.

The future work involves an investigation of the communication alternatives as the data transfer capacity directly influences the latency of distributed or remote computations. Furthermore, implementing separate logging of the components of the latency to be able to understand the operational aspects of the system.

An analysis of the power consumption and computational load in the different system configurations is necessary to understand how to deploy the inference function in the most efficient way in a full-size use case with a large number of vehicles.

5 Smart Home Use Case

The most commonly used features of applications in a smart living environment include using deep neural networks to detect and recognize objects, gestures, and faces. In many scenarios, the application is personalized for each individual user, so detection of identity and an easy way of controlling it is highly beneficial. These algorithms can be used for a smart kitchen, which supports the user's cooking, or a fitness coach, which trains the user.

A smart mirror is an appealing example of the combination of object, gesture, and face recognition. Figure 68 is showing an example of such a smart mirror. It consists of a display with a semi-transparent film attached to it. With this, the user can use it as an ordinary mirror in their entrance hall. However, the environment can also provide additional information, like the public transportation schedule or the weather forecast. Recognising the user can be used to personalize the information and gestures for changing the widgets and so on. A self-sustained life for elderly people can also be supported if a reminder for necessary items like an umbrella if it rains or the home keys are provided, or appointments are shown.

Figure 68: Smart mirror prototype utilizing a t.RECS edge server in an acrylic case

The focus of the smart home use case in this project was primarily on the most computationally intensive image processing techniques for these three features. In addition, speech and natural language processing are important topics for creating voice assistants, which have already found their way into everyday life. Due to being in the home environment, an essential aspect of such a trustworthy application is local calculation. This

is done by focusing on utilizing hardware developed within this project, and no image or information leaves the system. Of course, this must not be done at the expense of the electricity bill, which is why efficient AI hardware accelerators and adapted machine learning methods are used and implemented.

5.1 Developments & Optimisation

The main work in this use case is centred around a smart mirror demonstrator used as a user interaction interface with the intelligent living environment, see Figure 68. The data path structure of the demonstrator in this project is shown in Figure 69. It is derived from the use case of the former project LEGaTO [26].

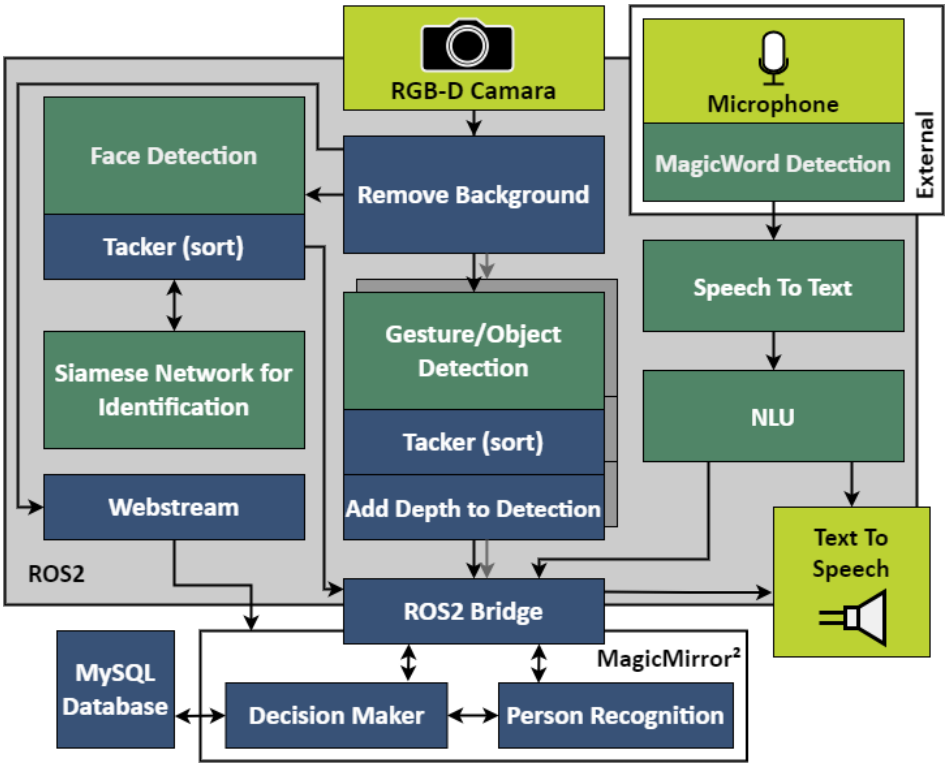


Figure 69: Data flow of the smart mirror demonstrator

As the primary input, an Intel RealSense camera records the user's image, and the background is removed. The image is shared with many nodes for different purposes, e.g., object detection. All detections are tracked using a tracking method called **s**imple **o**nline **r**eal-time **t**racker (sort) [27]. A Kalman filter is used to predict the next position of the object, and the Hungarian algorithm combines the new detection and these predictions. This aims to assign a unique ID so that the object can be traced over several frames. Additional information is also added to the detection, like depth for gestures and objects or identity and emotions for faces.

All collected data is streamed into the MagicMirror² [28] instance for visualization, and all decisions are made there. All person-related detections are combined within the person recognition to attribute all gestures and faces to the right person. This combination also enables one to keep track of a person even if the face is not visible for a moment.

The following subsections describe changes made to the general software stack of this smart mirror within this project to enhance performance.

5.1.1 ROS2 Implementation

In the original system deriving from the LEGaTO project, communication was realized via the framework OmpSs [29], GStreamer, and standard io. The various blocks formed sub-programs between which the data had to be exchanged (sometimes even across system boundaries). However, this became a bottleneck, limiting the framerate to 16 FPS.

In this project the various parts of the prototype were transferred into ROS2 nodes in C++ with different backends to simplify interchangeability and communication between the different algorithms, e.g. the object detection can be executed on the Nvidia Jetson modules, the Hailo-8 or the Xilinx FPGA. Initial difficulties of ROS2 with older Ubuntu versions of various modules and packet dependencies were solved, and the performance increased to 30 FPS, which is the limitation of the attached Intel RealSense camera. A NodeJS ROS2 bridge finally provides parts of the data stream to the MagicMirror² UI.

5.1.2 Restructuring the Face Recognition

Originally, face recognition was split into two parts. Faces were detected and tracked, and these tacked faces were periodically identified. For the first part, a derivative of a Retina Face model was evaluated, and a support vector machine was used to identify the seen user. The SVM step was forming a bottleneck for more trained persons, and other solutions were needed.

The comparison using the cosine similarity can be used to compare the tracked faces' feature vectors and saved values for recognized users and return the similarity [30]. This also implies that no image needs to be saved; only the feature vectors are needed. The feature vectors are 1024 floats in width calculated by a feature extractor, which enables high performance. Figure 70 depicts the interaction between face detection, user recognition, and emotion recognition. Images are fed into face detection, which calculates the bounding boxes around the faces. These faces are cropped out and sent to emotion detection and user identification using a feature extractor and the cosine similarity.

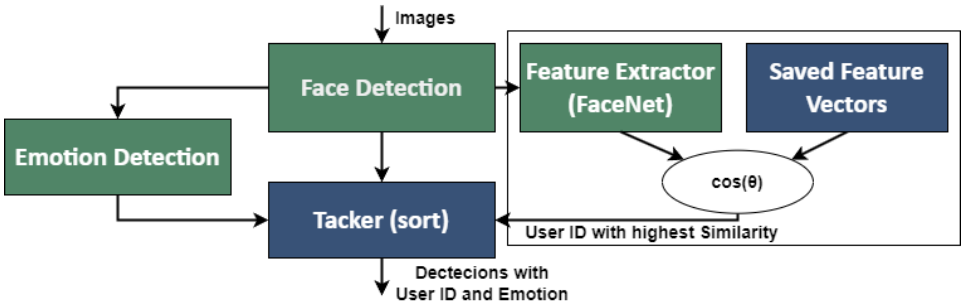


Figure 70: Structure of the face detection and recognition. Cropped images of faces are handed to emotion detection and the feature extractor. The additional information is added to the tracked faces.

The face-detecting model was removed in the last step, and the face class was integrated into the gesture dataset. This reduces the necessary amount of DNN and helps to run on less power-hungry hardware. This feature is optional to enable the usage apart from this use case.

5.1.3 Additional Feature Integrations

Similar to face identification, emotion detection and mask placement assessment were integrated but kept optional to stay comparable to the baseline setup. For both modules, open-source datasets and models were used.

Due to the direct interaction with the user, the user's emotions can be beneficial for decision-making. If a user is sad or depressed, nice words or other emotional support could

be provided. A neural network was introduced to recognize the six basic emotions (neutral, happy, angry, disgust, sad) with an accuracy of 83%.

In the frame of the COVID-19 pandemic, a neuronal network that verifies the correct fit of the mask was introduced. It determines whether the mask is worn correctly, incorrectly, or is missing altogether. The Accuracy of this model was around 90%.

Additionally, an LED strip was attached to the camera and screen to provide better lighting conditions for the user in front of the camera and screen. This improves user recognition as it had trouble recognizing faces in low light conditions.

5.1.4 Depth Information for Gesture Control

Due to the usage of the RealSense camera, depth information is already available. The distance of the gestures and object to the mirror is calculated and added to the bounding boxes in a third step of the processing pipeline and used for interaction with the user.

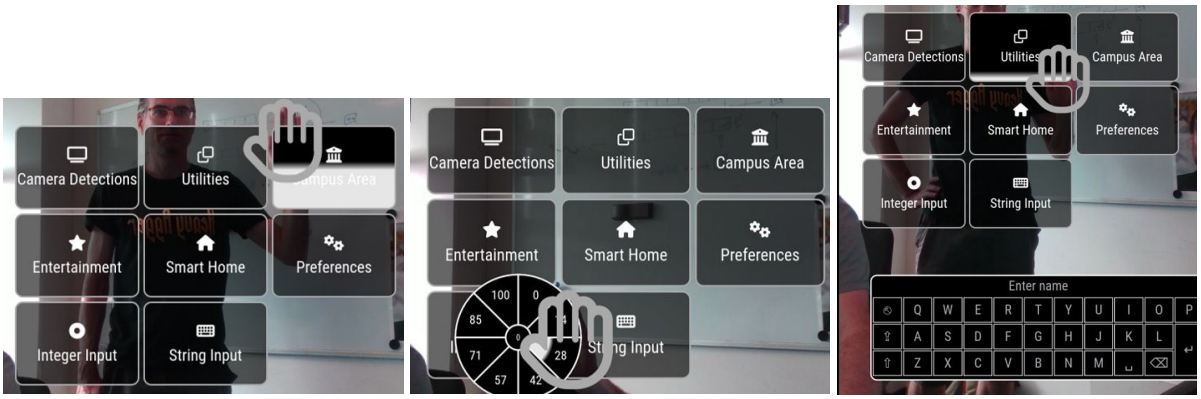


Figure 71: Examples for the menus used in the smart mirror prototype. The main menu on the left, a wheel for numbers in the middle, and the onscreen keyboard on the right. All are controlled by the distance of the flat right hand.

For the gesture control different menu structures are implemented, seen in Figure 71. A main menu using tiles is the main way for interaction. A wheel is used for inputting numbers and an onscreen keyboard is used for strings. The location and distance of the flat right hand is used as input. With the location the tile or element is selected and with reducing the distance this element then activated.

5.2 Hand Gesture Dataset and Automated Capturing

The creation of the gesture dataset was reworked entirely and automated. The idea of an image crawler using openly accessible sources was evaluated. Hand gestures, people, and faces in YouTube videos are to be used to generate a data set. A multi-stage process was implemented for this purpose. Firstly, YOLOv7 is used to recognize people reliably. These people were cut out of the image, and the landmarks for faces and hands were found with the help of Google MediaPipe [31]. This is necessary because the MediaPipe is limited to one person. Several people in one image are making the result unusable. Each person's

detections can then be fed back into the original image to determine the appropriate bounding boxes.

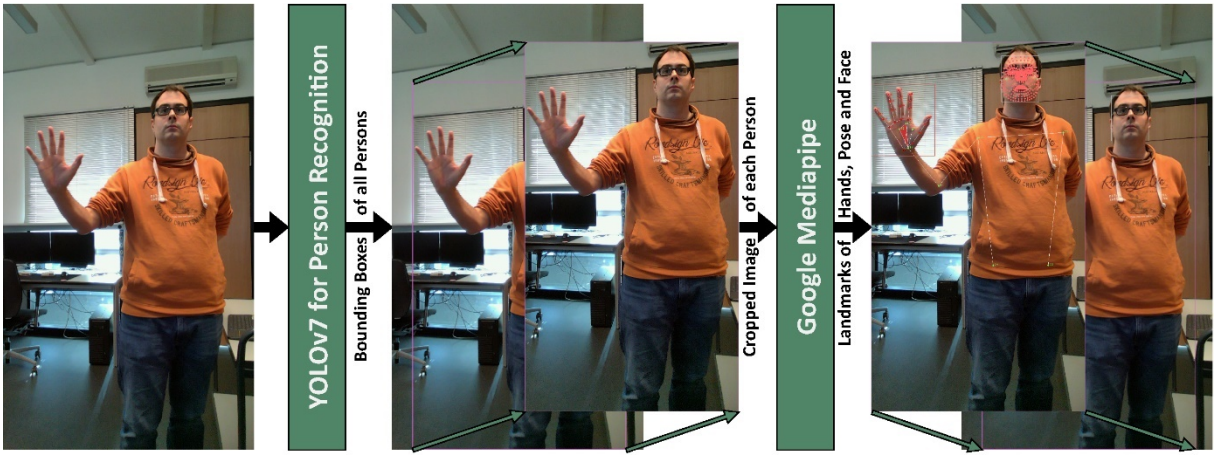


Figure 72: Example for the detection of landmarks within an image using YoloV7 and the google MediaPipe. YoloV7 is used to find persons in the image as the MediaPipe is limited to one person. This is afterwards used to extract the needed landmarks.

The described process of landmark extraction is shown in Figure 72. In the next step, a simple SVM or a machine learning algorithm is used to classify the image if a gesture is seen. Therefore, a small benchmark dataset is required to train this classification step. This small benchmark dataset was captured using the same algorithm, but a given gesture to be seen. The dataset consists of 38 gestures of 14 people with 100 images each. With the two steps of detecting landmarks and persons in images and a simple classification if a gesture is included, many YouTube videos can be analyzed autonomously without manual labeling.

A large and evenly distributed data set across the various gestures was collected and labeled using the entire tool flow. Around 3000 of these images for each of the 38 gesture classes were used to train YOLOv7 and YOLOv7-tiny, and the **m**ean **A**verage **P**recision (mAP) for each class is shown in Figure 73. In all classes, the mAP was above 0.73. This is an overall useful result for both versions. For performance reasons, the tiny version was primarily used for the smart mirror prototype until optimizations with the EmbeDL tools were conducted.

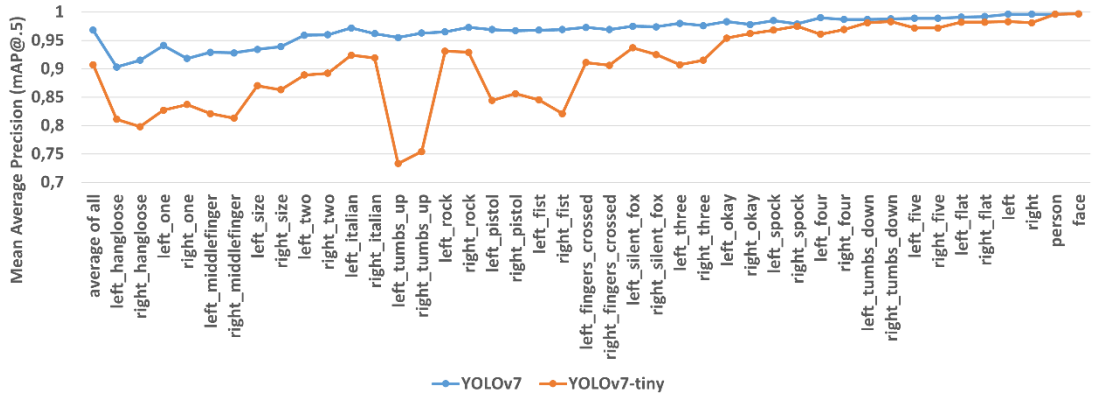


Figure 73: mAP at IoU threshold of .5 for YOLOv7 and YOLOv7-tiny for the captured gesture dataset

5.3 Optimization of YOLOv7

Two of the most used neural networks in this use case are based on the YOLOv7 architecture. To improve energy efficiency, reducing the required operations would be beneficial. Evaluations were conducted using the EmbeDL Toolchain [32] for pruning the object detection DNN, which is using the COCO dataset, and the gesture detection DNN,

which is using a self-created dataset. YOLOv7, with an input resolution of 640x640 pixels, is used for pruning in both cases. With the initial DNN, processing one image requires 104,5 **G**iga **F**loating **P**oint **O**peration**s** (GFLOPs). The focus here is on increasing energy efficiency by reducing the required GFLOPs. However, it should be noted that reducing the number of neurons is expected to decrease the accuracy, which is measured by the **m**ean **A**verage **P**recision (mAP) with an **I**ntersection **o**ver **U**nion (IoU) threshold of 0.5 or the average from 0.5 to 0.95. Therefore, a balance between GFLOPs and the mAP must be achieved.

For the pruning process, the EmbeDL tool calculates the score for each neuron, indicating its importance. The least essential neurons are removed, and the process is repeated until a predefined size is reached. Only the input size and output size remain unchanged, as these are based on the input image and the number of classes. Furthermore, no layers are removed. The resulting smaller DNN has lost accuracy and must be retrained. Each model considered here was retrained for 15-50 epochs after pruning. The number of epochs required depends on the model's size and the dataset's difficulty. The COCO data set, for example, requires a higher number of epochs than the gesture data set. An effort was made to avoid overfitting. The two following sections are showing the results for the two different datasets.

5.3.1 Optimization of Object Detection

Figure 74 shows the measured results for pruning YOLO trained with the COCO dataset [33] to different sizes, ranging from the original size to only 10% of it. The blue bars represent the GFLOPs required, while the two lines represent the combined map achieved. Additionally, the value for the tiny model version is included for comparison purposes. If only a small number of neurons are removed, the overall accuracy remains reasonably high; a higher drop is noticed after the size is reduced to smaller than 40%. Additionally, the YOLOv7 model adds extra detections at this pruning size, as shown in Figure 75. The 10% size version is also less accurate than the tiny variant with a comparable size. The layer structure and the number of layers can explain this. YOLOv7 has 306 layers, and YOLOv7-tiny has only 200 layers for inference. After observing hallucinations and problems in some classes, each object's accuracy was evaluated.

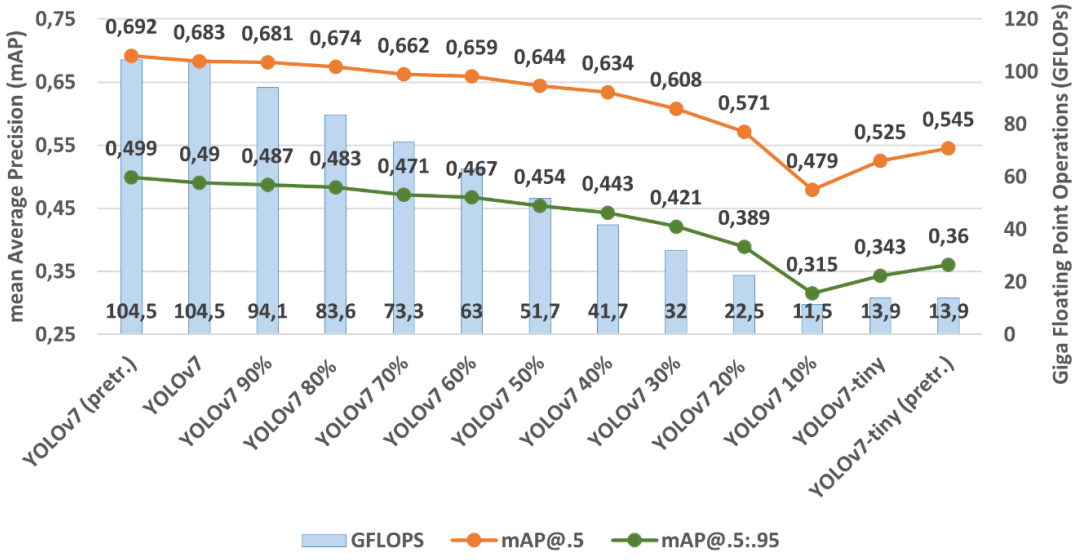
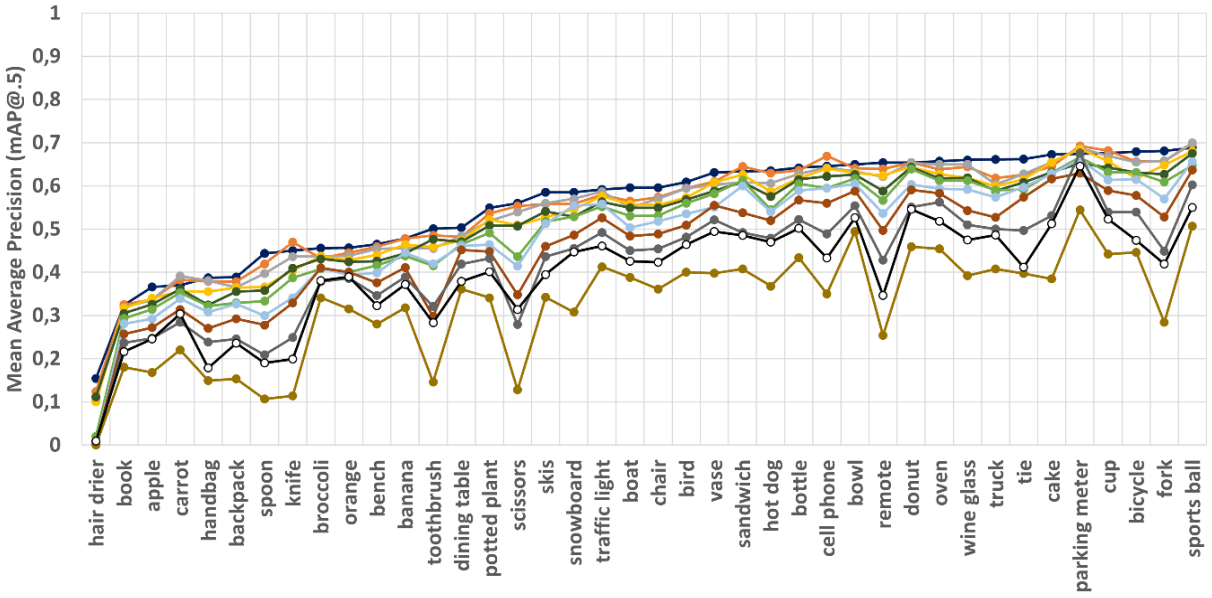


Figure 74: Comparison between different pruning sizes regarding the needed GFLOPs and the achieved accuracy for the object dataset.

Figure 75: Example for additional detections from smaller network sizes. Here 50% pruned YOLOv7 adds a cow

When comparing the individual object classes across the different pruning sizes of YOLOv7, it becomes apparent that some classes are 'forgotten' earlier than others, and some appear to be favored. Figure 76 depicts the accuracy for all 80 classes of the COCO dataset. It is ascending sorted for the top line, representing the initial pre-trained YOLOv7 model. The mAP accuracy of some objects is very low from the start. This is due to the dataset's lack of images for these classes, such as the hair dryer. Generally, a uniform decrease in mAP for higher-pruned models is expected. However, some classes are exceptions, such as toothbrush, scissors, remote, fork, and especially the toaster, which are forgotten faster. On the other hand, many classes are remembered longer, such as parking meter, sports ball and many classes with a higher mAP in the initial model. Further investigations into the relationship between all single neurons and the individual classes extend beyond this use case in this project.



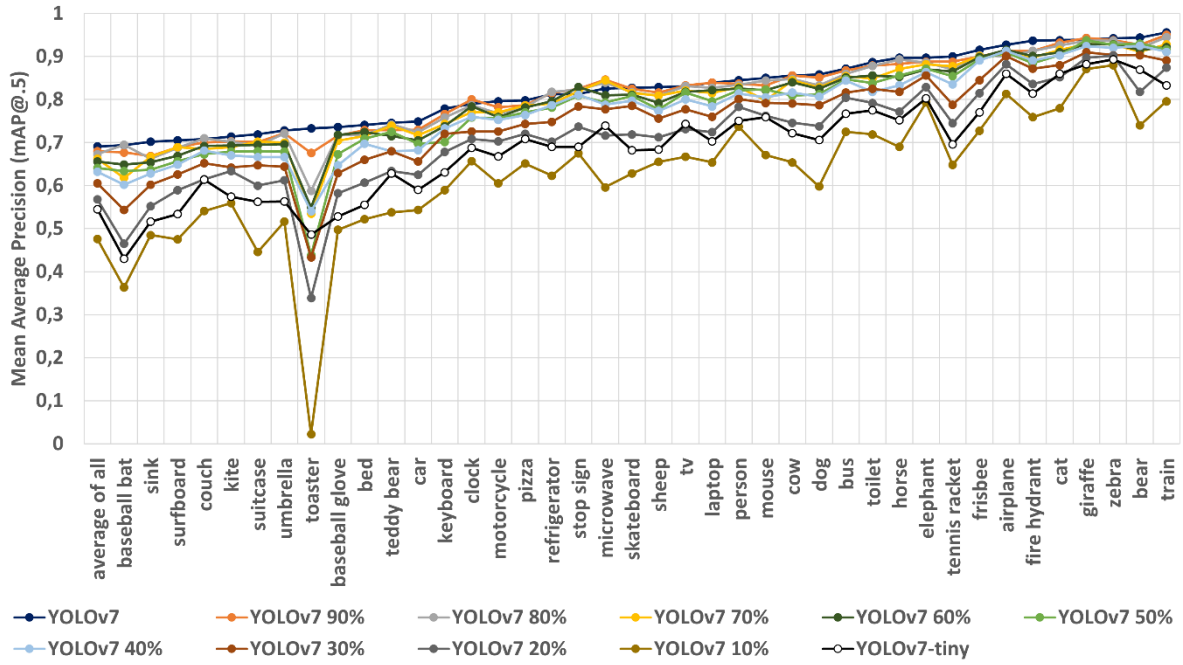


Figure 76: Accuracy comparison for all model pruning sizes and each class of the coco dataset individually

In the case of the smart mirror prototype, object detection is the biggest DNN and is primarily outsourced to accelerators like the Hailo-8. Reducing the needed GFLOPs is beneficial for reducing the amount of context switches on this accelerator. The initial YOLOv7 model is too big for a single context and split into five contexts on the Hailo-8. Therefore, for each image the context is switched five times, and this overhead is also consuming energy. The pruned to 60% sized model only uses four contexts instead. Figure 77 depicts the evaluation of the different pruning sizes and the resulting values in max FPS, latency and power consumption. As expected, the maximum FPS rate increases sharply as the size decreases. As the number of layers is kept the same, the latency only decreases slightly. Because the maximum FPS and the shortest latency are always determined, the energy consumption is only slightly reduced.

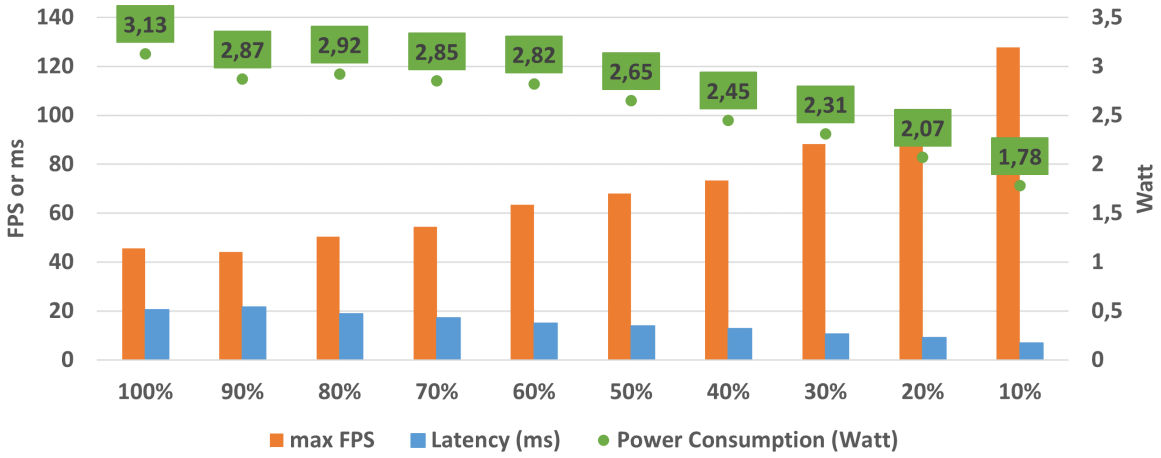


Figure 77: Benchmarked performance evaluation for varying model sizes of YOLOv7 on the Hailo8 accelerator. Measured are the maximal FPS and Latency on the left axis and the corresponding power consumption on the right

5.3.2 Optimization of the Gesture Detection

Similar to the previous chapter Figure 78 shows the comparison of the GFLOPs and the mAP for different pruned YOLOv7 models for the gesture dataset. A stronger focus was placed

on the smaller variants as these still have a very high level of accuracy with the gesture data set. Even when pruned to a comparable size, the pruned model shows a higher map than the tiny version. As the dataset is very simple, the mAP@.5 of all variants are above 90%. As only the average mAP of all classes was evaluated before, the tiny version was running in the smart mirror prototype so far.

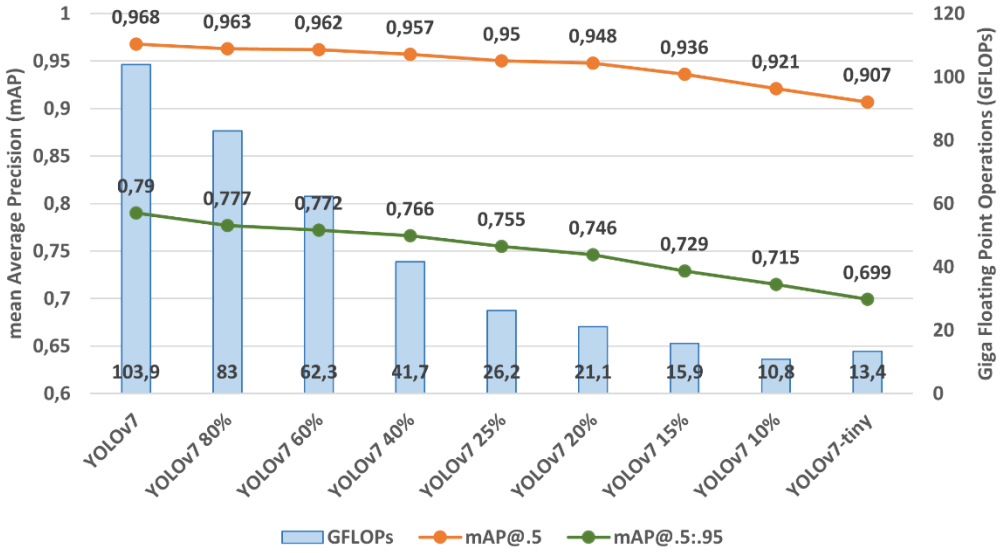


Figure 78: Comparison between different pruning sizes regarding the needed GFLOPs and the achieved accuracy for the gesture dataset

After calculating the mAP@.5 for all individual gesture classes a strong drawback was discovered, depicted in Figure 79. The thumbs-up class is not as easy for smaller models as the other gestures. As the thumbs up gesture is used as a direct command, hick-ups can be explained by this lower value and the pruned YOLOv7 model can help circumvent this behavior. The 15% size model shows slightly better accuracy by around the same calculation amount needed. In further prototypes, this model will be used instead of the previously used tiny version.

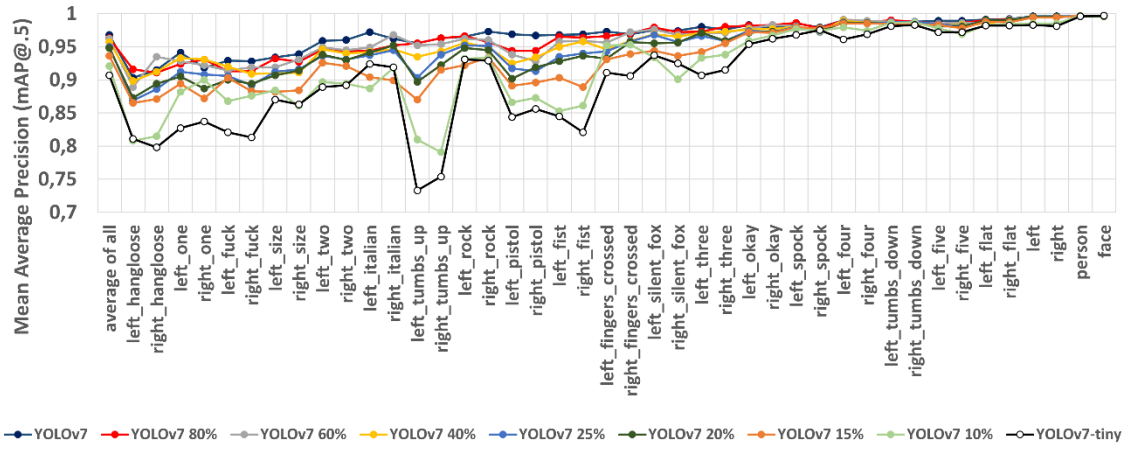


Figure 79: Accuracy comparison for all model pruning sizes and each class of the gesture dataset individually

5.4 FPGA usage for Object and Gesture Detection

In addition to the GPU implementations, the object and gesture detection network nodes were implemented on Xilinx FPGAs, utilizing the accelerators and the hardware-software infrastructure, developed in WP3 and WP4. First, a YoloV4 network trained with the COCO dataset was implemented, which is also part of the general model zoo of this project. As detailed in Deliverable D3.4 [34], the model was implemented on an UltraScale+ FPGA and a Versal XCVC1902 ACAP (Adaptive Compute Acceleration Platform), integrated into a VCK190 Evaluation Kit [35]. While the FPGA implementation was not able to provide the required performance, the implementation on the Versal achieved a performance of up to 195 FPS, which is comparable to the Jetson Orin AGX. For this benchmark, the power consumption of the Versal was 43.7 Watt, while the Orin required 60.2 Watt (192 FPS).

For easy integration into the smart mirror and evaluation of the complete environment, the FPGA-implementation was also encapsulated as a ROS2 node. Experiments combining the Nvidia AGX Orin and the Xilinx VCK190 were conducted, and the full smart mirror setup was running showing 30 FPS with a power consumption of around 83 Watt. For this implementation, only a small fraction of the Versal resources were utilized, implementing a small DPUCVDX8G Xilinx DPU with batch size one. This design can be run also on the new Versal edge devices, which will be integrated into the u.RECS. Running the compute-intensive DL-models on the FPGA significantly reduces the compute load of the Nvidia Orin. Hence, a small Orin NX can be utilized instead of the currently used larger Orin AGX, further reducing power. Therefore, the u.RECS combining an Orin NX and the Versal edge module, developed in WP4, will be an ideal platform for energy efficient implementation of the complete smart mirror environment.

5.5 Creation of a Virtual Mirror Image from 3D Point Clouds

In order to remove the necessity of the reflective foil of the first prototypes, the potential application of point clouds for reconstructing a virtual mirror image was explored. This film was applied to the screen and showed both the user image and the interesting information. Unfortunately, it had its drawbacks, as either the screen or the surroundings were too dark to see both. Nodes were implemented for this, and an attempt was made to implement this in the prototype.

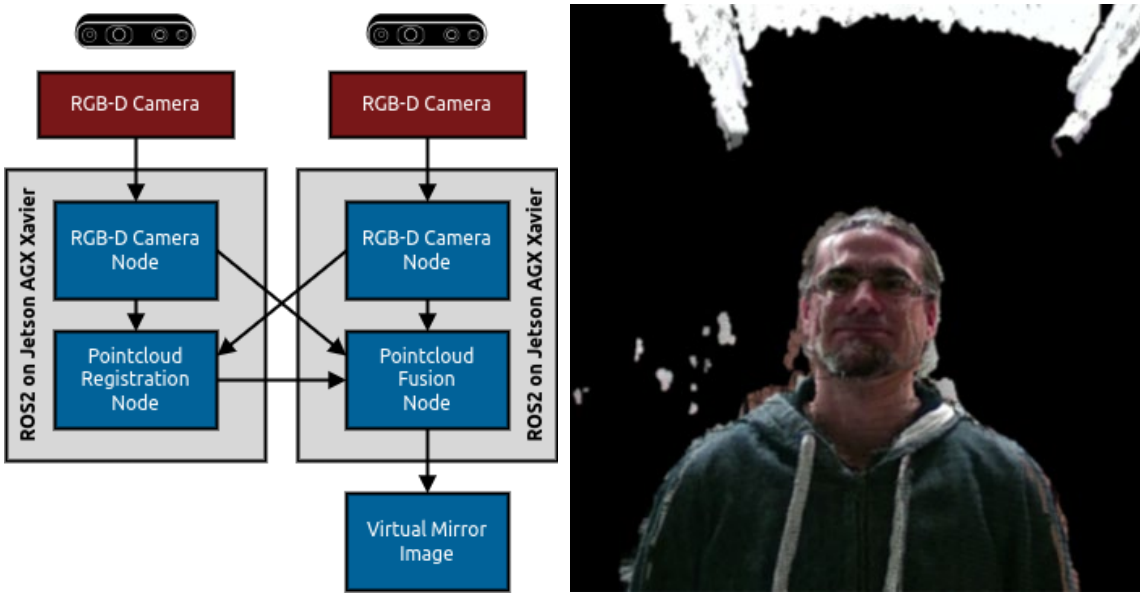


Figure 80: ROS2 node architecture for the virtual mirror image creation and an example of the result

For the registration of point clouds, a voxelized version of the generalized iterative closest point algorithm (VGICP) [36] was implemented, which provides an affine transformation, estimating the translation and rotation between the two cameras. The recalculation can be executed at a low frequency as the camera does not shift that often. Subsequently, a virtual camera with the merged point cloud can determine a frontal image. For this purpose, a processing pipeline containing ROS2 modules was implemented, and the processing was split between the two NVIDIA Xavier modules, as shown on the left in Figure 80. This first implementation achieved a performance of 27 FPS with a latency of 100ms, which is sufficient for real-time operation. The resulting image was not used as a mirror due to image artefacts. These artefacts are due to the point clouds of the used Intel RealSense cameras and are shown on the right in Figure 80. The evaluation of implementing solutions to this problem showed that the targeted hardware configurations were not sufficient to run everything in parallel, which is why energy efficiency was investigated further. As a result, the focus was on optimizing the DNN and using less energy-consuming hardware, and the work was not carried out further in the direction of the virtual mirror image.

5.6 Hardware Evaluation

This use case was built mainly around the smart mirror prototype and increasing the energy efficiency of running object, gesture, and face detection and recognition in parallel on embedded hardware. The structure of the dataflow was described in chapter 5.1. At the beginning of this project, the smart mirror prototype consisted of two Nvidia AGX Xavier interconnected on the t.RECS [34]. This setup consumed around 150 Watt and showed around 16 FPS for the three detections. The hardware was also changed after the first software updates, and different setups were evaluated. Figure 81 shows the measurements for different setups and combinations. After implementing the smart mirror modules into ROS2 nodes the performance was already increased to 30FPS while consuming around 100 Watt on the two AGX Xavier. The combination of a Nvidia AGX Orin with 8 cores and a Hailo-8 is marked in red, as it is the efficient setup evaluated in this project and was shown at the latest fairs. It achieved 30 FPS while consuming around 38 Watt. In this setup, most of the smart mirror prototype runs on the AGX Orin, and the YOLOv7 model for object detection is outsourced to the Hailo-8.

The different hardware modules are also interchangeable and, e.g., the combination of an Nvidia AGX Xavier and an AGX Orin would be possible, but not purposeful. The selection was kept simple by only evaluating combinations which have shown an increased performance chronological.

The blue dot represents the expected performance of the Nvidia Orin NX with a Hailo-8 for object detection. Due to unresolved issues surrounding the current limitation of the Nvidia Orin NX on the u.RECS, evaluations in this combination are still to to be conducted in its entirety. Apart from the UI of the smart mirror, everything is functional and running. However, an overcurrent protection error and slight hiccups can be noticed. This will be fixed in the new revision of the u.RECS.

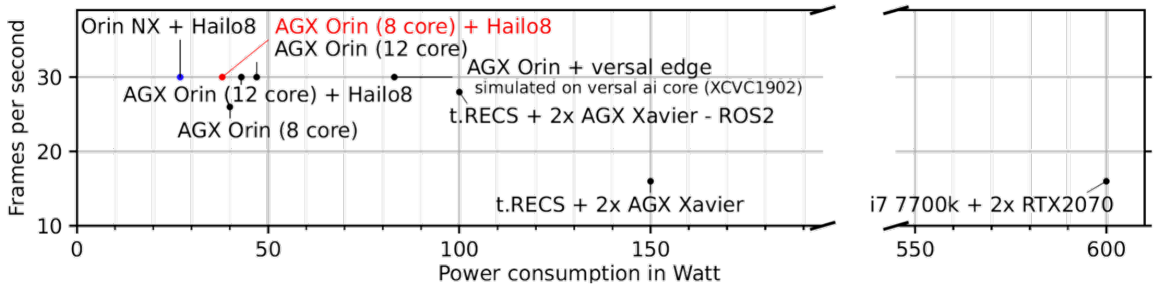


Figure 81: Power consumption and frames per second of the smart mirror on different hardware setups

If the combination of these values is considered, the baseline system using two Nvidia Xavier AGX has a performance value of 9.375 Watt / FPS. The latest smart mirror prototype based on the Nvidia AGX Orin and Hailo-8 is showing a performance of 1.266 Watt / FPS, while expected value for the Nvidia Orin NX and the Hailo-8 is 0.933 Watt / FPS. This shows an improvement of around factor 10x.

5.7 Local Voice Assistant

The previous smart mirror demonstrator centred significantly on image-processing algorithms. In addition to gesture recognition, a voice assistant is a powerful interaction capability. So far, however, only a rudimentary hotword detection has been used. In this, keywords like "show" and "weather" were detected, for example, to start the weather skill. To enhance this, a voice assistant based only on local processing was evaluated in this project. For a long time, the most suitable frameworks for speech-to-text (STT) were Coqui STT [37] or Vosk [38], but they were replaced by OpenAI's whisper [39]. For Text-to-speech (TTS) the models from Coqui [40] or PyTTSx3 [41] were evaluated and implemented into ROS2 nodes. The Coqui TTS shows a more humanlike voice than PyTTSx3.

For data protection reasons, the STT node was the important focus of this project. Audio recorded in the home environment must be encrypted and must never be easily readable. Therefore, how a neural network can be executed within a secure area was investigated. For this purpose, the whisper model was to be executed within the OP-TEE area of a Nvidia AGX Xavier. For this purpose, TensorFlow lite should be used as the backend and executed with the help of WASM. Difficulties were caused by missing operations of a too-old TensorFlow version and very limited memory. The deliverable [42] in WP5 deals with this in more detail.

The natural language processing solutions with SpaCy [43] and RASA [44] are evaluated and show comparable results. The data flow is shown in Figure 82. A simple skill system was implemented based on user intent, and the local voice assistant could present cafeteria and weather information or tell a joke. A version is implemented on a Raspberry Pi 4 using 11 Watt and showing a latency of 40 seconds using the Coqui frameworks or 3 seconds using the other two. This high latency is due to the limited performance of the Raspberry and could be improved using more powerful hardware.

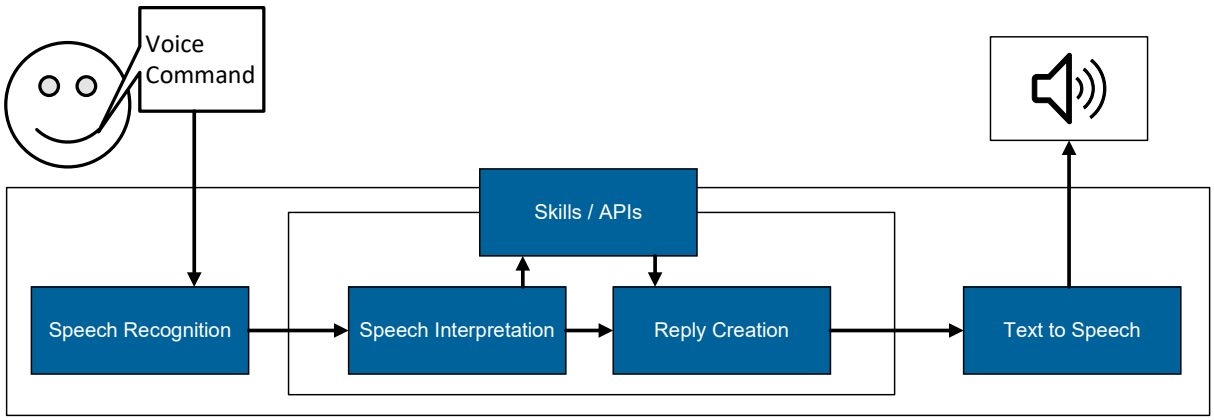


Figure 82: System design of the local voice assistant

5.8 Secure Smart Microphone with Hotword Recognition

As an extension of the integration within a smart home environment, we conceived a wireless smart microphone device as an accessory to the smart mirror demonstrator. The device is based on an Arduino Nano RP2040 Connect [45] microcontroller with a built-in microphone. Spoken keywords are recognized by a small machine-learning model running on the microcontroller. Upon detection of the hotword, a transcript of the word and the

sound wave is sent to a secure service over Wi-Fi. The communication is TLS encrypted and the secure service runs inside a trusted execution environment (TEE) based on Intel Software Guard Extensions (SGX), to ensure the spoken words cannot be spied on. Figure 83 shows an overview of the secure smart microphone architecture.

Currently, the neural network of the smart microphone is using the TensorFlow Lite Micro machine-learning framework and is trained on a Google spoken word dataset containing 35 words [46]. It can recognize a subset of eight different words: “yes”, “no”, “up”, “down”, “left”, “right”, “on”, “off”, and “Marvin” with an accuracy of around 79 percent. In a pre-processing step, slices of the recorded audio waveforms are converted from the time domain into the frequency domain using the short-time fourier transform (STFT). This conversion results in spectrograms showing frequency changes over time and are represented as 2D images. This conversion allows the use of established image recognition machine-learning models to classify audio data. A simple model, consisting of just two layers, with a size of 48 kB fits into the memory of the microcontroller, having the drawback of a lower accuracy compared to more advanced voice recognition architectures, but retaining a better latency due to no communication overhead.

Once the hotwords have been identified, the microcontroller establishes a secure connection to an edge computing node using TLS encryption. Initially it was planned to dispatch the recognized commands to a program running in a secure enclave, bootstrapped by Intel SGX. An enclave is an isolated area of memory which cannot be subverted by high-privileged software, such as the operating system and the hypervisor. The software running in the enclave should have received the voice commands and stored them in an in-memory database. But we encountered unforeseen problems during the implementation: The software development for TEEs is constrained by the absence of the POSIX API and thereby system functions, e.g. “fopen”. But these are required by machine learning frameworks like TensorFlow which were planned for the task of running our neural network models. We removed these functions from the depending libraries and were able to compile TensorFlow into a WebAssembly program. But executing it in a trusted runtime [47], the missing functions led to errors we were not able to resolve.

Nonetheless, in order to implement the speech recognition feature, a decision was made to implement the audio receiver node outside of the trusted zone and without WebAssembly as a requirement. The audio data received by the wireless microphone is now processed using the state-of-the-art speech recognition model Whisper by OpenAI [39], which can generate a transcript of spoken sentences. Although not employing a trusted environment, the data is sent through an encrypted connection, and a possible attacker has to break into the system and gain root privileges to access any audio data in memory. At this point, the system as a whole is already compromised.

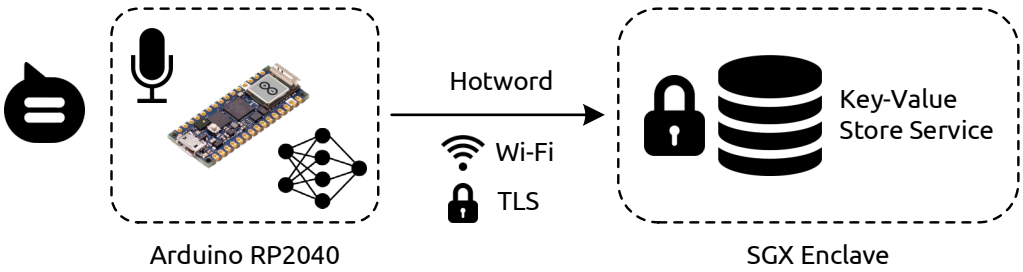


Figure 83: Secure smart microphone architecture overview

5.9 Evaluation of the Key Performance Indicators

The achieved improvements regarding the KPIs are listed below. Some are not trivial to measure and therefore estimated. The baseline system is the smart mirror setup at the beginning of the project, consisting of two AGX Xavier on a t.RECS running the software with 16 FPS by consuming 150 Watt.

The main objective in this project was to increase the energy efficiency by 10x while maintaining 30 FPS and a low latency.

Performance Metrics:

- **Latency:** Calculating the detection of objects, gestures, and faces are all done in 22 ms. The big YOLOv7 represents the biggest bottleneck and was accelerated using EmbeDL.
- **Achieved performance:** The imaged-based detection nodes for face, gesture and objects are running with a performance of 30 FPS. This is the maximum performance of the camera. Depending on the accuracy and hardware accelerator a processing performance from 42 up to 124 FPS can be achieved.

Resource Metrics:

- **Resources:** A number of different hardware realizations has been evaluated
  - t.RECS with 2x Nvidia AGX Xavier (8-Core ARM Carmel, 512-Core Volta, 32 GB RAM)
  - t.RECS with Nvidia AGX Orin (8-core Arm Cortex-A78AE, 1792-core Ampere , 32GB RAM / 12-core Arm Cortex-A78AE, 2048-core Ampere, 64 GB RAM) and a Hailo-8 (26 TOPS, 32 MB SRAM) or versal core Ai (Versal XCVC1902 ACAP).
  - u.RECS with Orin NX (8-core Arm Cortex-A78AE, 1024-core Ampere, 16 GB Ram) and a Hailo-8 (26 TOPS, 32 MB SRAM) or versal edge Ai (Versal XCVE2302 ACAP).
- **Cost:** Listed below, please find the cost of the different setups. In case VEDLIoT hardware the VEDLIoT hardware is utilized, prices reflect the cost per unit for a medium-sized production quantity. For third-party components, prices are based on medium quantities where volume discounts are available.
  - t.RECS with 2x Nvidia AGX Xavier: 2116 €
  - t.RECS with Nvidia AGX Orin: 2803 €
  - t.RECS with Versal XCVC1902/VCK190: 13146 €
  - u.RECS with Orin NX and Hailo-8: 934 €
  - u.RECS with Orin NX and Versal Edge.AI: 1440 €
- **Power / Energy:** The power consumption was decreased from 150 Watt to around 38 Watt and is expected to be decreased to around 28 Watt.

Quality Metrics:

- **Accuracy:** The accuracy of the optimized models are depending on the pruning degree and the training data set. Good results for the YOLOv7 trained with the coco dataset are between the initial 69.2% and 64.4%. For the gesture dataset the accuracy could be increased from 90,7% for the initial YOLOv7-tiny to 93.8% for YOLOv7 version in a comparable size.

Combined Metrics:

- **(Energy) Efficiency:** The latest smart mirror prototype based on the Nvidia AGX Orin and Hailo-8 achieved 1.266 Watt / FPS. The expected value for the Nvidia Orin NX and the Hailo-8 is 0.933 Watt / FPS. The baseline system showed 9.375 Watt / FPS.

6 Conclusion

This document is the third of three deliverables about the development of the four project use cases and the transition from traditional algorithms to a machine learning approach and its optimisation. It presents the final results including comparisons of the optimised metrics and KPIs vs. the baselines and defined goals.

All four use cases have been successfully developed and optimised. Both Industrial IoT use cases have been developed from scratch, although the base idea was already available in similar formats. The AI part of the automotive use case was also developed from scratch while having the research focus of distributed AI in mind. Finally, the Smart Home use case was based on previous projects, but massively improved in terms of performance and features. Also, for all use cases, the defined metrics and KPIs have been measured and evaluated, therefore the Milestone 8 “Final evaluation and benchmarking” is fulfilled.

For both industrial use cases, a systematic workflow has been progressively developed and implemented for the creation of AI-based solutions for cyber-physical systems. This process encompasses problem analysis, data collection and analysis, model training and optimization, and hardware acceleration.

In the motor monitoring use case, we not only integrated deep learning capabilities into the smart end-field device for data processing, but also optimized the hardware toward better energy efficiency. Additionally, a comprehensive IoT system was established for the purpose of demonstration. For the arc fault detection use case, we constructed a real-time DC series arc fault detection system from the ground up and further refined it for optimal performance. By creating a testbench for arc fault simulation and optimizing the data collection procedure, we expanded the variety of datasets, enhancing the model's robustness.

Despite achieving the goal of optimized runtime and energy efficiency, challenges persist in integrating AI into the industrial use cases, particularly for anomaly detection. The accuracy of deep learning models requires improvement, a metric significantly influenced by dataset size and quality. Addressing this issue will be a primary focus for future development in both use cases. This will be achieved by conducting a more in-depth problem analysis and enhancing DL-based algorithms further. This is possible as software optimization and hardware acceleration are poised to facilitate the optimal performance of deep learning algorithms.

The smart home use case significantly improved performance and especially energy efficiency. With this step, realizing a real product is more feasible in the context of living environments. With the new gesture interaction functionality and a local voice assistant, critical points for a pleasant user experience are set. The evaluated safety and security methods are promising to handle any privacy concerns. Further energy efficiency improvements are expected with an Orin NX and an accelerator on the u.RECS platform. This improvement will be made in conjunction with the integration of the edge versal AI module developed within this project. Gamification and more user interaction will be implemented on the software side to get an even better experience, especially with regard to fairs.

The work on the automotive use case within Vedliot has shown that it is possible to divide calculations of a NN between two geographically separated locations with computational nodes. There is a need to do further work regarding the communciation between these locations in order to reduce the latency of the distributed computations. The accuracy of the inference is promising but also needs further work. However, it is important to emphasize

that the goal was to show that distributed calculations are viable, not to completely handle a saftety-critical function. That part of the project was successful, and the implementation of the detection part of the automatic emergency brake function shows promise.

7 References

[1] VEDLIoT EU project, D2.3 - Pilots/use case specification, 2021.

[2] VEDLIoT project, "D7.2 - First report on use case development and optimisation," 2022.

[3] VEDLIoT Project, "D7.3 - Second report on use case development and optimisation," 2022.

[4] Siemens, "Konnektivität für SIMOTICS Niederspannungsmotoren," [Online]. Available: https://www.siemens.com/de/de/produkte/antriebstechnik/digitalisierung-antriebstechnik/konnektivitaet.html. [Accessed 4 1 2024].

[5] VEDLIoT EU project, "Delivereable 7.2 - First report on use case development and optimisation," 2022.

[6] VEDLIoT EU Project, "Deliverable 7.3 - Second report on use case development and optimisation," 2022.

[7] VEDLIoT EU Project, "Deliverable 2.1 - Intermediate report on the methods for requirement, specification, performance metrics and verification of context limited AI processing systems," 2021.

[8] VEDLIoT EU Project, "Delivearble 4.7 - SIEMENS Report on cognitive IoT hardware optimizations," 2024.

[9] MaximIntegratedAI, "ai8x-training," [Online]. Available: https://github.com/MaximIntegratedAI/ai8x-training. [Accessed 04 01 2024].

[10] Siemens AG, "Coaty," Siemens AG, [Online]. Available: https://coaty.io/. [Accessed 04 01 2024].

[11] MongoDB, "MongoDB," MongoDB, [Online]. Available: https://www.mongodb.com/de-de. [Accessed 04 01 2024].

[12] S. Lu, B. Phung and D. Zhang, "A comprehensive review on DC arc faults and their diagnosis methods in photovoltaic systems," *Renewable and Sustainable Energy Reviews,* vol. 89, pp. 88-98. 10.1016/j.rser.2018.03.010, 6 2018.

[13] R. Weiss, L. Ott and U. Boecke, "Energy efficient low-voltage DC-grids for commercial buildings," *IEEE First International Conference on DC Microgrids (ICDCM),* pp. 154-158, 2015.

[14] UL Standard 1699B, "UL 1699B: Standard for Safety of Arc-Fault Circuit-Interrupters," Underwriters Laboratories, Northbrook, IL, USA, 2018.

[15] EmbedL, "Embedl," Embedl, [Online]. Available: https://www.embedl.com/. [Accessed 04 01 2024].

[16] N. HOTZ, "What is CRISP DM?," [Online]. Available: https://www.datascience-pm.com/crisp-dm-2/. [Accessed 15 12 2023].

[17] NVIDIA, "Jetson Xavier NX-Serie," [Online]. Available: https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-xavier-nx/. [Accessed 04 01 2024].

[18] S. Lu, A. Sahoo, R. Ma and B. Phung, "DC Series Arc Fault Detection Using Machine Learning in Photovoltaic Systems: Recent Developments and Challenges," in *8th International Conference on Condition Monitoring and Diagnosis (CMD)*, 2020.

[19] E. Denney, G. Pai, R. Berthold, M. Fladeland, B. Storms and M. Sumich, "Assuring ground-based detect and avoid for UAS operations," in *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, Colorado Springs, CO, USA, 2014.

[20] NVIDIA, "NVIDIA TensorRT," [Online]. Available: https://developer.nvidia.com/tensorrt. [Accessed 04 01 2024].

[21] D. Ö. Andreas Ask, "D6.5 Report on merging, distributing and sharing of resources," 2024.

[22] R. Griessl, "D4.1 First report on cognitive IoT hardware platform and microserver development," 2022.

[23] Oxford Technical Solutions, "RT User Manual," 20 02 2020. [Online]. Available: RT User Manual. [Accessed 22 12 2023].

[24] Magna Electronics, "Magna Enhances ADAS Capabilities by Joining 5G Innovation Program," 03 12 2023. [Online]. Available: https://www.magna.com/stories/news-press-release/2023/magna-enhances-adas-capabilities-by-joining-5g-innovation-program. [Accessed 22 12 2023].

[25] iPerf3, "iPerf - The ultimate speed test tool for TCP, UDP and SCTP," [Online]. Available: https://iperf.fr/. [Accessed 22 12 2023].

[26] LEGaTO project, "LEGaTO project," [Online]. Available: https://legato-project.eu/. [Accessed 29 04 2022].

[27] A. B. e. al., "Simple Online and Realtime Tracking," *CoRR,* vol. abs/1602.00763, 2016.

[28] M. Teeuw, "MagicMirror² project," 2016. [Online]. Available: https://magicmirror.builders/.

[29] Babarcelona Supercomputing Center, "The OmpSs Programming Model," [Online]. Available: https://pm.bsc.es/ompss.

[30] H. V. N. a. L. Bai, "Cosine Similarity Metric Learning," in *Asian conference on computer vision*, Springer, 2010, pp. 709--720.

[31] Google, "Google MediaPipe for Developer," [Online]. Available: https://developers.google.com/mediapipe.

[32] VEDLIoT project, "D6.4 Final report on requirements and resource aware optimization," 2023.

[33] M. M. S. B. L. B. R. G. J. H. P. P. D. R. C. L. Z. P. D. Tsung-Yi Lin, "Microsoft {COCO:} Common Objects in Context," *CoRR,* vol. abs/1405.0312, 2014.

[34] VEDLIoT project, "D3.4 Final report on the DL accelerator design," 2024.

[35] Xilinx, "VCK190 Evaluation Board User Guide (UG1366 v1.0)," 2021.

[36] "github - fast GICP Algorithmen," [Online]. Available: https://github.com/SMRT-AIST/fast_gicp.

[37] coqui, "coqui," coqui, [Online]. Available: https://coqui.ai/.

[38] alphacephei, "vosk - speech recognition toolkit," [Online]. Available: https://alphacephei.com/vosk/.

[39] OpenAI, "Introducing Whisper," [Online]. Available: https://openai.com/research/whisper.

[40] coqui, "coqui Text to Speech," [Online]. Available: https://github.com/coqui-ai/TTS.

[41] "pyttsx3," [Online]. Available: https://github.com/nateshmbhat/pyttsx3.

[42] VEDLIoT project, "D5.4 Integrated and extended Security, Safety and Robustness mechanisms and tools," 2024.

[43] E. AI, "Spacy website," [Online]. Available: https://spacy.io/.

[44] "Rasa," [Online]. Available: https://rasa.com/.

[45] A. Foundation, "Documentation - Nano RP2040 Connect," [Online]. Available: https://docs.arduino.cc/hardware/nano-rp2040-connect.

[46] P. Warden, *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition,* arXiv e-prints, arXiv: 1804.03209, 2018, pp. arXiv e-prints, arXiv: 1804.03209,.

[47] J. Ménétrey, M. Pasin, P. Felber and V. Schiavoni, "Twine: An Embedded Trusted Runtime for WebAssembly," in *ICDE'21: Proceedings of the 37th IEEE International Conference on Data Engineering*, Chania, Greece, 2021.

[48] A. Jani, "Maxim Showcases Efficient Custom AI," *Microprocessor Report,* 2021.

[49] L. Gwennap, "XMOS Xcore.ai Adds Vector Unit," *Microprocessor Report,* 2020.

[50] B. Wheeler, "RISC-V Enables IoT Edge Processor," *Microprocessor Report,* 2018.

[51] B. Wheeler, "Bitmain SoC Brings AI to the Edge," *Microprocessor Report,* 2019.

[52] M. Demler, "Syntiant NDP120 Sharpens Its Hearing," *Microprocessor Report,* 2021.

[53] L. Gwennap, "Intel Gains Myriad Customers," *Microprocessor Report,* 2018.

[54] M. Demler, "Coherent Logix Configures Edge AI," *Microprocessor Report,* 2020.

[55] M. Demler, "Hailo Illuminates Low-Power AI Chip," *Microprocessor Report,* 2019.

[56] M. Demler, "Blaize Ignites Edge-AI Performance," *Microprocessor Report,* 2020.

[57] M. Demler, "Flex Logix Moves Into Chips," *Microprocessor Report,* 2019.

[58] N. Developers, "Nvdla primer," 2021. [Online]. Available: http://nvdla.org/primer.html.

[59] T. Moreau, "A hardware-software blueprint for flexible deep learning specialization," *arXiv:1807.04188,* 2019.

[60] X. Zhang, "an Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs," 2018.

[61] [Online]. Available: https://github.com/IBM/AccDNN.

[62] M. Demler, "Vsora Drives to Deliver Petaflops," *Microprocessor Report,* 2020.

[63] M. Demler, "Andes plots RISC-V vector heading," *Microprocessor Report,* 2020.

[64] M. Demler, "Ceva NeuPro accelerates neural nets," *Microprocessor Report,* 2018.

[65] M. Demler, "Imagination Series4 tiles tensors," *Microprocessor Report,* 2020.

[66] M. Demler, "Ceva SensPro2 Doubles AI Throughput," *Microprocessor Report,* 2021.

[67] "DPUCZDX8G for Zynq UltraScale+ MPSoCs," [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/dpu/.

[68] M. Demler, "Think Silicon Spins AI Accelerator," Microprocessor Report, 2020. [Online].

[69] A. Jani, "SiFive VIU7-256 Takes Vector Lead," *Microprocessor Report.*

[70] A. Jani, "SiFive Brings Vectors to S-Series," *Microprocessor Report,* 2021.

[71] M. Demler, "Cortex-M55 Supports Tiny-AI Ethos," *Microprocessor Report,* 2020.

[72] L. Gwennap, "Cortex-A55 Improves Memory," *Microprocessor Report,* 2017.

[73] L. Gwennap, "Cortex-A75 Has DynamIQ Debut," *Microprocessor Report* , 2018.

[74] L. Gwennap, "Arm Dot Products Accelerate CNNs," *Microprocessor Report* , 2018.

[75] J. Roesch. [Online]. Available: URL: http://dx.doi.org/10.1145/3211346.3211348.

[76] "Apache Software Foundation," [Online]. Available: https://tvm.apache.org/docs/.

[77] G. Developers, "FlatBuffers Documentation," [Online]. Available: https://google.github.io/flatbuffers/index.html.

[78] G. Developers, "TensorFlow Lite Documentation," [Online]. Available: https://www.tensorflow.org/lite/guide?hl=en.

[79] A. Demidovskij, "learning workbench: comprehensive analysis and tuning of neural networks inference.," 2019.

[80] W.-F. Lin, " Onnc: a compilation framework connecting onnx to roprietary deep learning accelerators.," in *International Conference on Artificial Intelligence Circuits and Systems* , 2019.

[81] W.-F. Lin, "Onnc-based software development platform for configurable nvdla designs," in *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2019.

[82] N. Rotem, "Graph Lowering Compiler Techniques for Neural Networks," 2019.

[83] L. Gwennap, "Kneron Delivers Efficient AI," *Microprocessor Report,* 2020.

[84] L. Gwennap, "Kneron KL720 Boosts Efficiency," *Microprocessor Report,* 2020.

[85] L. Gwennap, "GreenWaves GAP9 Goes Faster," *Microprocessor Report,* 2020.

[86] L. Gwennap, "Kendryte Embeds AI for Surveillance," *Microprocessor Report,* 2019.

[87] M. Demler, "Syntiant NDP120 Sharpens Its Hearing," *Microprocessor Report,* 2021.

[88] L. Gwennap, "LeapMind jumps on binary networks," *Microprocessor Report,* 2020.

[89] Nvidia, "Datasheet of Nvidia Jetson Xavier NX," [Online]. Available: https://developer.download.nvidia.com/assets/embedded/secure/jetson/Xavier%20NX/DG-09693-001_v1.5.pdf. [Accessed 19 01 2022].

[90] SGeT - Standardization Group for Embedded Technologies e.V., "SMARC Standard," [Online]. Available: https://sget.org/standards/smarc/. [Accessed 19 01 2022].

[91] Espressif, "Espressif IoT Development Framework," [Online]. Available: https://github.com/espressif/esp-idf. [Accessed 19 01 2022].

[92] PlatformIO labs, "Platform IO Website," [Online]. Available: https://platformio.org/. [Accessed 19 01 2022].

[93] B. Wunder, R. Weiss, L. Ott, M. Szpek and U. Boeke, "Energy efficient DC-grids for commercial buildings," *IEEE 36th International Telecommunications Energy Conference (INTELEC),* pp. 1-8, 2014 .

[94] EUROPEAN NEW CAR ASSESSMENT PROGRAMME, "Test protocol - AEB/LSS VRU systems," 2021.

[95] Ioffe, S., & Szegedy, C., "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *International conference on machine learning,* pp. 448-456, June 2015.

[96] ROS project, "ROS - Robot Operating System," [Online]. Available: https://www.ros.org/. [Accessed 29 04 2022].

[97] M. Demler, "Ethos-N78 Boosts AI Efficiency," *Microprocessor Report,* 2020.

[98] S. Lu, B. Phung and D. Zhang, "A comprehensive review on DC arc faults and their diagnosis methods in photovoltaic systems," *Renewable and Sustainable Energy Reviews,* pp. 88-98, Volume 89 2018.

[99] O. S. R. Foundation, "ROS - Robot Operating System," [Online]. Available: https://ros.org/.

[100] coqui, "coqui Speech to Text," [Online]. Available: https://github.com/coqui-ai/STT.

[101] NVIDIA, "Nvidia TensorRT Overview," [Online]. Available: https://developer.nvidia.com/tensorrt.

[102] VEDLIoT EU Project, "Deliverable 3.3 - Evaluation of the DL accelerator designs," 2022.

[103] Xilinx, “DPUCVDX8G for Versal ACAPs Product Guide (PG389 v1.1),” 2022.

8 List of Figures