D5.3

Machine Learning based Detection of Attacks into Anti-Hacking Device

Topic	SU-ICT-01-2018
Project Title	Artificial Intelligence-based Cybersecurity for Connected and Automated Vehicles
Project Number	833611
Project Acronym	CARAMEL
Contractual Delivery Date	M18
Actual Delivery Date	M19
Contributing WP	WP5
Project Start Date	01/10/2019
Project Duration	30 Months
Dissemination Level	Public
Editor	ALTRAN
Contributors	0INF, 8BELLS, AVL, DT-Sec, I2CAT, PANA, SID, UCY, UPAT

Document History			
Version	Date	Modifications	Source
1	22.06.2020	Initial Table of Contents	Altran
2	07.04.2021	First Draft Version	All Partners
3	13.04.2021	Updated version for Internal Review	All Partners
4	29.04.2021	Updated version based on feedback received for reviewers.	All Partners

DISCLAIMER OF WARRANTIES

CARAMEL has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 833611. The content of this deliverable does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).

DISCLOSURE STATEMENT

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronyms and abbreviations

Acronym	Meaning
AdaFM	Adaptive Feature Modification
AHD	Anti-Hacking Device
AI	Artificial Intelligence
API	Application Programming Interface
ARM	Advanced RISC Machines
ASIC	Application Specific Integrated Circuit
AT	Authorization Tickets
CARLA	Computerized Approach to Residential Land Analysis / Open-source simulator for autonomous driving research.
CAVs	Connected and Autonomous Vehicles
CCAM	Cooperative, Connected and Automated Mobility
CNTK	Microsoft Cognitive Toolkit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DSOs	Distribution system operators
ECDF	Empirical Cumulative Distribution Function
EKF	Extended Kalman Filter
ETSI	European Telecommunications Standards Institute
EV	Electrical Vehicle
FPS	Frame per Second
GFX	GreenFlux
GPS	Global Positioning System
GPU	Graphics Processing Unit
HW	Hardware
ICT	Information Communication Technology
IDS	Intrusion Detection System
IIoT	Industrial Internet of Things
IoT	Internet of Things
ITS	Intelligent Transport Systems
KVM	Kernel-based Virtual Machine
LIDAR	Light Detection and Ranging
LSTM	Long Short-Term Memory
ML	Machine Learning
OBD	On-Board Unit
PDE	Partial Differential Equations
PDS	Parallel Detection Stream
PKI	Public Key Infrastructure

ROF Model	Rustin-Osher-Fatemi Model
SIEM	Security Information Exchange Protocol
SoC	System on a Chip
SoO	Signals of Opportunity
SSD	Single Shot MultiBox Detector
TCN	Temporal Convolutional Networks
Tflite	Tensorflow Lite
TOPS	Tera Operations per Second
TPU	Tensor Processing Unit
TSFC	Two Stage Fully Connected
TSOs	Transmission system operators
TV	Total variation
V2V	Vehicle to Vehicle
V2X	Vehicle to Everything
VANET	Vehicular Ad Hoc Network
WP	Work Package

Executive Summary

This deliverable will provide a summary of machine learning (ML) and other heuristics algorithms developed as part of CARAMEL for threat detection and their deployment into the anti-hacking device (AHD), so that the AHD can be integrated as an additional controller into the vehicle for passive real-time threat detection in the vehicle. Real-time means as sufficiently fast to be useful in the chosen scenarios. The threat detection algorithms were developed as part of WP3 and WP4 activities of CARAMEL, hence detailed information about the algorithms can be found in D3.2 [1], D3.4 [2], D3.6 [3], D3.7 [4], D4.2 [5] and D4.3 [6].

This deliverable will discuss a general approach for deployment of threat detection algorithms into the AHD, based on the Docker technology. It also provides details on the hardware (HW) platforms identified for development of the AHD i.e. GOOGLE CORAL and NVIDIA JETSON, in CARAMEL. It provides a summary of most of the threat detection algorithms developed in CARAMEL and explains the deployment of these algorithms into the AHD. The deployment process describes real-time preprocessing such as data cleaning, data aggregation, and data transformation of the respective algorithms. It also details the implementation of algorithms into the AHD, i.e. model deployment & Docker configuration. Its further documents time-based model performance using different hardware platforms for the algorithms.

Based on different use case specifications, there are a few algorithms which will not be deployed in the AHD and a couple of them utilizing a simulated AHD for threat detection.

1 Introduction

1.1 Document Scope

The scope of the document is to summarize ML and other heuristics algorithms for attack detection being developed as part of WP 3 and WP 4 in CARAMEL and their deployment strategy into the AHD by focusing on pre-processing, implementation and performance measurement for the algorithms. Pre-processing activity shall cover the real time processing of data that includes data cleaning, data aggregation and data transformation. Implementation of ML algorithms into the AHD shall include the deployment of the models, Docker creation and configuration. Finally, performance measurement shall focus on the performance metrics of the ML algorithms implemented on the hardware platforms. The document will introduce the Docker system as it is one of the key technologies utilized for implementation of different algorithms into the AHD. The document shall provide a high-level description of the different hardware platforms and deep learning framework for the AHD.

1.2 Document Structure

This document is structured as follows: Section 2 provides a brief overview of the CARAMEL project. Section 3 provides a general approach to deploy ML algorithms into the AHD. The section introduces Docker containers which will be utilized for implementation of ML algorithms, details on deep learning framework used and information on hardware platforms for the AHD. Section 4 provides a summary of different ML and other heuristics algorithms that have been developed for CARAMEL use cases in WP 3 and WP 4. The section also covers the deployment process of different algorithms, which includes the aspect of real-time processing of data in the vehicle, implementation into the AHD and performance measurement.

2 CARAMEL Overview

2.1 Overview

Automotive vehicles today have evolved from a simple mechanical machine to a complex combination of digital-mechanical machines with different software's managing different functions, an internal network for communication between vehicle components and an external connectivity network for more advanced functions. As they are becoming more digitized with external connectivity's, the possibilities of Cyber Attacks have increased tremendously with extreme damaging effect. Hence, CARAMEL's objective is to address modern vehicle Cyber Security challenges proactively by also employing Artificial Intelligence (AI) & ML techniques and to explore methods to mitigate the associated safety risks.

The automotive industry is strongly moving towards safe, green, and intelligent transportation, however, to secure the cooperative, connected and automated mobility (CCAM) environment they would need to capitalize on the ICT sector due to the maturity achieved by them in Cyber Security over the years. Even though a lot of Cyber Security approaches & techniques can be reused by the automotive industry, the solutions would still need to be adapted to the environment as it is very different to information communication technology (ICT) environment. CARAMEL foresees 3 key areas where Cyber Security innovations are required: i) Autopilot, ii) vehicle to everything (V2X) communications and iii) Electrical Charging stations, and so the use cases developed as part CARAMEL encompasses the technologies in these key areas.

CARAMEL intends to follow the layered approach with the notion of Defence in Depth, 1st line of defence would be to extend and integrate advanced security technologies to secure the attack surface for i) Autonomous, ii) Connected and iii) Electro Mobility vehicles. 2nd line of defence would be to design & develop tailored intrusion detection and prevention tools. Finally, 3rd line of defence would be to have a software solution for monitoring, detection and sending warnings/alarms. Figure 1 reminds of the "architecture" of CARAMEL and its scope.

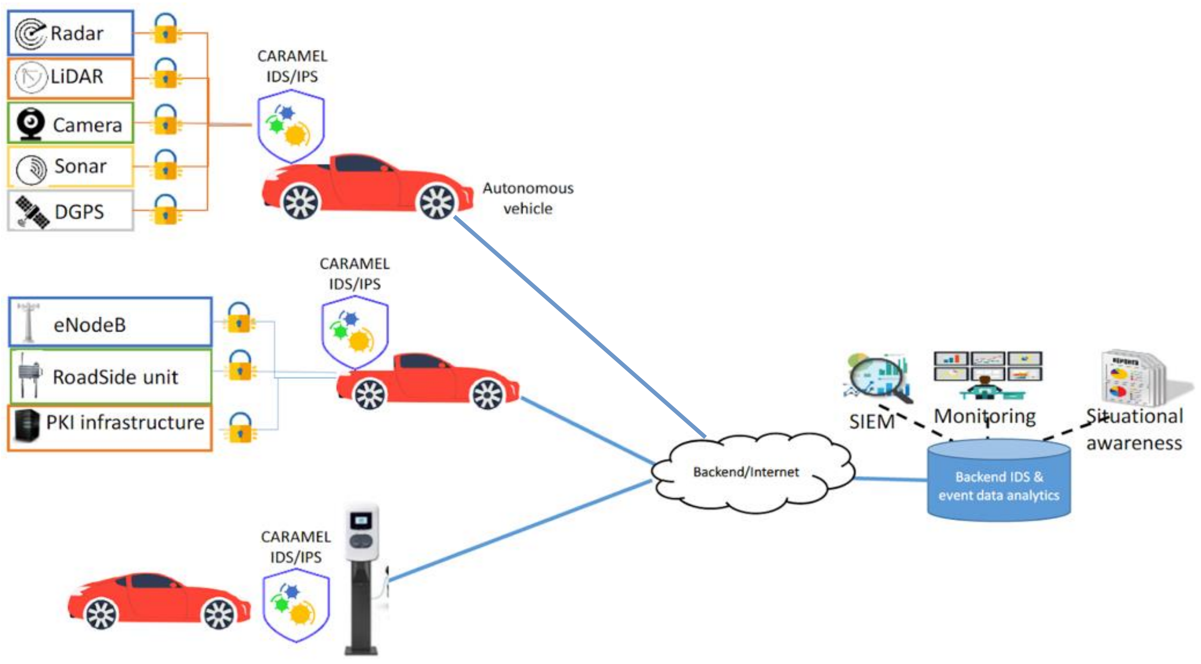


Figure 1: CARAMEL High-Level Architecture

3 General approach of deploy ML-Algorithms in Anti-Hacking Device

The CARAMEL AHD is designed as a passive intrusion detection device that is integrated as an additional controller into the vehicle. The AHD *passively* listens to the car's internal busses and systems, processes and aggregates raw data from sensors and communication controllers and uses ML and other heuristics to detect possible attacks against the vehicle's systems.

It then *actively* creates attack reports (events) and sends them to the CARAMEL backend. Details of the integration of the AHD into the different CARAMEL scenarios are described in the CARAMEL specification deliverable D2.4 [7].

The AHD needs to be updated very frequently to run updated attack detection algorithms to counter newly discovered attack vectors. This requires frequent updates of the AHD firmware and application load. From a vehicle safety perspective any corruption of the AHD by bad actors must be avoided at all costs.

3.1 Hardware platforms

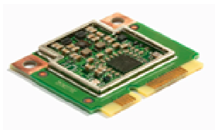
AI and ML have progressively changed the paradigms of interacting with the environment. For years, the impulse achieved by AI tasks was strongly supported by the enormous parallel computing capabilities of graphics processing units (GPUs) which were high power consumption devices. Due to the new technology revolution around internet of things (IoT), Automotive digitalization, industrial internet of things (IIoT), etc., the demand for use of low power consumption devices have grown. Now with platforms like Google Coral [8], Nvidia Jetson [9], Neural Compute Stick [10], etc., the ability to run AI / ML algorithms on low power consumption devices has been achieved. The next subsections discuss two of the hardware platforms selected in CARAMEL i.e. Google Coral Platform and Nvidia Jetson Platform.

3.1.1 Google Coral Platform

The Coral platform is an Application Specific Integrated Circuit (ASIC) also known as edge tensor processing unit (TPU). The edge TPU was developed, taking as a basis the TPU architecture used in Google's servers, which enables ML inferences in a more effective process than the inference carried on GPUs or central processing unit (CPUs). The edge TPU is a small ASIC that can perform advanced ML applications at high inferences speed with low energy consumption.

The specifications demonstrate that a single edge TPU device can compute four tera operations per second (TOPS) with an energy requirement of 0.5 watts per TOPS [11]. In addition to having achieved a robust and compact product for AI applications, the system architecture is flexible since the co-processor is available for a wide range of systems. Some of the available CORAL products are mentioned in Table 1.

Table 1: Tech specifications of available CORAL hardware devices

Device	Name	Connector	Dimensions	Reference
	USB Accelerator	USB 3.0 Type-C* (data/power)	65 mm x 30 mm	[12]
	**Mini PCIe Accelerator**	Half-size Mini PCIe	30 x 26.8 mm	[13]
	M.2 Accelerator A+E key	M.2 (A+E key)	22 mm x 30 mm (M.2-2230-A-E-S3)	[14]
	Dev Board	On-chip	88 mm x 60 mm x 24mm	[15]

The available CORAL devices achieve a simple and scalable prototype since the models compiled for specific architectures can be later moved to new systems despite the differences in hardware such as computers or embedded systems. All this scalability is possible by virtue of the functionalities built-in Keras and TensorFlow [11].

TensorFlow is Google’s platform for the development and deployment of ML algorithms in a wide range of systems from embedded systems, mobile devices such as mobile phones, consumer laptops and PC’s up to large distributed systems composed of large arrays of CPUs and GPUs [16].

Keras is an Application Program Interface that achieves the development and deployment of ML models straightforwardly, reducing the amount of code required for the most common functionalities needed for ML tasks [17]. The application programming interface (API) is provided for Python.

TensorFlow is the provider of the primitive low-level functionalities required for creation of ML tasks. At the same time, Keras provides a user-friendly set of high-level neural networks library built-in for different ML frameworks such as TensorFlow, Microsoft Cognitive Toolkit (CNTK), Theano.

TensorFlow Lite (TFlite) is an open-source deep learning framework for on-device inference. The CORAL pipeline requires the use of TFlite for the purpose of deploying ML models on mobile and IoT devices. This restriction is due to the requirements of mobile devices where the limit to their performance is usually the amount of available memory and the speed of information processing. Therefore, TFlite focuses on two essential components: a small binary size and low latency [18]. To overcome previous restrictions, the TFlite library is composed of an interpreter that helps execute inferences of already optimized models on different HW with constrained performance. Besides, it contains a converter that includes a set of functions to export full TensorFlow models into a smaller version compatible with the interpreter.

3.1.1.1 *Deployment*

The process for the deployment of AI / ML algorithms using TFlite as shown in Figure 2:

1. Take a base model and train it with a clean dataset.
2. The trained model must be converted in order to be compatible with the targeted supported devices.
3. Deployment into the device.
4. Further optimization of the model can be performed to increase the efficiency with a smaller memory footprint and maintaining the accuracy of the model.

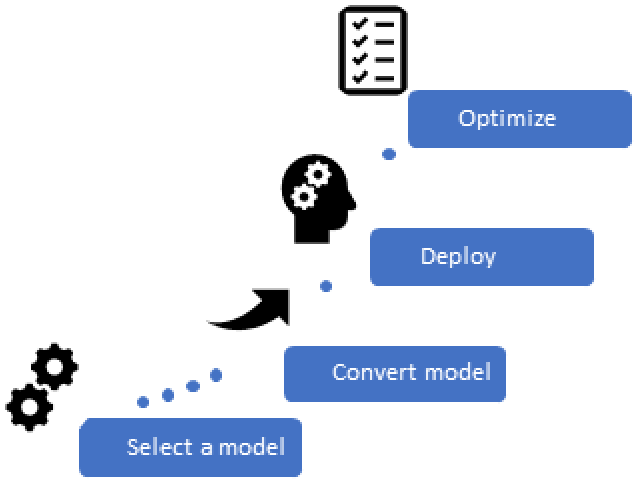


Figure 2: Dataflow for deployment of a model.

Before the deployment of models in an edge TPU co-processor, a quantization process must be effectuated. The quantization aims to convert all 32-bit ML model's parameters which encode a specific task, namely weights and activation functions, into a lower data type, with a negligible impact on its accuracy. In this case, the CORAL platform fully supports 8 bit. There are two different quantization techniques already provided by TensorFlow Model Optimization. In both cases reducing the floating-point of 32 bits to only 8 bit already means a reduction of 4 times the original size, while the improvements in latency are also considerable.

1. Quantization aware training: This technique enables training the predefined model using only low precision datatypes. Quantization aware allows the training of a neural network in a way that emulates the results during inference, that is, it produces weights of only 8 bits which leads to a lower size footprint of the model, achieving reaching higher inference speeds. It has been proven that the performance when training these models is comparable to using a high precision training technique.
2. Post-training quantization: This technique requires an already ML model trained with high-precision data types. The quantization will reduce the number of bits needed per stored weights on the ML model down to 8 bits. This process is usually simple since it bases its workflow on a previously trained model. However, a requirement for this technique is the necessity of having access to the training dataset to compute statistics from a representative dataset.

3.1.2 NVIDIA Jetson Platform

NVIDIA Jetson is a series of embedded computing boards from NVidia. The Jetson models all carry a Tegra processor (or system of a chip (SoC)) from NVIDIA that integrates an advanced RISC machines (ARM) architecture CPU. Jetson is a low-power system and is designed for accelerating ML applications (see Table 2).

Table 2: Tech specifications of available NVIDIA Jetson hardware devices

Device	Name	Connector	Dimensions	Reference
	Jetson nano development kit	On-chip	69.6 mm x 45 mm	[19]
	Jetson AGX	On-chip	10.67 x 10.67 x 10.16 cm	[20]

In the project we will use several NVIDIA Jetson devices where the performance and cost is optimized by the CARAMEL partners for the use case at hand. In the ML lab in Berlin the top-of-line NVIDIA Jetson AGX device is available to project partners for integration testing.

APIs, hardware, and also Docker images for the different NVIDIA Jetson devices are mostly compatible (apart from some ML resource considerations), so that an application designed for a lower-spec NVIDIA Jetson device will all work on a higher-specced device.

3.2 Introduction to Docker

Docker brings the concept of apps to the server. A Docker container encapsulates all resources an application needs to run. This means that all dependencies such as shared libraries, helper programs, and static data are combined into an image file that contains a complete Linux operating system but without the Linux kernel. Docker containers can be instantiated from this image, but they use the underlying Linux kernel without emulating a whole hardware environment.

This approach uses less resources than full hardware virtualization as has been made popular by software from VMWare or the kernel-based virtual machine (KVM) virtualizer native to Linux. Still, containers are separated from each other and the host operating system. This is enabled by Linux features such as namespaces and cgroups.

Docker container is started from a container image that is just a file, easily being transported from one host to another. Installation, update, and copying is just reduced to simple file operations.

Any number of Docker image can be instantiated from a container image since write operations are local to one container and do not affect other containers derived from the same image. For many applications ready-made container images are available e.g. from the Docker hub that can be started immediately. The configuration for these ready-made images is prepared by the creator of the image as much as possible. Variable parts are set up automatically upon container start-up.

An important advantage of containers is the good scalability. If new instances of an application are needed it is just necessary to start new containers. After stopping them the resources are quickly deallocated again.

Additionally, it is easy to update Docker images and therefore the embedded application: In the Docker world all non-static data and configuration are stored outside of the container in Docker volumes. The updated container can just be started with the old data and configuration. If some functions are not working properly, the operator can also easily rollback to the old version of the container. This improves availability and agility when working containers compared to the traditional model of direct installation on the host system. It also facilitates totally new development and deployment models such as devops.

3.3 Benefits of Docker based deployment

The AHD encapsulates the actual detection algorithms and also some system services into Docker containers. This deployment scheme has several advantages like the ability of updating the detection algorithms without a full firmware update. Additionally, the detection algorithms are separated by the protections offered by the Docker runtime against any mutual interference. As a last security measure, the AHD only accepts signed Docker images from pre-defined trusted sources, effectively also implementing multi-level security for Docker implementation on the AHD.

To this end a Docker registry based on the Harbor open source software registry will be provided for the project by DT-Sec that offers secure management of signed Docker images.

4 ML-Algorithm for Attack Detection

This section provides a brief introduction of the different scenarios created by CARAMEL. Furthermore, the section provides a summary of different algorithms a few based on ML that have been developed for CARAMEL use cases in WP 3 and WP 4. The section also covers the deployment strategy of some of the algorithms into the AHD by focusing on pre-processing, implementation and performance measurement for the algorithms. Pre-processing activity covers the real time processing of data that includes data cleaning, data aggregation and data transformation. Implementation of ML algorithms into the AHD include the deployment of the models, Docker creation and configuration. Finally, performance measurement focuses on the performance metrics of the ML algorithms implemented on the hardware platforms.

As part of WP6, the AHD will be integrated into the Panasonic Car by attaching the CAN bus, listening to all relevant data exchange and system variables. Attacks detected by the AHD will be sent to CARAMEL Backend Solution, which is described in D4.4 [62].

4.1 Physical Adversarial Attacks

4.1.1 Introduction of Physical Adversarial Attacks

Deep Learning solutions are increasingly becoming common in Connected and Autonomous Vehicles (CAVs). Over the last years, Deep Learning has produced state-of-the-art and human competitive performance on numerous computer vision and other related tasks to autonomous vehicles [21]. Nevertheless, in the last few years a number of researchers have shown that the Deep Learning architectures are vulnerable to various vision-based attacks [22]. These types of attacks can result in unexpected and potentially dangerous behaviours on autonomous vehicles. For example, a physical modification on the external environment such as traffic signs can lead autonomous vehicles to perform unexpected actions [23].

As described in the use case WP2.1 [24] under section 2.2.1, the physical adversarial attacks are the type of attacks or modifications to a physical entity outside the CAVs system. These types of attacks are especially targeted to Deep Learning models. In the use case 2.1, the traffic signs were selected as a suitable candidate for demonstration of the physical attacks. Following are the few notable reasons why the traffic signs were chosen:

- The relative visual simplicity of road signs.
- Road signs exist in a noisy unconstrained environment with challenging physical conditions such as the weather, lighting, distance and camera point of view (i.e., angle),
- Road signs play an important role in transportation safety.
- A reasonable threat model for transportation is that an attacker might not have control over a vehicle’s systems but is able to modify the objects in the physical world that a vehicle might depend on to make crucial safety decisions.

4.1.1.1 *Summary of ML-Algorithm*

A number of Deep Learning architectures has been researched and developed to tackle traffic signs’ attacks in the CARAMEL project. The architectures can be divided into three parts a) traffic signs anomaly detection b) traffic sign reconstruction and finally c) traffic signs recognition. In this scenario, it is assumed that the traffic detector (the Deep Learning model which can locate the traffic signs in the given scene) is not affected by the attacks. In Figure 3, the flow chart shows the entire pipeline for the physical Adversarial Attacks. Table 3 shows the short summary of the ML Algorithms, for more details about the developed ML solution please refer to D4.2 [5].

Table 3: Brief description of the algorithms developed for anomaly detection and mitigation pipeline.

Type of Algorithms	Description
Traffic Signs Detection and Localisation	Traffic signs detection and localisation is the first step in capturing traffic signs in the scene. For the purpose of physical adversarial attack, the popular Mobilenet Single Shot MultiBox Detector (SSD) v2 [25] was retrained with synthetic data generated from CARLA simulator [26] to detect the traffic signs.
Traffic Signs Anomaly Detection	The primary goal of the traffic sign anomaly detection is to identify if the traffic signs that were detected are tampered. As shown in Figure 4, when the traffic signs are normal, the model sends the signs to the traffic signs recognition model. Whereas, if traffic signs are attacked, it can alert the user as well as send the tampered signs to be reconstructed. The Figure 4, shows the traffic sign anomaly detection in action when the attack is present, demonstrating that it is able to detect the abnormal traffic signs as well as to highlight the location of the signs. The localisation of the attacked signs is an essential part of the model as there can be multi traffic signs in the scene and localising the signs can help to identify the exact location.
Traffic Signs Reconstruction	The traffic signs reconstruction is an effort to robustify the physical adversarial attacks detection model. The core task of the model is to reconstruct attack free meta-traffic signs when an attack is found by an anomaly detection model. The newly reconstructed traffic signs are then passed to the traffic signs recognition model. The Figure 3, shows the condition in which the traffic signs reconstruction is activated whereas Figure 4, shows the performance of the reconstruction model.
Traffic Signs Recognition	The traffic sign recognition model recognises the type of the traffic signs i.e. the label of the traffic signs. In Figure 4, it can be seen in both images that the traffic recognition model is able to identify that the particular sign is a stop sign. The robustification of the pipeline is clearly visible as even after the attack has been detected, the reconstruction model is able to reconstruct the traffic sign and the recognition model is able to correctly display the label.

4.1.1.2 *Deployment into Anti-Hacking Device*

The AHD in the CARAMEL project represents a device which is secure and easy to integrate in the CAVs system. The CARAMEL project utilises a simulated environment to demonstrate the attacks and the related scenarios. Nevertheless, the developed solution has been developed in such a way that it can also run in the AHD. The deployment of the algorithm is performed in two phases:

- **Simulation environment** - CARLA simulator [26] is used to simulate the synthetic environment. CARLA (computerized approach to residential land analysis) is an open-source simulator for autonomous driving research. This process is performed outside the AHD, similar to the process of capturing data of a real-world environment with a camera in an autonomous vehicle. The detection of the traffic signs is also provided by third party tools such as the MobileNet SSD v2 [25] Deep Neural network.
- **Physically adversarial attack detection system** - As shown in Figure 3, mainly the anomaly detection and robustification pipeline is deployed in the AHD.

4.1.1.2.1 Pre-processing Activities

Number of algorithms are used for the pre-processing of the detected traffic signs. The primary reason for pre-processing data is to fit with data format and other requirements of the Deep Learning model. Normalisation, resize, cropping and histogram equalisation are some pre-processing methods which are applied to the captured traffic signs before being fed into the anomaly detection model.

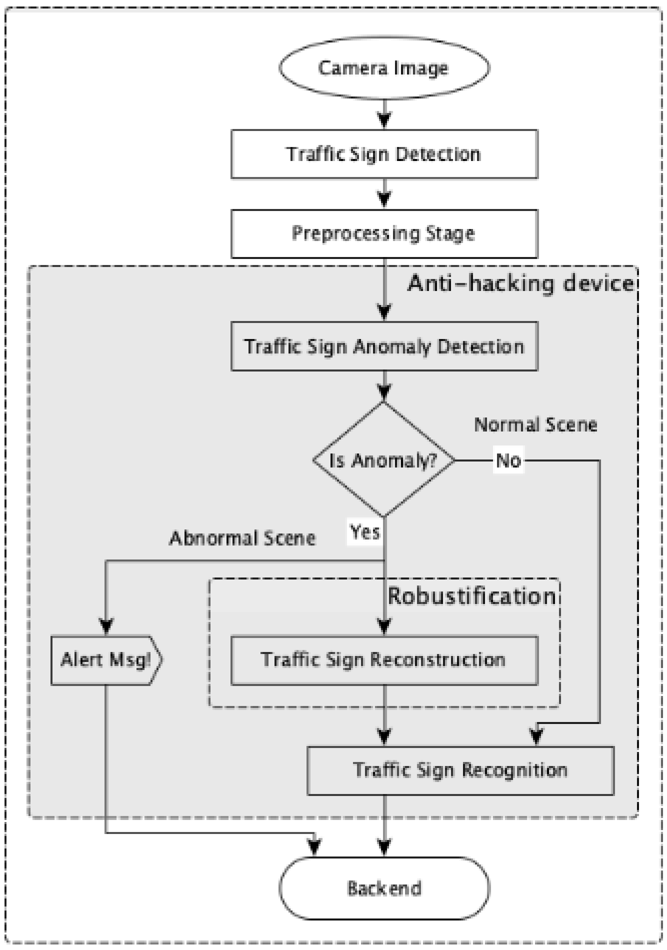


Figure 3: Flowchart of the traffic sign anomaly detection and mitigation pipeline.

- **Normalisation** - Normalisation has a range of meaning in statistics. However, the most common form of normalisation is the feature scaling. Feature scaling is the method of scaling the intensity into a range between 0 and 1 range.
- **Resize** - Resize is the processing of converting data into certain dimensions. In the case of image data, the resize process usually means scaling width and height of the image.
- **Cropping** - Cropping is a process of cutting or splitting data into smaller chunks. In image data, unlike resize which reshapes the entire image, the cropping process extracts a desired shape from the image.
- **Histogram Equalisation** - Histogram equalisation is an image processing method for adjusting the contrast of the image using the image histogram of the given image.

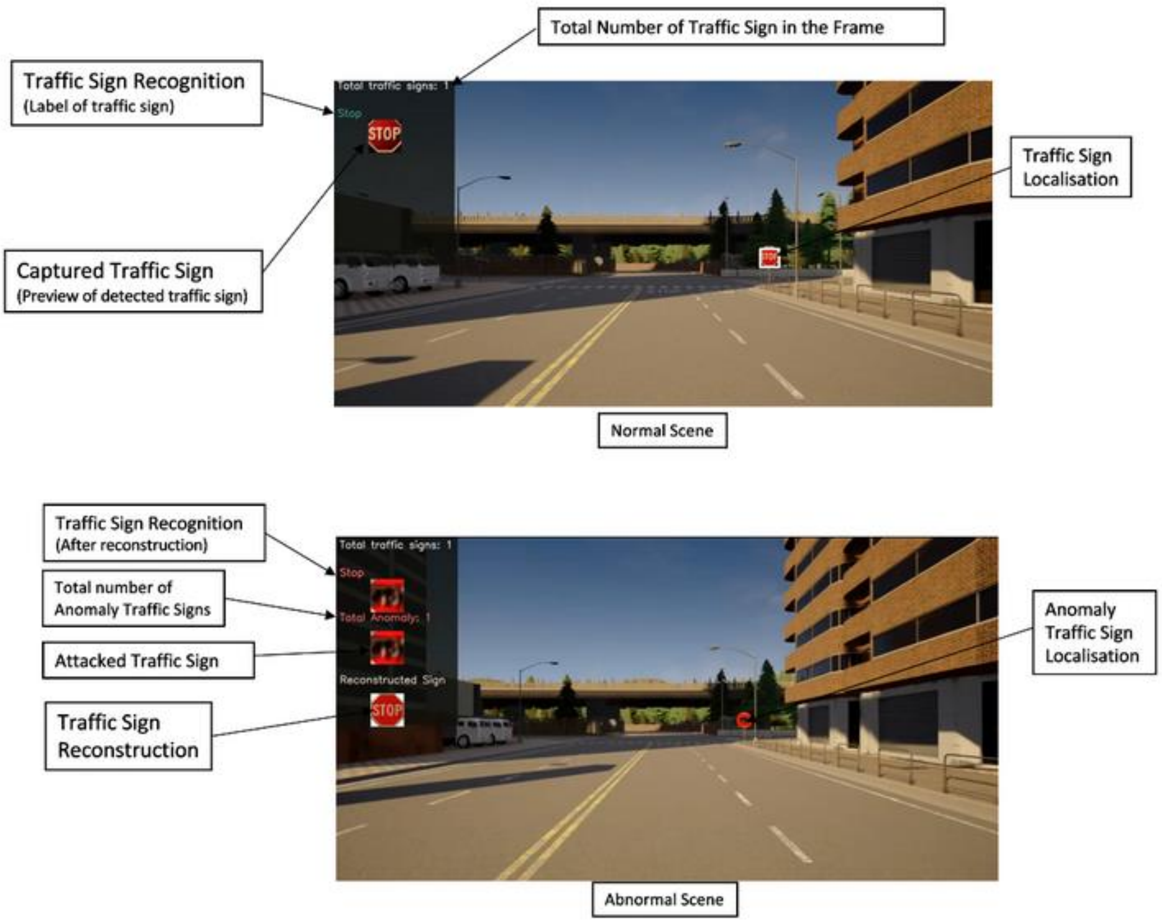

Total Number of Traffic Sign in the Frame
Traffic Sign Recognition (Label of traffic sign)
Total traffic signs: 1
Stop
Traffic Sign Localisation
Captured Traffic Sign (Preview of detected traffic sign)
Normal Scene
Traffic Sign Recognition (After reconstruction)
Total number of Anomaly Traffic Signs
Total Anomaly: 1
Attacked Traffic Sign
Reconstructed Sign
Traffic Sign Reconstruction
Anomaly Traffic Sign Localisation
Abnormal Scene

Figure 4: The top image shows the normal scene. The bottom image shows the scene with attacked traffic signs and their mitigation.

4.1.1.2.2 Implementation into Anti-Hacking Device

All the algorithms that are implemented in the AHDs are packaged in Docker containers. As stated in section 3, the Docker has a number of benefits over direct integration at hardware. The key idea is to compartmentalise the dependencies of algorithms as a portable container and easily deploy in multiple hardware products without directly interfering with the hardware operating system.

A dedicated Docker container has been implemented for this pipeline. The following command runs the client-side pipeline. The Docker container contains all the necessary files and libraries such as python3, tensorflow, anomaly detection, reconstruction and recognition models.

```
├── ConfigManager.py
├── Other
│   ├── mask.pkl
│   └── traffic_sign_lables.csv
├── LogManager.py
├── ModelManager.py
├── Models
```

```
│   ├── anomaly_detection.json
│   ├── traffic_signs_recognition.json
│   └── traffic_reconstuction.json
├── pipeline_helper.py
├── README.md
├── requirements.txt
├── run_anomaly_detection_pipeline.py
├── UtilityManager.py
└── Weights
        ├── anomaly_detection.h5
        ├── traffic_signs_recognition.h5
        └── traffic_reconstuction.h5
3 directories, 16 files
```

Figure 5: Folder structure of the traffic sign anomaly detection and mitigation pipeline

Figure 5 shows the overall folder structure of the client-side pipeline. The developed Deep Learning architecture is stored in *json* format and is located in ***./Models*** folder whereas the trained models weights are stored in the ***./Weights*** folder.

The Docker containers are uploaded in a private Caramel server - *caramelx.mine.bz* and the container can be easily downloaded to the AHDs. Figure 6 shows the command to download the container from Caramel repository and Figure 7 shows the command to run the anomaly detection and mitigation pipeline in Docker environment.

```
$ docker login caramelx.mine.bz
$ docker pull caramelx.mine.bz/jetson:v1
```

Figure 6: Command to download the Docker container in the AHD

And to run the pipeline

```
$ docker run -it --rm --runtime nvidia --network host \
caramelx.mine.bz/jetson:v1 \
python3 /traffic_sign_pipeline/run_anomaly_detection_pipeline.py \
-p 8000 -t 25 -r 'tensorflow'
```

Figure 7: Command to run the anomaly detection and mitigation pipeline in Docker container

The above command runs the pipeline and listens to port (p) 8000 with a timer (t) 25 seconds time out.

4.1.1.2.3 Model performance measurement within the Anti-Hacking Device

In order to measure the performance (frame per second (FPS)), the model was transferred in the AHD - for instance Jetson Nano and AGX. A test dataset of 100 cropped traffic signs was captured and these cropped signs were pre-processed before performing the test. We carried two different experiments, five times in a) running the models in a native environment and b) in the Docker container. The Table 4 and Table 5 shows the average results of the experiment in the Jetson Nano and the Jetson AGX devices.

Table 4: Benchmark performance of model inference per second (FPS) in Nvidia Jetson nano.

Model Type	Native (FPS)	Docker Container (FPS)
Traffic sign anomaly detection	~ 29	~ 5 - 6 (85-79% ↓)
Traffic sign recognition	~ 41	~ 6 (85% ↓)
Traffic sign reconstruction	~ 30	~ 6 (80% ↓)
All model	~10 - 11	~ 1 – 2 (81% ↓)

Table 5: Benchmark performance of model inference per second (FPS) in Nvidia Jetson AGX.

Model Type	Native (FPS)	Docker Container (FPS)
Traffic sign anomaly detection	~ 45 - 48 (55% ↑)	~ 11 (45% ↑)
Traffic sign recognition	~ 85 - 88 (51% ↑)	~ 12 (50% ↑)
Traffic sign reconstruction	~ 46 - 49 (38% ↑)	~ 12 (50% ↑)
All model	~ 18 - 19 (42% ↑)	~ 3 - 4 (50% ↑)

In Table 4, we can observe within the dockerise container the model performance in Jetson nano degraded significantly on average 82% slower than the native environment. Likewise, the Table 5 shows, the Jetson AGX performed much better in both native as well as container environments. On average, AGX performed 46% faster model inference natively than the Jetson nano. Similar performance improvement can be seen in the Docker environment too (on average 48.75 % faster).

4.2 Attack on the Camera / Lidar Sensor

4.2.1 Introduction of Detection and Mitigation of Image Deterioration Attacks using Deep Learning

This solution uses modules developed in deliverables D3.2 [1] and D4.2 [5]. The algorithm first uses deep learning and convolutional neural networks to perform image restoration and remove noise and artifacts. On top of that, additional components are built in order to use the input and generated images for the purpose of detecting attacks and anomalies. Herein we describe how this framework can be implemented on the AHD.

4.2.1.1 *Summary of ML-Algorithm*

Our framework is directed for running inference as an AHD. The configuration structure shown below is deployed in Embedded GPU device - Jetson Nano. The framework was implemented to be deployed in different receive input of different formats. It can run inference by loading isolated or continuous image frames; by loading video files as well as running live through frames received by a connected camera. In addition, the context of receiving image data directly though the CARLA simulator is being implemented.

As explained in D4.2 [5], an attack detection process that can take advantage of the image restoration techniques described in D3.2 [1] (referred to as DriveGuard) was developed. The framework can run in parallel a detection process to the image restoration process thus operating on the restoration outcome. The goal of the overall process is to be able to detect anomalies on the input image and then substitute the input image with the mitigated one, otherwise it should propagate the original input image to the perception module. Thus, enabling the image understanding process to remain robust in the face of image anomalies and different forms of degradations.

We establish 3 main different approaches for detecting attacks (Table 6), describe in D4.2 [5] and summarised below. They rely on the DriveGuard anomaly mitigation autoencoder model that was proven to be effective on retaining the quality of clear images but also on improving the quality of distorted images.

Table 6: Different Mitigation and Detection approaches on the AHD

Type of Algorithms	Description
DriveGuard Anomaly Mitigation and Detection (Baseline Empirical Threshold Approach) - BETA	The evaluation metrics deployed for assessing DriveGuard's mitigating performance were used to empirically determine a weighted equation responsible for assessing the occurrence of an attack. The evaluation metrics returned by the comparison of the input and the generated images are used as weighted input variables to the formula: $T=(a*PSNR+2b*SSIM+c*MSE)+k$
DriveGuard Anomaly Mitigation and Detection - PDS (Parallel Detection Stream Using Latent Features)	A detector that works as a parallel second stream to the network using the bottleneck features. The bottleneck features are dispersed to the DriveGuard Decoder and our detector in order to provide a simultaneous prediction of an attack with the reconstructed image.
DriveGuard Anomaly Mitigation and Detection - TSFC (Two-stage Extension with Fully Connected)	DriveGuard Model Extension as a second stage responsible for generating a detection output with an extended stream, comprised of fully connected layers.

4.2.1.2 *Deployment into Anti-Hacking Device*

The AHD is a lightweight secure and easy to integrate in the CAVs system device. Our framework is deployed and tested in NVIDIA Jetson devices which enable the fast and efficient inference of Deep Learning techniques while being inexpensive.

The algorithms were trained to anticipate attacks of different kinds, so the framework can be deployed in dynamic environments. The AHD can be tested under two different conditions summarized below (Figure 8). The attack injection is carried out on the AHD itself for testing purposes, however, under real operating conditions the image source will already have been attacked.

- **Real World Deployment detection system** - The anomaly detection and distortion mitigation pipeline are deployed in the embedded AHD, with the use of a camera as a real-time input data source.
- **Simulation environment** - is used to simulate the synthetic environment. The virtual camera of an autonomous vehicle agent within the CARLA simulator can be used to provide input to the AHD. The data can be stored and transferred to the AHD for testing. Data collected using the CARLA Simulator are under diverse weather conditions like Clean; Cloudy; rainy weather as well as Midday, Sunset, Sunrise and Night light, so that the image reconstruction is more difficult. In addition, under this scenario the data can already be injected with the attack, so the AHD handles only the detection and mitigation.

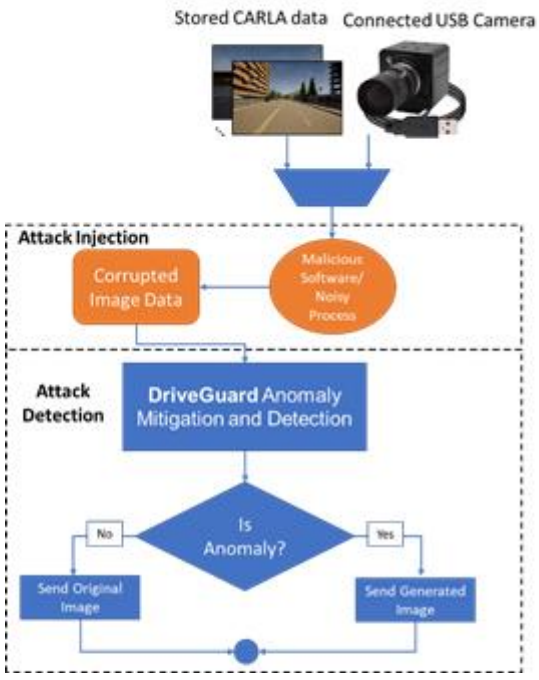


Figure 8: Illustration of the processes running on the AHD.

4.2.1.2.1 Pre-processing Activities

The following tasks are optionally applied so that the input data match what the pipeline expects. These have been extensively discussed in D5.2. The most basic ones are highlighted here for completeness.

- **Resize:** Resize the input image if is of higher resolution so that the image has a specific number of rows and columns. Need to calculate the right number of columns necessary to preserve the aspect ratio. This is mostly necessary to maintain a sufficient frame-rate.
- **Normalize:** Image normalization is a typical process in image processing that changes the range of pixel intensity values. Its normal purpose is to convert an input image into a range of pixel values that are more familiar or normal to the senses, hence the term normalization. In the context of the anomaly detection pipeline specific normalization techniques can help the underlying machine / deep learning techniques to converge faster and hence, they need to be applied at test time as well.
- **Color space conversion:** Color space conversion is the translation of the representation of a color from one basis to another. This typically occurs in the context of converting an image that is represented in one color space to another color space. Typically, common deep learning pipelines operate in the RGB color space format, hence, any other format needs to be converted accordingly.

4.2.1.2.2 Implementation into Anti-Hacking Device

The proposed approach can be packaged in Docker containers for the deployment in the AHDs. The Docker approach enables us to deploy the framework in different devices in an isolated environment without directly impeding with the corresponding operating system of the device.

Prerequisites:

- The Device has to be connected to the internet during the installation process.
- The Docker package has to be installed, following: https://docs.docker.com/get-docker/

The user has to download our Docker container to the AHD and then install the Docker by entering into the directory that contains 'dockerfile' and running the command:

```
$ sudo docker build -t apapac03/driveguard:jetson.2 .
```

The dockerfile then is responsible for the setup of the environment and the installation of the required packages.

Following the installation you can activate and run the framework using the command:

```
$ sudo docker run –rm –gpus all apapac03/driveguard:jetson.2 .
```

The configuration is based to the one shown in D4.2 [5] with minor changes in the structure of the directory due to the implementation through a Docker container. In order to minimise the User interference with the device, we added a lower-level layer main manager with the script 'Main.py', which is responsible for directing:

- Framework: DriveGuard/ DriveGuardandDetect, through argument -f
- Input Source: Camera/ Simulator/ Pre-recorded Data, through argument -s

The arguments passed in the 'Main_...' scripts in the version shown in D4.2 [5] are now passed to the aforementioned command as shown in Figure 9:

Autoencoder Model: STAE, SFAE, DL_FC, DL_Bottleneck, through argument -a

```
├── dockerfile
├── DataFolder/
├── OutputFolder/
├── Other/
│   ├── RunCamera.py
│   └── Readme_Deeplab_foreval.txt
├── src/
│   ├── Main.py
│   ├── DG_Det_RunBaseline_fromData.py
│   ├── DG_Det_RunBaseline_Carla.py
│   ├── DG_Det_RunBaseline_Camera.py
│   ├── DG_Det_RunDL_fromData.py
│   ├── DG_Det_RunDL_Carla.py
│   ├── DG_Det_RunDL_Camera.py
│   ├── DG_Run_fromData.py
│   ├── DG_Run_Carla.py
│   ├── DG_Run_Camera.py
│   ├── Main_Run_DeepLab.py*

│   ├── Tools

│   │       ├── utils.py

│   │       └── dataLoaders.py

│   ├── Models

│   │       ├── DriveGuard.py

│   │       └── DriveGuardandDetect.py

│   ├── Weights

│   │       ├── STAE_H68.pth

│   │       ├── DL_FC_E61.pth

│   │       ├── DL_BottleNeck_E34.pth

│   │       └── SFAE_H71.pth

├── README.md

└── requirements.txt

4 directories, 3 subdirectories, 24 files
```

Figure 9: Code structure

4.2.1.2.3 Model performance measurement within the Anti-Hacking Device

The performance of the different models was built and tested on 3 different devices on a dataset of 100 images. The 3 devices used are the:

- Desktop PC with NVIDIA Quadro P400 GPU and Quad-core Intel CPU processor of 2.5 GHz.
- An NVIDIA Jetson Nano 4GB
- AN NVIDIA Xavier NX

The speeds were calculated in FPS inferenced. We measured the inference speeds of the whole pipeline of the different Models as explained in the earlier section as well as the speeds according to the time needed for the GPU processes alone. Since the different devices have different specifications, we tried to evaluate these speeds to identify their limitations and the main cause of latency in each case.

Table 7: Benchmark performance of model inference FPS in Device A: NVIDIA Quadro P400 + Quad-core Intel processor, 2.5 GHz

Device A: NVIDIA Quadro P400	**Implementation**	
Detector Model Type	**Whole Pipeline (FPS)**	**GPU Process (FPS)**
DriveGuard (Standalone)	~76.5-77.5	~470-550
DriveGuard + Two Stage Fully Connected (TSFC)	~63.5-64	~83.5-84
DriveGuard + Baseline Empirical Threshold Approach (BETA)	~66-67	~470-550
DriveGuard + Parallel Detection Stream (PDS)	~65-66	~240-260

First, in Table 7 through the baseline implementation using an NVIDIA Quadro P400, we can observe from the performance statistics on this device that the main cause of latency is the CPU processing time. Since the Autoencoder is designed to be lightweight and fast on a GPU which can calculate convolutions at extreme speeds, the DriveGuard as a standalone process reaches speeds of about 500 FPS on a GPU but it is limited to about 77 FPS when detached to the CPU. The difference is not that significant on the TSFC detector which utilizes fully connected neural network layers as an extension to the DriveGuard mitigator. The GPU is significantly slower on calculations on this type of layers than on convolutions.

Therefore, the difference in speed between the GPU processing time and the whole pipeline is much smaller. For the BETA approach the GPU process speed is the same with the standalone DriveGuard since the model is not extended and the same calculations occur by the GPU. However, the comparisons to compare the two images produce an extra bottleneck latency for the CPU which delays the whole pipeline and lowers the FPS by about 10. In the case of the PDS model, even if the GPU processing time is half of the BETA DriveGuard it performs under the same speeds since the extra CPU processes of comparing the input and generated images are diminished.

Table 8: Benchmark performance of model inference (FPS) in Device B: Nvidia Jetson Nano

Device B: NVIDIA Jetson Nano	**Implementation**		**Docker**
Detector Model Type	**Whole Pipeline (FPS)**	**GPU Process (FPS)**	
DriveGuard (Standalone)	~4.5	~69-70	~3.5
DriveGuard + Two Stage Fully Connected (TSFC)[1]	0.8	~2-5	N/A
DriveGuard + Baseline Empirical Threshold Approach (BETA)	~3.5-4	~69-70	~2.5
DriveGuard + Parallel Detection Stream (PDS)	~3.5-4	~55-60	N/A

Table 9: Benchmark performance of model inference (FPS) in Device C: Nvidia Jetson Xavier NX

Device C: NVIDIA Jetson Xavier NX	**Implementation**		**Docker**
Detector Model Type	**Whole Pipeline (FPS)**	**GPU Process (FPS)**	
DriveGuard (Standalone)	~13-15	~240-250	N/A
DriveGuard + Two Stage Fully Connected (TSFC)	~3	~21	N/A
DriveGuard + Baseline Empirical Threshold Approach (BETA)	~12-13	~240-250	N/A
DriveGuard + Parallel Detection Stream (PDS)	N/A	N/A	N/A

[1] The FPS shown for the Docker do not include visualisation latency

In Table 8 and Table 9 we can observe that the Jetson Devices' performances are affected similarly with the Desktop PC performance by the CPU limitations. When comparing the different models, the Jetson Nano is significantly larger in size of Weights which causes the Jetson Nano device to be inconsistent in performance while in some cases it crashes while inferencing. For the rest of the models the performance of the whole pipeline is about 15 times slower than the GPU processing on both Jetson Devices. Analogously the decrease in speed is much greater in the Jetson Devices since they are much more limited in CPU performance while there are optimised embedded compute unified device architecture (CUDA) machines.

Since the implementation as an AHD is aimed to be packaged in Docker containers their performance when ran on Docker applications are much more important.
When running the models on a Jetson Nano in a Docker Container there is a decrease of about 20 percent in speed. Therefore, there is a significant trade-off between the advantages of a Docker Container in reproducibility and portability with the performance. We will be working to resolve technical issues that may affect performance.

4.2.2 Introduction of Mitigation of Adversarial Attacks using Lidar Sensor

A late fusion strategy was chosen for the proposed mitigation algorithm, because given that the camera sensor is attacked, the data coming from it would not be reliable for post-processing. In late fusion, two classifiers are used to generate a final and robust decision. The proposed high-level architecture is shown in Figure 10.

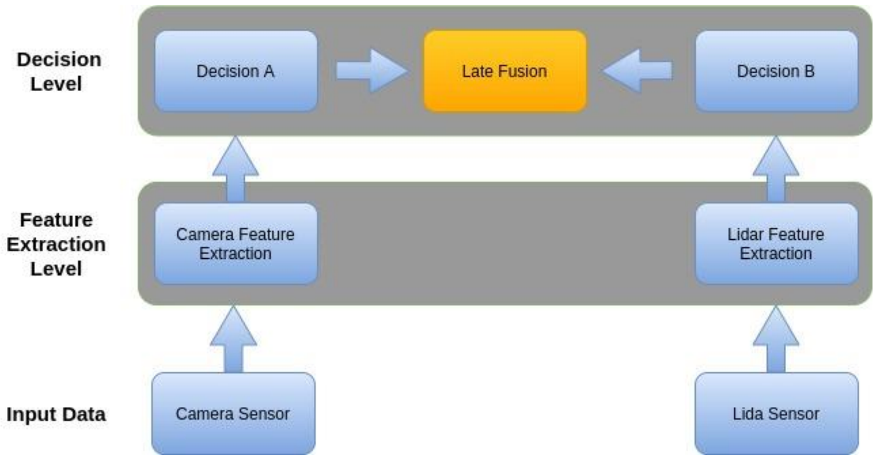


Figure 10: Sensor fusion architecture describes where the late fusion occurs.

More specifically, the output of a robust 2D image segmentation model is fused with the output of a 3D object detection model based only on light detection and ranging (LIDAR) data. So, by combining these two outputs, we can decide whether the camera sensor has been attacked or not. Hence, a safe situation derives when majority of the 3D detected objects has also been detected in the 2D model. Otherwise, an alert is raised to the user and the final decision of the perception engine is coming only from the LIDAR sensor.

4.2.2.1 *Summary of ML-Algorithm*

The proposed framework consists of multiple algorithms. Data from camera and LIDAR sensor are given as input to the whole procedure. The final output contains a flag, indicating whether an external attack exists on the camera sensor or not, and the detected objects in the scene. More specifically, the proposed framework utilized modules from previous work packages. Hence, it consists of the following modules: (2D) Image Denoising from D3.2 [1], (2D) Image Segmentation from D4.2 [5], (3D) Object

Detection and Fusion Results coming from the current work package. The previous modules are briefly summarised in Table 10. For more details, please refer to the relevant deliverables.

Table 10: Brief description of algorithms implemented for the mitigation of attacks on the camera sensor using the LIDAR sensor

Type of Algorithm	Description
(2D) Image Denoising	The proposed denoising method [26] achieves arbitrary-level image restoration and continual model modulation, with little computation cost, in a unified CNN framework. The framework is built upon an Adaptive Feature Modification (AdaFM) layer that modifies the middle-layer features with depth wise convolution filters. By simply tweaking an interpolation coefficient, the intermediate model AdaFM-Net could generate smooth and continuous restoration effects without artifacts. Overall, the 2D denoised image is exported in this stage.
(2D) Image Segmentation	The segmentation model of DeepLabv3 [27] is characterized by an encoder-decoder structure. The encoder module encodes multi-scale contextual information by applying atrous convolution at multiple scales, while the simple yet effective decoder module refines the segmentation results along with the object boundaries. Overall, 2D image segmentation is exported in this stage, in which each class is denoted with a different label.
(3D) Object Detection	The model of PointRCNN [28] was proposed which is a bottom-up point cloud-based 3D bounding box proposal generation algorithm. It generates a small number of high-quality 3D proposals via segmenting the point cloud into foreground objects and background. Overall, 3D bounding boxes for the detected objects are exported in this stage.
(3D) Projection to (2D) Image Plane	This module implements the projection of 3D data to the 2D image plane for post-processing.
Fusion Results	Results coming from both sensors are fused in this stage. If the majority of the 3D projected data match the 2D initial data, then we have a safe situation. Otherwise, we suppose that the camera sensor has been attacked. In the first case, a safe flag and the outputs of the camera sensor are exported. In the second case, an alert is shown to the user and the 3D projected data are taken into consideration.

4.2.2.2 *Deployment into Anti-Hacking Device*

The AHD in the CARAMEL project represents a device which is secure and easy to integrate into the CAVs system. The pipeline was built to anticipate the state-of-the-art adversarial attacks. In real scenarios, it is supposed that the image coming from the camera sensor will be already attacked. For testing purposes, the attack is performed into the AHD of Jetson. The proposed framework was tested in real data using the KITTI dataset [29]. An illustration of our solution is shown in Figure 11, in which the modules with red colour have been developed in the current work package.

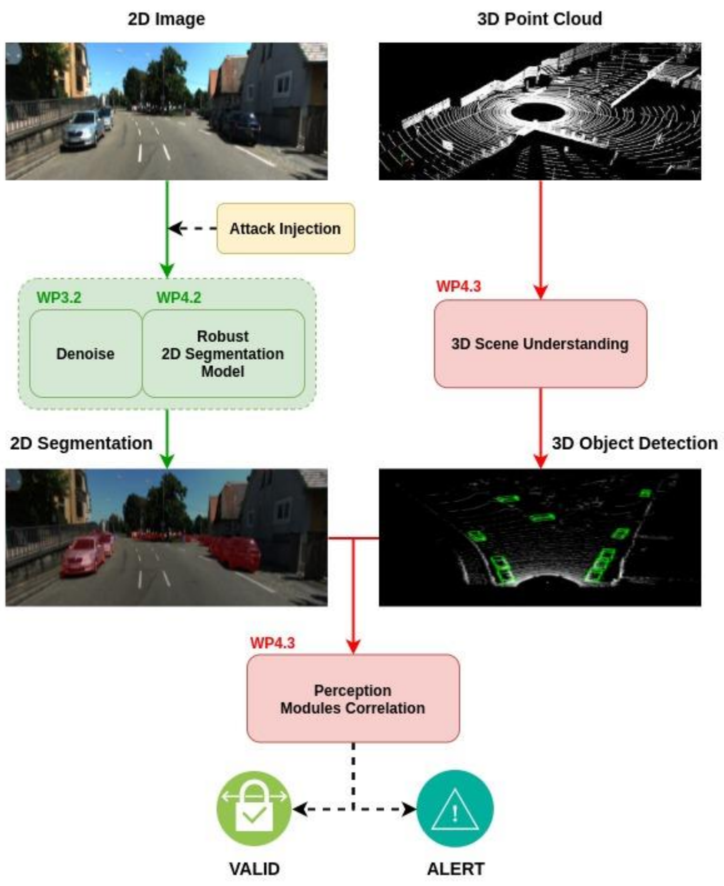


Figure 11:Overall pipeline of the mitigation technique

4.2.2.2.1 Pre-processing Activities

Some of the pre-processing activities that have been applied to input data so as to reach the pipeline expectations are briefly described in this section. The following activities have been applied only to image data.

- **Crop:** Crop images to remove redundant regions of the image, such as the sky.
- **Resize:** Resize the input image if is of higher resolution so as to make the algorithms run faster and at the same time maintain the accuracy at high level.
- **Normalize:** Modify the range of pixel intensity values.

4.2.2.2.2 Implementation into Anti-Hacking Device

The proposed framework has been packaged in Docker containers for the deployment in the AHD. The user needs to follow the next steps, see Figure 12 in order to test our framework.

I. 1. NVIDIA driver (>450)
II. 2. docker CE (>19.03)
III. 3. Install Nvidia-docker2
   a. Setup the stable repository and the GPG key:

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
 && curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add - \
&& curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
```

b. Install the nvidia-docker2 package (and dependencies) after updating the package listing:
```
$ sudo apt-get update
$ sudo apt-get install -y nvidia-docker2
```
c. Restart the Docker daemon to complete the installation after setting the default runtime:
```
$ sudo systemctl restart docker
```
IV. 4. docker pull chranagno/model:latest
V. 5. git clone <name of jetson repo>
VI. 6. docker run --rm -it --gpus all -v $(pwd)/jetson:/workspace/jetson caramel:model
VII. 7. cd jetson
VIII. 8. python run.py

Figure 12: Instruction for installation in Jetson Tx2

4.2.2.2.3 Model performance measurement within the Anti-Hacking Device

The proposed architecture has been evaluated into AHD of Jetson Tx2. Two pillars are being processed in parallel (Figure 11). The first one processes images data, while the second one processes LIDAR data. As it regards to the benchmark performance, the images have been scaled and cropped (the sky has been removed as redundant region). Also, the LIDAR data have been captured from the KITTI dataset using an HDL-64E Velodyne LIDAR sensor. Overall, the proposed framework does not run in real-time as you can observe from Table 11. However, some optimization steps will be investigated to make the algorithms faster into the AHD. For example, as it regards to the LIDAR data, one solution would be to subsample the Velodyne points or capture the environment using an HDL-16E Velodyne sensor.

Table 11: Benchmark performance of model inference per second (FPS) in Nvidia Jetson Tx2

Model Type	Performance (FPS)
Image Denoiser	~6-7
2D Image Segmentation	~6-7
3D Object Detection	~1-2

4.2.3 Introduction of Denoising Algorithm on Camera Sensor

Camera sensors on board of Connected Autonomous Vehicles have already been proven vulnerable to a wide range of attacks, even though the field of research is not yet mature enough to have exhausted all possibilities. Thus, a non-exhaustive shortlist of potential attacks that have been examined so far, is:

1. **Spoofing**, i.e. placing false objects like traffic lights or traffic signs in the field of view of the cameras, so that the autonomous vehicle takes the wrong decision [30][31].
2. **Blinding**, i.e. shedding high-brightness infrared (or even multiple randomly alternating wavelengths) lasers on the camera sensor, preventing them from capturing effective images for the perception stage [32].
3. **Adversarial attacks**, which mostly harm Deep Learning based systems, as they are deliberate perturbations of the input to lead to erroneous output. Several examples of these methods have been covered in [33].

In recent research papers [30][33]-[41], taxonomies of potential attacks based on their nature, attacker type and severity have been studied further, however the variety of attacks is still limited to very obvious ones, bar the adversarial attacks which are much more elaborate and imaginative.

Image deterioration attacks aim to alter the input image in order to lead the vehicle perception modules to fail. In contrast to adversarial examples these attacks are not guided by a target label, but the objective is to cause a general drop in the image quality so that the perception module's output becomes erroneous. These attacks are rather simple and do not require an attacker to have knowledge or access to any perception model and thus can be considered as more common. Causes of image quality loss can be because of problems with the camera lens (e.g., scratch lens), erroneous/invalid data (blank image regions), presence of dead, stuck, or hot pixels. In addition, there is also a scenario where an attacker manages to access critical vehicle systems and manipulate the camera image via modification through internal malicious software. Given such deliberate or unintentional events the objective is to be able to restore the image quality to a good enough level so that the perception module (e.g., semantic segmentation) can still perform with adequate accuracy. Methods of image restoration are a topic of research in the last decades. Filtering approaches such as Gaussian Filters used in studies [42][43][44] where they apply transformations of pixels based on the information provided by a window of neighbors. The lack of stronger adaptivity to the underlying structure of the image and objects of interest is a major drawback of these classical approaches. Other approaches include the Bilateral Filter [43][45], based on the Gaussian Filter, which integrates pixel-wise spatial and photometric disparity. The advantage is that the computational complexity remains low, but it employs adaptivity to the window used. On the downside, in low disturbed images it doesn't perform accordingly. Other works measure frame quality assessments as features, and then the computed features are analyzed for fault detection [46] or try to remove environmental noise such as [47]. These traditional methods are investigated in [43].

4.2.3.1 *Summary of Denoising Algorithm on Camera Sensor*

The conventional filters such as averaging filter, median filter, and 2D Gaussian filter are efficient in smoothing the noise, but also have the disadvantage of blurring image edges [48][49]. For this reason, some nonlinear image filters, which produce more satisfactory noise reduction results and preserve better the integrity of edges and detail information, were introduced [50]. The nonlinear noise removal techniques based on Partial Differential Equations (PDEs) have been extensively studied in the last two decades [51][52]. The PDE models are the best candidates for a very efficient image denoising. We propose several robust PDE-based models for noise removal and image restoration in our previous works [51][52][53]. In CARAMEL we develop novel PDE-based image denoising approach based on nonlinear diffusion. The idea of using the diffusion in image processing arose from the use of the Gaussian filter in multiscale image analysis. Nonlinear diffusion methods reduce noise and enhance contours in images [54][55][56]. Numerous nonlinear PDE denoising approaches based on diffusion have been introduced since the early work of Perona and Malik in 1987 [55]. The anisotropic diffusion developed by them, also called Perona-Malik diffusion, was intended to smooth the image while preserving its edges. Many denoising schemes based on this influential work have been proposed in the last 25 years [55][56].

Recently, a new concept and a new task have been introduced in image analysis: decompose a given image $f: \Omega \longrightarrow R$ into two components $u + v$, such that u is a cartoon representation of f, a simplified piecewise smooth approximation, while v is the oscillatory component, consisting of texture and noise. The canonical, standard, previous models in image analysis start from the assumption that an image f is the sum of two components, $f = u + v$, such that only u contains the important main features of f. These models represent u as a function of bounded variation (in $BV(\Omega)$ or in a subset, $SBV(\Omega)$), allowing for discontinuities along curves, therefore sharp edges and contours in the image u. The residual $v := f - u$ is modeled in the standard models by a function in $L^2(\Omega)$. The main canonical examples are the total variation (TV) minimization of Rudin-Osher-Fatemi [57], Rudin-Osher [58], and the Mumford-Shah segmentation model [59]. The first model decomposes f into $u + v$, such that $u \in BV(\Omega)$ and $v := f - u \in L^2(\Omega)$, while the second model gives a decomposition $f = u + v \, with \, u \in SBV(\Omega)$ (special functions of bounded variation), and $v := f - u \in L^2(\Omega)$, as in the Rustin-Osher-Fatemi Model (ROF model). These and other related models retain only the piecewise smooth component u from f and remove away the texture + noise component $v := f - u$.

4.2.3.1.1 Perona Malik Anisotropic Diffusion

To avoid the edge blurring of Gaussian scale-space, in CARAMEL we employed as denoising engine a nonlinear diffusion PDE was proposed by Perona & Malik [52][55]

$$\frac{\partial u}{\partial t} = div(k\nabla u) = k\nabla^2 u + \nabla k \nabla u \qquad (1)$$

where k is a varying diffusion coefficient (dependent possibly on any of $x, y, t, u, \nabla u$) whose purpose is to favor intra-region over inter-region smoothing, i.e., to inhibit or reduce smoothing at strong edges. This can be accomplished if $k = g(\|\nabla u\|)$, and g is a smooth nonincreasing function with $g(0) = 1$, $g(r) \geq 0$, and $\lim_{r\to\infty} g(r) = 0$. With such a choice for k, the diffusion will be small (large) when the edges are strong (weak), where the edge strength is measured by $\|\nabla u\|$. Typical choices of g include $g(r) = exp(-ar)$ and $g(r) = \frac{a}{(a + r^2)}, a > 0$. Two problems with the diffusion scheme (1) are the amplification of noise by the gradient and sensitivity to initial conditions for certain choices of g. These problems were solved in Catte, Coll, Lions & Morel [56] by replacing the estimated edge strength $\|\nabla u\|$ with $\|\nabla G_s * u\|$, i.e., by smoothing with a Gaussian at some fixed scale s before taking the gradient. This solution did not have a clear geometric interpretation and the stability of the model was not guaranteed as $s \to 0$. A further improved solution was given by Alvarez, Lions & Morel [53][52] and its simplest form is

$$\frac{\partial u}{\partial t} = g(\|\nabla G_s * u\|)div\left(\frac{\nabla u}{\|\nabla u\|}\right)\|\nabla u\| \qquad (2)$$

The term $div\left({}^{\nabla u}/_{\|\nabla u\|}\right)$ performs a pure anisotropic diffusion by diffusing u only in the direction orthogonal to ∇u and not all in the direction of ∇u. The contrast term $g(\|\nabla G_s * u\|)$ controls the speed of diffusion and thus enhances the edges. From the level set method of curve evolution, evolving u according to (2) propagates all its level sets with a normal speed equal to their curvature $(-curv)$ times $g(\|\nabla G_s * u\|)$.

4.2.3.1.2 Total Variation

Another approach used for removing cyberattack on camera sensor is Total Variation scheme. TV is an effective tool to solve image denoising and many other image processing problems. For the denoising problem, it is necessary to create automatic image processing methods based on parameters estimation of the corresponding models. For TV image denoising problem, almost methods focus on TV-L2 norm. The denoising model with TV-L1 norm is impressive to treat the salt-and-pepper noise. The general TV problem is described as (3):

$$T(u) = F(b \mid Au) + \lambda\|\nabla u\|_1 \qquad (3)$$

and by stressing that a non-negativity constraint (**u** ≥ 0) may be imposed to (7); also, throughout this paper we will consider that the discrete version of $\|\nabla u\|_1$ is given by $\left\|\sqrt{\sum_{n\in C}(D_x u_n)^2 + (D_y u_n)^2}\right\|$, where $n \in C = \{1\}$ (grayscale images case) or $n \in C = \{1, 2, 3\}$ (or "red," "green", and "blue," for the color images case; also note that C could represent an arbitrary number of channels), the horizontal and vertical discrete derivative operators are denoted by D_x and D_y, respectively, and that the scalar operations are applied elementwise, so that, for example, $u = \sqrt{v} \Rightarrow u[k] = \sqrt{V[k]}$. We also note that (3) is the generalization of TV regularization to color images with coupled channels, typically used within the color or vector-valued TV framework, which coincides with the discrete version of ‖∇**u**‖ 1 for grayscale images ($n \in C$ = {1} in (3)), that is $\left\|\sqrt{(D_x u)^2 + (D_y u)^2}\right\|_1$. Due to the non-differentiability of $\|\nabla u\|_1$, it is sometimes replaced by the smooth approximation $\frac{1}{2}\sum_k \tau_\varepsilon(v_k)$ where $v^2 = \sum_{n\in C}(D_x u)^2 + (D_y u)^2$ and some sensible choice of function $\tau_\varepsilon(.)$such $\tau_\varepsilon(v_k) = 2\sqrt{v_k^2 + \varepsilon^2}$ or such the Huber function $\tau_\varepsilon(v_k) = \{\varepsilon^{-1}v_k^2 \; if \; |v_k| \leq \varepsilon^2 \; 2|v_k| - \varepsilon \; if \; |v_k| > \varepsilon^2$

4.2.3.2 *Cyber-attack mitigation on the LiDAR Sensor*

To robustly perceive the environment, CARAMEL will perform sensor fusion of the LIDAR and camera using the particle swarm optimization algorithm, without the aid of any external calibration objects. The proposed algorithm automatically calibrates the sensors and registers the LIDAR range image with the stereo depth image. The registered LIDAR range image functions as the disparity map for the stereo disparity estimation and results in an effective sensor fusion mechanism. Additionally, we perform the image denoising using the modified non-local means filter on the input image during the stereo disparity estimation to improve the robustness, especially at nighttime. To evaluate our proposed algorithm, the calibration and registration algorithm is compared with baseline algorithms on multiple datasets acquired with varying illuminations.

4.2.3.2.1 Proposal of the Solution

At first, we will briefly review the sensor geometry of stereo and LIDAR. Velodyne HDL-64E LIDAR and Camera stereo camera attached to the vehicle represents the sensors used in our algorithm. Stereo vision is the process of estimating the depth of a scene from two or more camera views. Each camera generates an image of the scene using perspective projection in the form of the pinhole camera model [36]. On the other hand, the 3D LIDAR is a range sensor that measures the distance of the surrounding environment by analyzing reflected light from objects, illuminated from a laser. Typically, the sensor measurements are represented in the spherical coordinate system, given as (θ, ϕ, d). The $\theta$and ϕ represent the azimuth and elevation angle, respectively, while the d corresponds to the distance observed at those angles. Note that azimuth and elevation represent the rotation about the vertical and horizontal axes respectively. A comparison between the sensor geometry of the LIDAR and the optical camera is provided in. Unlike the perspective geometry of the optical camera, the LIDAR range image has a constant sampling angle per pixel. More specifically, the pixels of the optical camera subtends increasingly large angles with the increase in distance [36].To form the LIDAR image, the LIDAR spherical distance measurements (θ, ϕ, d) are radially mapped to the 2D image plane by,

$$\begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = \begin{bmatrix} r_u & 0 & t_u \\ 0 & r_v & t_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \phi \\ t \end{bmatrix} = \begin{bmatrix} r_u\theta + t_u \\ r_v\phi + t_v \\ 1 \end{bmatrix} \qquad (4)$$

where r_u, r_v, t_u and t_v represents the intrinsic 2D scaling and translation parameters. Note that only (θ, ϕ) are used for the radial mapping to (u, v). Following the mapping, for a given range image pixel position (u, v) the distance d of the corresponding spherical coordinate (θ, ϕ) is identified and stored. In this paper, we propose to integrate the automatic calibration of the LIDAR and stereo within the LIDAR image formation. Thus, the LIDAR image generated from the spherical data would be inherently calibrated and registered with the stereo data. To integrate the automatic calibration and registration, firstly, we transform the LIDAR spherical coordinates to the LIDAR Cartesian coordinates (x_l, y_l, z_l) by,

$$\begin{bmatrix} x_l \\ y_l \\ z_l \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & \sin\phi & 0 \\ 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\phi\sin\theta\ d \\ \sin\phi\ d \\ \cos\phi\cos\theta\, d \\ 1 \end{bmatrix} \qquad (5)$$

Given the Cartesian LIDAR coordinates, we then estimate the extrinsic calibration parameters between the LIDAR and stereo Cartesian coordinate systems. Following the optimization of the extrinsic calibration parameters, we obtain the calibrated LIDAR cartesian coordinates $(\overline{x_l}, \overline{y_l}, \overline{z_l})$. Following the calibration estimation, the transformed LIDAR Cartesian coordinates are inverse mapped back to the LIDAR spherical coordinate system [36] to generate the LIDAR range image. This is given as,

$$d = \sqrt{\overline{x_l}^2 + \overline{y_l}^2 + \overline{z_l}^2} \qquad (6)$$

$$\theta = \tan^{-1}\left(\frac{\overline{x_l}}{\overline{z_l}}\right) \qquad (7)$$

$$\phi = \sin^{-1}\left(\frac{y_l}{\sqrt{\overline{x_l}^2 + \overline{y_l}^2 + \overline{z_l}^2}}\right) \qquad (8)$$

Finally, by substituting the inverse mapped spherical coordinates in Equation 4, the LIDAR range image is formed by,

$$u_l = r_u \tan^{-1}\left(\frac{\overline{x_l}}{\overline{z_l}}\right) + t_u \quad (9)$$

$$v_l = r_v \sin^{-1}\left(\frac{y_l}{\sqrt{\overline{x_l}^2+\overline{y_l}^2+\overline{z_l}^2}}\right) + t_v \quad (10)$$

A more detailed analysis of the methodology and the distinct steps of the solution is provided in Deliverable 3.5.

4.2.3.3 *Deployment into Anti-Hacking Device*

Panasonic's solution will not run on the AHD but on the computational unit installed in the car due to the real time restrictions that it needs to satisfy for being demoed.

4.3 Location Spoofing Attack

4.3.1 Introduction of GPS spoofing attack detection by using ML technique

Global positioning system (GPS) is used in a multitude of civilian as well as military related applications. Civilian GPS signals are unencrypted and therefore present a vulnerability which can be exploited by an attacker. If such an attack is undetected, vehicle can be steered from its desired trajectory and that can lead to various safety hazards.

In order to detect anomalies in GPS data, we propose the use of ML algorithms, Long Short-Term Memory (LSTM) and Temporal Convolutional Networks (TCN), to predict position of the vehicle and find outliers within a multivariate time series.

Modelling of a GPS attack is a very challenging task. Given the fact that the attacker gains complete control over the GPS receiver output, no real model of the GPS spoofing attack exists [60]. In our case, to model the attack, an offset was introduced to original latitude, longitude and altitude values in the dataset. Fault was introduced for multiple adjacent time steps. The goal of our model is to detect this faulty subsequence's. A detailed description of the implementation is given in the D3.4 [2] - Multi-modal Data Fusion Module for Responding Reliably to the Threats of V1 and V2. The deployment of the GPS spoofing attack detection from AVL will be implemented in Model.CONNECT which is an AVL inhouse tool to create the simulation environment.

4.3.1.1 *Summary of ML-Algorithm*

In GPS spoofing attack detection, the ML & AI techniques can be implemented mainly in two stages. At first in the detection of the spoofing attack and next in the mitigation action taken after the detection. In CARAMEL it is mostly implemented to detect the attack and then other vehicle related parameters are considered to get the location information so that GPS can be overlooked for the localization of the vehicle.

Table 12: Brief description of algorithm implementation for the mitigation of attacks on the GPS sensor

Type of Algorithms	Description
Comparing the GPS and vehicle parameters to obtain the location information	Obtaining the data from the GPS sensor along with the vehicle parameters to compare the location information of the vehicle is the first step in GPS spoofing attack detection. The data obtained from two sources will go through the pre-processing step.
Spoofing attack detection	After the pre-processing step, the data is sent to the ML algorithm which is a Neural Network. At training session, the network would have been trained with a similar dataset so that it can identify if there is any deviation in the values recorded.
Warning / Alert message	After the Convolutional Neural Network detects the anomaly a warning would be displayed to the driver so that he can ignore the GPS data from then on and rely on other sensor parameters to know the location information.
Spoofing attack mitigation	Spoofing mitigation performs countermeasures against the attack by helping to retrieve the location information of the victim GPS receiver or by using any alternate solution to get the location data.

4.3.1.2 *Deployment into Anti-Hacking Device*

The AHD in CARAMEL project contains all the ML & AI implementations which are developed to detect and mitigate the risks and threats on the sensors. The spoofing attack detection and mitigation technique which is developed for overcoming the vulnerabilities of the GPS sensor will be tested in the testbed setup by including a test GPS receiver. The GPS receiver which receives the current location information will be tampered in a certain way as to create the spoofing attack scenario. After the received signals are tampered, it will be sent along with the vehicle data to the ML & AI model. The vehicle data at this stage is obtained by using a simulation environment named VTD and Model.CONNECT. After the testing of the algorithm in the later stage the whole set up would be packaged as an ARM-based Docker container and tested in the AHD of CARAMEL.

4.3.1.2.1 Pre-processing Activities

In the process of obtaining a clean dataset for the ML&AI algorithm to predict the possibility of spoofing attack, there are several pre-processing activities which needs to be performed. There are several techniques and algorithms involved in the whole pre-processing stage. A summary of all these steps are mentioned below.

Outlier Detection: The first step is to detect outliers in measurements. This is done by defining boundary values for each feature. The measurements that are outside of the defined boundaries are considered as outliers by ML model.

Feature Engineering: In this step, along with the obtained measurements, new combinations of existing features are created based on the features' correlations and interactions.

Feature Selection and reduction: After applying the feature engineering and filtering technique to filter the outliers, the next prominent step in creating a sustainable dataset is performing the feature selection and reducing the unwanted feature from obtained data. This step helps in forming quality data, which is more generalized and simplified in structure. To perform feature selection and reduction, a scalable end-to-end tree booting system called XGBoost is applied. XGBoost is a gradient boosting algorithm used in many ML applications to increase the ML model's speed and performance [61].

In Testbed implementation stage using simulation environment the received data from GPS will be tampered at a certain period using a fault injection algorithm. The ML & AI model will receive the data to perform classification or detection step and then follow the actions accordingly. The below Figure 13 depicts the pipeline which has been followed to detect and mitigate the attack on GPS sensor using Neural Network.

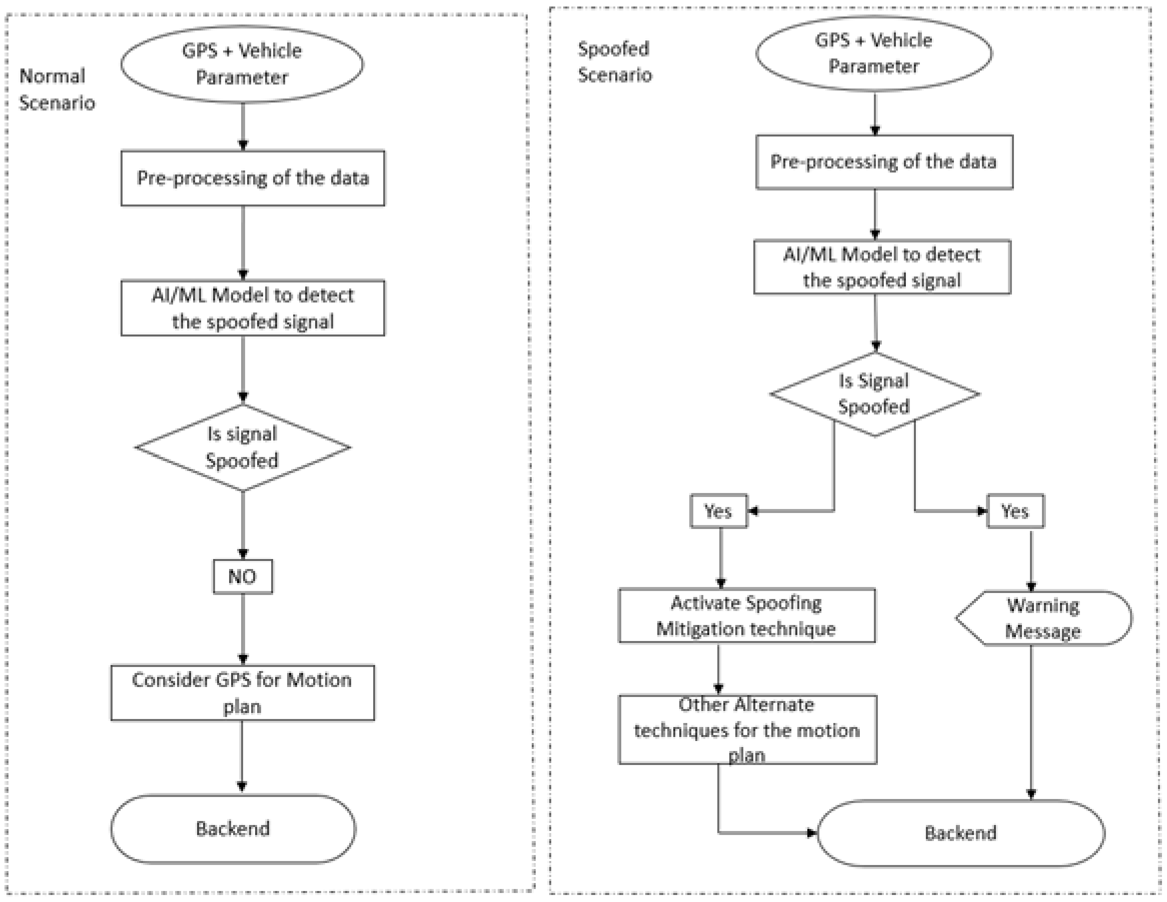

Normal Scenario
GPS + Vehicle Parameter
Pre-processing of the data
AI/ML Model to detect the spoofed signal
Is signal Spoofed
NO
Consider GPS for Motion plan
Backend
Spoofed Scenario
GPS + Vehicle Parameter
Pre-processing of the data
AI/ML Model to detect the spoofed signal
Is Signal Spoofed
Yes
Yes
Activate Spoofing Mitigation technique
Warning Message
Other Alternate techniques for the motion plan
Backend

Figure 13: Depicts the flow of spoofing attack detection in GPS in both attacked and normal scenario.

4.3.2 Introduction of GPS Spoofing Attack in Collaborative Vehicles

GPS Spoofing attack is a well-known limitation of GPS sensor that threatens the safety and security of vehicles. An attacker may cause additional challenges to the transportation sector, by simultaneously launching the attack to a number of moving vehicles. Furthermore, should the compromised vehicles be vehicle to vehicle (V2V) connected to enhance their cooperative awareness ability by sharing common but rather heterogeneous data, then GPS spoofing is able to provoke significant damages, even leading to traffic accidents. To this end, detecting and mitigating the effect of the cyberattack in a number of networked vehicles, is critical to their safety and intact operation. Each vehicle on its own is able to defend against spoofing, but not very accurately re estimate its position. Moreover, it is not easy for individual vehicles to detect the compromised ones that belong to the same group of collaborating vehicles but don't have direct V2V connection. Thus, a central node / fusion centre, i.e. road side unit or the 5G cloud, after collecting individual measurements, will be responsible for detecting all possible attacked vehicles, while at the same time estimating their locations by removing the impact of spoofing and informing them accordingly. For that purpose, we developed a collaborating defence mechanism, originally described in deliverables D4.3 [6] and D4.4 [62], that it will not be deployed in the AHD. In this simulated approach, cooperating vehicles of the same vehicular ad-hoc network, are required to transmit their measurements to the 5G cloud, which will be responsible for mitigating the effects of spoofing.

4.3.2.1 *Summary of ML-Algorithm*

The collaborating defence mechanism against GPS spoofing attack is in fact a simulated approach, not deployed into the AHD. In this scenario, N vehicles generated by CARLA simulator collect and exchange measurements like absolute position, relative distance and relative angle, via V2V communication and sensors such as GPS and LIDAR. Afterwards, the collected measurements are broadcasted to a fusion centre, namely 5G cloud, which in turn will estimate and inform vehicles about their locations more accurately than GPS actually does. However, an attacker may spoof a number of vehicles of vehicular ad hoc network (VANET), exploiting well known GPS vulnerabilities and thus severely degrade GPS accuracy. To this end, a general cost function can be formulated, correlating the broadcasted measurements and the true positions of vehicles with the impact of spoofing, and minimized by the cloud in a centralized manner. As described in deliverable D4.4 [62], the general form of the cost function is provided by:

$$\underset{\boldsymbol{x}^{(t)},\boldsymbol{o}^{(x),(t)}}{\operatorname{argmin}} \left\| \tilde{\boldsymbol{L}}^{(t)}\boldsymbol{x}^{(t)} - (\boldsymbol{b}^{(x),(t)} - \boldsymbol{q}^{(x),(t)}) \right\|^2 + \lambda_3 \left\| \boldsymbol{o}^{(x),(t)} \right\|_1$$

$$\underset{\boldsymbol{y}^{(t)},\boldsymbol{o}^{(y),(t)}}{\operatorname{argmin}} \left\| \tilde{\boldsymbol{L}}^{(t)}\boldsymbol{y}^{(t)} - (\boldsymbol{b}^{(y),(t)} - \boldsymbol{q}^{(y),(t)}) \right\|^2 + \lambda_4 \left\| \boldsymbol{o}^{(y),(t)} \right\|_1$$

Extended Laplacian matrix $\tilde{L}^{(t)}$ describes the connectivity topology of VANET, vectors $x_i^{(t)}$ and $y_i^{(t)}$correspond to the location of vehicles that must be estimated, vectors $b^{(x),(t)}$and $b^{(y),(t)}$contain the broadcasted measurements, while $o^{(x),(t)}$ and $o^{(y),(t)}$denote the outliers of GPS position, caused by the spoofing. CVX software can be utilized to estimate the positions and the outliers of spoofing.

As described in deliverable D4.3 [6], a vector containing the Euclidean distances between the GPS and the estimated locations is formed. Afterwards, k-means clustering algorithm, with k = 2, is applied on the corresponding distances, producing two clusters with associated centroids. Cluster with the largest centroid contains, in fact, the distances that correspond to spoofing. As such, the ids of compromised vehicles can be identified. Finally, cloud informs vehicles about their estimated (and without the impact of attack) locations and raises an alert if a specific vehicle is spoofed.

4.3.3 Introduction of In-vehicle sensor fusion solution for detecting location spoofing attacks

The implementation is based on the in-vehicle sensor fusion solution for detecting location spoofing attacks that was developed in T4.3 and described in D4.3 [6]. The solution computes a parallel GPS-free location stream using a localization/tracking algorithm based on Extended Kalman Filter (EKF) which consists of a *prediction* step and a follow-up *update* step. First, in the *prediction* step, the future location of the vehicle is estimated based on the previous vehicle location by using a vehicle mobility model (i.e., bicycle model) and fusing readings from the vehicle's on-board sensors. Second, in the *update* step, the EKF approach is applied to refine the predicted vehicle's location with GPS-free global location information. For instance, a localization algorithm that processes Signals of Opportunity (SoO) received by a dedicated hardware receiver in the vehicle could be used to obtain such GPS-free global location measurements with some uncertainty, as described in D4.3 [6]. To identify a possible GPS location spoofing attack, the estimated vehicle's location after the *update* step is compared with the current GPS location reading (i.e., *comparison* step) in terms of their deviation; if the deviation is larger than a pre-defined *threshold*, then an attack is detected and an alarm is raised. In the following we describe how the solution is implemented on the AHD.

4.3.3.1 *Summary of ML Algorithm*

The ML algorithm is applied *offline* to select the threshold value that is used by the solution during the *online* operation (i.e., when the vehicle is moving) to detect a location spoofing attack. Essentially, if the deviation between the estimated vehicle's location and the current GPS location exceeds the pre-determined threshold value, then an attack is detected. The deviation is defined by means of the distance between these two locations and different distance metrics can be used. For instance, the well-known Euclidean distance $e_d = \left||x - \hat{x}|\right|$ can be used, where x and $\hat{x}$, are the GPS location and the estimated location, respectively. Alternatively, the more sophisticated Bhattacharyya distance can be used, which also considers the inherent uncertainties in the two locations, in the form of corresponding covariance matrices; see D4.3 [6] for the details. This is feasible in practice, because commercial GPS receivers not only provide the (Long, Lat) location of the vehicle, but also an uncertainty score, i.e., the horizontal accuracy in meters[2], that can be transformed into a covariance matrix. The estimated vehicle location provided by the EKF location is also accompanied by a covariance matrix that represents the uncertainty in this estimated location, which is dynamically computed and adjusted as the vehicle is moving and new sensor measurements are incorporated.

In short, the ML algorithm for selecting the threshold works as follows: Initially, an attack-free time period is assumed while the vehicle is moving and a set of n locations $\hat{x}_i$ and x_i is collected where $i = 1, \dots, n$ denotes the location samples. The respective distances between the locations are computed, using either the Euclidean or the Bhattacharyya distance, leading to a vector of distances e_d. Thus, the Empirical Cumulative Distribution Function (ECDF) of e_d can be obtained, which is essentially the sorted version of the distances from the lowest to the highest value. Assuming that the system can tolerate $(1 - \alpha)$% false positives, then we can select the threshold as the a_{th} percentile of the distance errors e_d, chosen from the ECDF, as shown in D4.3 [6] (Figure 3.9c and Figure 3.12c).

To robustify the solution and overcome unexpected fluctuations, either at the GPS locations or the EKF-estimated vehicle locations or both, that introduce undesirable errors in the distance values, filtering can be applied to the distance values by means of a sliding window, e.g., moving average, as shown in D4.3 [6] (Figure 3.7). Alternatively, filtering can be applied to the attack detection level, where a sliding window can be used for a more robust decision based on majority voting. In this case, a decision for GPS location attack detection is taken based on a set of subsequent detection estimates of size defined by the length of the sliding window.

[2] The horizontal GPS accuracy is typically provided by commercial GPS receivers as the radius of 68% confidence. That is, if we draw a circle centred at the location's latitude and longitude, and with a radius equal to the accuracy value, then there is a 68% probability that the true location is inside the circle.

4.3.3.2 *Deployment into Anti-Hacking Device*

Docker is an open-source platform used primarily for developing, shipping, and running applications, while Docker containers are lightweight, can be shared with others effortlessly, and there is no need to count on what is presently installed on the operating system of the host. The in-vehicle attack detection solution is tested in the NVIDIA Jetson Nano 4GB device, which enables a fast, efficient and cost-effective deployment.

The algorithm for selecting the threshold can be applied offline using either data from a real vehicle or synthetic data from the CARLA simulator. Therefore, the solution can be applied to dynamic environments without any assumption about the magnitude of the GPS location spoofing attack.

Given the attack detection threshold, the solution takes as inputs: i) the speed of the vehicle, ii) the compass reading, iii) the GNSS reading, i.e., vehicle position: Lat, Long, Altitude and Time information, iv) the steering angle of the vehicle, and v) the IMU sensor reading including the gyroscope and accelerometer values. In addition, the estimated location of the vehicle required for the update step is provided by an external hardware/software component, e.g., an SDR-based implementation of the localization algorithm that uses SoO measurements.

For testing purposes, the attack injection is carried out on the AHD itself; however, in real-life conditions the GPS locations will be already disturbed by the attack. The AHD hosting the attack detection solution can be tested in the following two scenarios:

- Real-life environment – The in-vehicle GPS location spoofing attack detection solution is deployed in the embedded AHD and uses the vehicle's sensory data as real-time input sources.
- Simulation environment – The in-vehicle GPS location spoofing attack detection solution is deployed in the embedded AHD, while the sensors of an autonomous vehicle agent instantiated within the CARLA simulator are used to provide the required inputs to the AHD. That is, the data are collected in the CARLA simulator and then stored on the AHD. These data may cover different scenarios with respect to the magnitude of the attack (e.g., in the form of additional noise or constant bias in the GPS readings) and the accuracy of the localization algorithm. The attacked data are transferred to the AHD to run and test the solution.

4.3.3.2.1 Pre-processing Activities

No special data pre-processing activities are required and the attack detection solution can process the raw sensor data, as described above. Generic data pre-processing techniques can be applied, if needed, including i) outlier detection and mitigation methods, ii) handling missing data, iii) time synchronization of sensory data delivered with variable sampling frequencies, e.g., up/down-sampling etc.

4.3.3.2.2 Implementation into Anti-Hacking Device

We used Docker to package the in-vehicle sensor fusion attack detection solution for execution in the AHD.

Prerequisites:

1. The device must be connected to the internet until the installation process complete.
2. The Docker package must be installed in the operating system. It can be download from the following link, https://docs.docker.com/get-docker/

In the beginning, the user has to download the Docker container we created in the AHD. To install, the user must enter a specific directory that contains a file named "Dockerfile" and run the command:

```
$ sudo docker build -t caramel/gpsspoofingdetection:jetsonV01 .
```

The "Dockerfile" will build the Docker image and install all the necessary packages and dependencies. After building the Docker image, it is essential to confirm that the image is shown in the list and to check it the user needs to enter the following command:

```
$ sudo docker images
```

If the image is shown in the list, then the image is built successfully and the user can run it to see the results. To run the image, the user needs to enter the command below:

```
$ sudo docker run caramel/gpsspoofingdetection:jetsonV01
```

After running the command, it will show desired output in the terminal.

```
|── Dockerfile
|── src/
|   |── main.py
|   |── measurments.csv
|── README.md
|── requirements.txt
```

Figure 14: Structure of directory.

4.3.3.2.3 Model performance measurement within the Anti-Hacking Device

We tested the python script implementing the proposed solution on two different devices. These are:

1. Desktop PC with NVIDIA
2. NVIDIA Jetson Nano embedded computing device

On the desktop PC, we tested the python script in two ways to measure the execution time using a dataset that contains a total of 1,749 data points, i.e., all the sensor readings that are needed for the solution to work. Firstly, we ran the script in the terminal by using the python command, and secondly, we ran the Docker image built with the same python script. The overall execution time is less than 1.5 seconds in both cases, as shown in Table 13. When the Docker image is deployed and runs on the Jetson Nano device, it is expected that the execution time will be increased and it is measured around 12 seconds for the whole the dataset. Still the execution time per data point is around 7 milliseconds.

Note that using the using either the Euclidean or the Bhattacharyya distance has only marginal effect on the execution time, while the time is measured using a sliding window of size 5.

Table 13: Execution time of python script on devices.

Device	Method	Execution Time
Desktop PC	Using Terminal	01.30 seconds
	Using Docker Image	01.40 seconds
NVIDIA Jetson Nano	Using Docker Image	11.47 seconds

4.4 Attack on the V2X message transmission

4.4.1 Introduction of Attack on the V2X message transmission

As specified in D2.1 [24], there are 4 use cases focused on the attacks on the V2X message transmission:

- Use case 1: The attacker is a fake vehicle that generates messages with some invalid data.
- Use case 2: The attacker is a fake vehicle that sniffs and replays messages of compliant vehicles.
- Use case 3: The attacker is a compliant vehicle but supplanting identity (example: a normal vehicle sends information as an ambulance).
- Use case 4: The attacker is a vehicle trying to track another one which changes the Authorization Tickets (AT) using the proposed changing algorithm.

The counterattack of the three first use cases are based on the standard ETSI ITS-G5 security procedures and protocols deploying a Public Key Infrastructure (PKI) with a new distribution system of Revoked Certificates Lists, nevertheless, they do not use ML techniques. On the other hand, use case 4 plans to use a ML algorithm which is explained in detail in D4.1 and also it is summarized hereafter.

In this scenario, connected vehicles communicate with the infrastructure and with the surrounding vehicles through V2X messages. These messages contain information such as the position, the velocity and other parameters of the vehicle as is stated by ETSI[3]. Additionally, the messages are authenticated with the AT which also acts as a pseudo-anonymous signature.

A malicious spoofer, listening to all the V2X messages in an area, might try to track a targeted vehicle by analyzing all V2X messages signed with the same AT. This spoofer would obtain the trajectory of the car and this information could be cross correlated with additional data obtained by other means that would reveal the identity of the driver. To counterattack these attempts against privacy, the connected vehicle can change the AT at any time and break the tracking line that the spoofer has been following. But even in this scenario, a spoofer can try to reconstruct the trajectory of a vehicle, by associating messages signed with different AT to the same car. Different algorithms, including several ML approaches, can be implemented to accomplish this task by considering all the information available in the V2X messages. In order to mitigate the tracking capabilities of the spoofers a ML algorithm is proposed in section 4.4.1.1.

4.4.1.1 *Summary of ML-Algorithm*

To mitigate the effectiveness of these tracking algorithms, the AT scheduler algorithm depicted in Figure 15 is proposed. This algorithm decides when is the best moment to change the AT of a car by evaluating how easily is to track the car considering the V2X messages in the area. More precisely, a tracker tries to associate each V2X message sent by the target car with a subgroup of old messages that were sent by the same vehicle. Although the length of this subgroup can be variable, in practice, the tracker only needs to link the new message with the latest message sent by the target car to track it. The system stores a buffer of old V2X messages coming from the surrounding vehicles and the target car itself and is trained to discern which of those old messages correspond to the same target vehicle. This way, it is possible to evaluate periodically how well the car can be tracked by an external spoofer. If the target messages are associated with the correct subgroup with a high score, it means that the car can be easily tracked. This tracking score is computed every time the car wants to send a V2X message, and with these scores over time and considering the remaining ATs available in the car and the time remaining to get new ATs, an AT scheduler is implemented. This AT scheduler implements an optimal stopping algorithm and is in charge of deciding when is the best moment to change the AT to make more difficult the tracking to the spoofers.

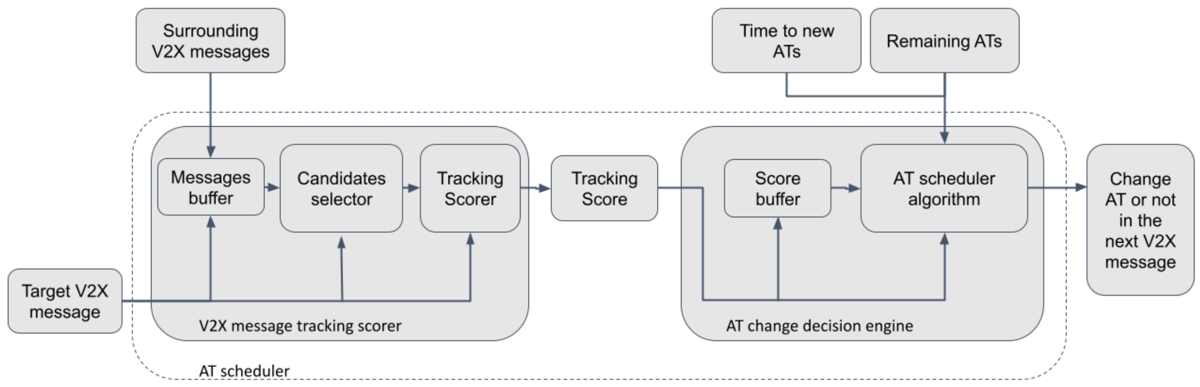


Figure 15: AT Scheduler Algorithm pipeline.

[3] ETSI EN 302 637-2, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service"

4.4.1.2 *Deployment into Anti-Hacking Device*

4.4.1.2.1 Pre-processing Activities

The AT scheduler requires a pipeline that transforms the raw V2X messages to actual features needed by the ML algorithm. The incoming V2X messages are parsed in order to extract the features considered by the AT scheduler V2X messages tracking scorer. These features may include, but are not limited to:

1. The time difference between the received messages and the target car's generated messages.
2. The variation between the message position coordinates and the expected position that the target car should have at the time of the received message.
3. The variation between the message's velocity, acceleration and jerk components and the target's car expected values of those components at the time of the received message.

Then the computed features are buffered inside the AT scheduler. Additionally, the candidates that enter the tracking scorer modules are filtered by a candidate selector algorithm. This candidate's selector algorithm only takes the features of the messages that are most likely to belong to the same car by sorting the features of the messages that have the smallest variations with respect to the target car.

4.4.1.2.2 Implementation into Anti-Hacking Device

The three main modules that compose the AT scheduler are implemented in a Docker container for easy hardware-agnostic deployment in an AHD. This approach provides an isolated and independent environment.

4.4.1.2.3 Model performance measurement within the Anti-Hacking Device

The system performance has been tested on a Raspberry Pi v3[4] with 924 MB of RAM (see Table 15 for full specifications). Although the device has a 4-core CPU, only one core has been used to perform the test. Results are shown in Table 14. The process of extracting the features of the last 50 messages in the buffer takes approximately 50 ms. The most probable 5 candidates are then selected to be classified by a random forest with 100 trees of max depth 7, which takes another 70 ms. This execution time can be further reduced by limiting the number of trees and depth of the random forest. Overall, the system can process approximately 30 messages per second, both messages from the surrounding cars and target messages. This is assuming that a target message is sent every 100 ± 50ms and that the messages sent by the surrounding cars are just added to the buffer. The system takes 406 MB of memory and uses all the available processing capabilities of a single CPU core.

Table 14: Model performance measurements of the V2X message tracker

	Raspberry Pi v3
Processing capability	30 msg/s
RAM usage	406 MB
CPU usage	100% of a single core
Feature extraction of 50 candidates	~70 ms
Random forest classification of N=5 candidates	~70 ms

Table 15: Raspberry Pi v3 specifications

[4] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

SoC	Broadcom BCM2837
CPU	4× ARM Cortex-A53, 1.2GHz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 Ethernet, 2.4GHz 802.11n wireless

4.5 Smart Charging Abuse

4.5.1 Introduction of Smart Charging Abuse

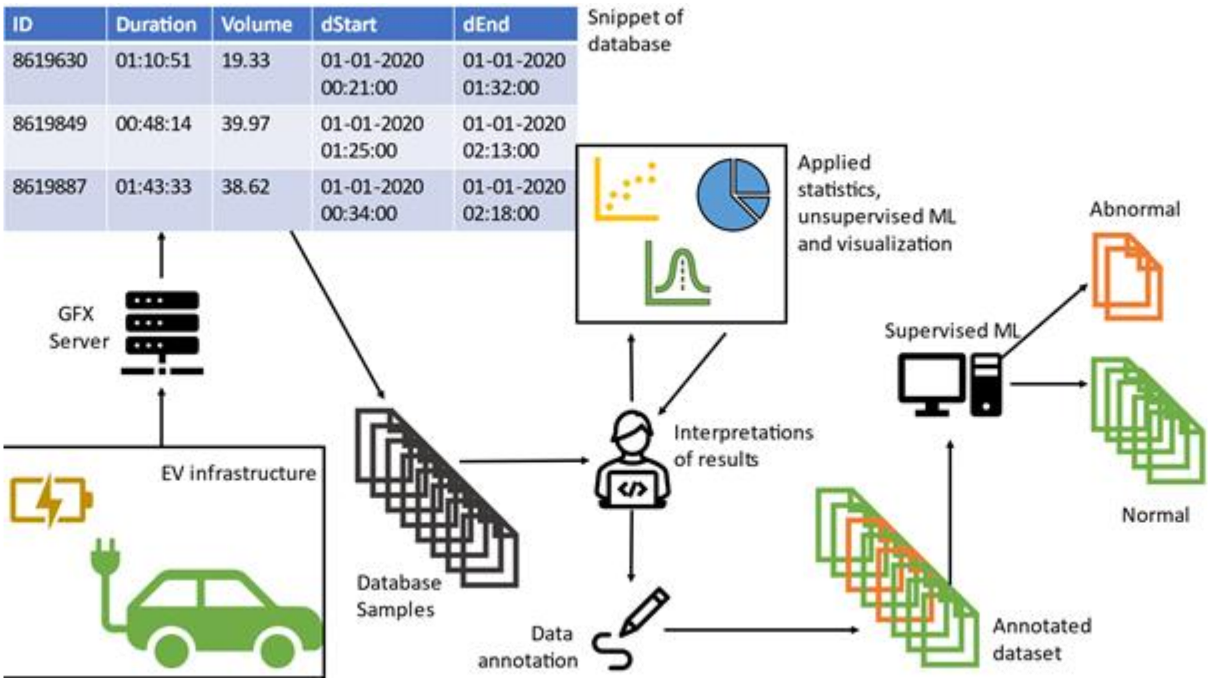
The smart charging abuse scenario in CARAMEL consists of attacks in the charging stations creating an unavailability in the service. The attackers engage in a number of connections (physical or remote bypassing the authentication scheme) with the charging stations performing connect and disconnect actions that lead to huge load to the electrical grid. The different connections could bring the GreenFlux's (GFX) infrastructure on the limit of its capacity and create a chain of problems and failures to European distribution system operators (DSOs) and transmission system operators (TSOs).

The goal of this scenario is to quickly detect the attack with the usage of a ML-pipeline designed to cover the needs of the use case while also providing a decent level of abstraction to be adopted from industrial partners outside of the consortium. GFX software with the integration of the ML-algorithm for the Smart Charging Abuse will be able to detect those attacks, store them in an incident database, and inform the system administration for the potential threats.

4.5.1.1 *Summary of ML-Algorithm*

For the Smart Charging Abuse detection, a semi-supervised ML-pipeline is used. The usage of semi-supervised learning provides flexibility in anomaly detection especially to unlabelled datasets that contain both normal and abnormal data. The Smart Charging Abuse ML-pipeline solves the problem that is created with the lack of labels in the dataset and it can detect abnormal data in order to build a model for the classification of any new incoming data without class bias.

The first step of the ML-algorithm is done with the usage of applied statistics, unsupervised ML and visualization. Then, the output data can be used as a primary observation by the human factor and early interpretations are possible. That same data (output data from the first step of the ML-algorithm) from the unsupervised learning are annotated in order to go through the supervised learning for the final decision whether a smart charging abuse has been made (normal class) or not (abnormal class).



ID	Duration	Volume	dStart	dEnd
8619630	01:10:51	19.33	01-01-2020 00:21:00	01-01-2020 01:32:00
8619849	00:48:14	39.97	01-01-2020 01:25:00	01-01-2020 02:13:00
8619887	01:43:33	38.62	01-01-2020 00:34:00	01-01-2020 02:18:00

Figure 16: Semi-supervised Machine-Learning pipeline

The proposed human-in-the-loop methodology offers constant feedback to the ML pipeline improving it while offering adaptability and resilience on zero-day attacks. Moreover, the organizations that select to adopt the method feel more confident for the level of cyber-security in their infrastructure since a security administration has the overview of the system and can intervene whenever it is required. Finally, the outcome of the algorithms can be integrated into existing security information exchange protocol (SIEM) and intrusion detection system (IDS) solutions enhancing the cyber-security of the organization.

4.5.1.2 *Deployment into Anti-Hacking Device*

The CARAMEL AHD is a secure and lightweight offering additional security to the vehicles that will be deployed. Due to its lightweight nature dictated from limitations of every integrated system, it will not be deployed as a physical device in this pillar. Instead, the deployment of the ML pipeline will take place in the GFX's infrastructure using only the software architecture of the AHD, in a virtualized environment on a cloud service where the virtualized AHD will communicate with the main infrastructure updating it in near-real time according to the latest updates.

4.5.1.2.1 Pre-processing Activities

The GFX security administrators need to train the ML system at regular intervals (e.g. bi-monthly), assign the automatically execution of the algorithms in short regular intervals (less than 15 minutes) while also offer human supervision in the entire process of the semi-supervised ML pipeline.

4.5.1.2.2 Implementation into Anti-Hacking Device

Based on D2.1, in the electromobility scenario the AHD will not be physically deployed in electrical vehicles (EVs) as in the other use cases [24]. Instead, the anti-hacking services for the smart charging abuse scenario will be offered via a cloud services in a virtualized environment. The deployment and execution of the semi-supervised ML pipeline takes place in the GFX infrastructure.

4.5.1.2.3 Model performance measurement within the Anti-Hacking Device

In the smart charging abuse scenario, the semi-supervised ML pipeline is committed to inform the connected users for a potential threat in less than 15 minutes. The 15 minutes time limit defines the upper boundaries for both the execution of the ML algorithms.

The implementation details of the anti-hacking services for the smart-charging abuse scenario will be provided with the system integration in the WP6, Task 6.1 “Integration Configuration and Deployment”.

5 Conclusion

This deliverable has summarized all machine learning ML and few of the heuristic's algorithms for attack detection and mitigation being developed as part of CARAMEL in WP3 and WP4. Also, the deliverable discusses the deployment strategy of the algorithms in the CARAMEL environment and anti-hacking device (AHD) by focusing on pre-processing, implementation, and performance measurement for the algorithms.

Section 2 of the deliverable provides a quick overview of CARAMEL and the three key areas for the project i.e. Pillar 1: Autonomous vehicle, Pillar 2: Connected vehicle and Pillar 3: Electro mobility, including CARAMEL architecture and scope.

Section 3 provides details of AHD. The AHD is designed as a passive intrusion detection device that is integrated as an additional controller into the vehicle. The AHD *passively* listens to the car's internal systems including communication busses to collect raw data from sensors & communication controllers for processing as well as aggregation. ML / other heuristics algorithms are applied on this process and aggregated data to detect & mitigate possible attacks scenarios. It then *actively* creates attack reports (events) and sends them to the CARAMEL backend. The AHD encapsulates the actual detection algorithms and also some system services into Docker containers. This has several advantages: It allows update of detection algorithms without a full firmware update. Additionally, the detection algorithms are separated by the protections offered by the Docker runtime against any mutual interference.

In Section 4, project partners have documented how algorithms developed in WP 3 & WP 4 of the CARAMEL project can be integrated as Docker containers into the different hardware versions of the AHD, giving configuration and performance details for scenarios from all three pillars. In Pillar 1, for physical adversarial attack use case, various deep learning models for detection and mitigation of anomaly traffic signs were integrated in two different types of AHD hardware's. A detailed evaluation of the models in terms of inference speed in different AHD hardware's were presented. For image deterioration attack use case, it has been demonstrated how the approach for detection / mitigation can be implemented on an embedded AHD with promising results. Further improvements will be made to this initial implementation during the integration phase of the project. In case of adversarial attack on camera sensor use case, a ML solution has been proposed for mitigation. By combining outputs from different modalities, robust solutions and improve situational awareness can be created. Thus, the LIDAR sensor has been utilized to increase safety and mitigate attacks on the camera sensor. Also, the proposed solution has been implemented into the AHD and has been validated through extensive experiments using realistic data collected with the CARLA simulator.

In Pillar 2, in-vehicle location spoofing detection solution was implemented in python and deployed on the AHD (i.e., NVIDIA Jetson Nano embedded device) demonstrating execution time of less than 10 milliseconds per data point to successfully detect a GPS location spoofing attack. For attack on V2X message transmission use case, results show that the authorization tickets (AT) scheduler running on the AHD can analyses the stream of V2X messages received by the vehicle in order to decide the best moment to change the AT and avoid the tracking of the vehicle. The AHD in which the AT scheduler was tested was a Raspberry PI v3 that could process up to 30 V2X messages per second. Assuming that, each vehicle sends around 10 V2X messages per second (1 every 100 ms), the Raspberry PI would be capable of processing all the V2X messages sent by three surrounding vehicles. If there were more vehicles producing V2X messages around (i.e.: in dense traffic situations), it would be necessary to ignore some of the V2X messages received. In case it was needed to process a higher number of messages, it would be necessary to improve the efficiency of the ML model of the AT scheduler or host it in a more powerful AHD hardware.

In Pillar 3, for smart charging abuse use case, the software architecture of the AHD will be adopted and deployed as a virtualized Docker-based anti-hacking solution to the cloud that inherits the principles of separation and update agility of detections mechanisms from the physical devices designed for vehicle integration.

References

[1]. CARAMEL: D3.2 Cyberthreat Detection Using Sparse and Deep Priors – Interim - Final. 2020.
[2]. CARAMEL: D3.4 Multi-modal Data Fusion Module for Responding Reliably to the Threats - Interim - Final. 2020.
[3]. CARAMEL: D3.6 – Cyberthreat Detection and Response Techniques for Cooperative Automated Vehicles. 2021.
[4]. CARAMEL: D3.7_Cyberthreat Detection and Response Techniques for Plug-in Electrical Vehicles. 2021
[5]. CARAMEL: D4.2 Robust to Cyberattack Machine Vison Models Based on Improved Training Methods and Anomaly Detection Deep Networks. 2021
[6]. CARAMEL: D4.3 CARAMEL Situational Awareness Solution based on the Machine Learning Applications. 2021
[7]. CARAMEL: D2.4 System Specifications and Architecture. 2020.
[8]. "Products | Coral", Coral. [Online]. Available: https://coral.ai/products/. [Accessed: Oct- 2020].
[9]. "Autonomous Machines", NVIDIA Developer. [Online]. Available: https://developer.nvidia.com/embedded-computing. [Accessed: Oct- 2020].
[10]. "Intel® Neural Compute Stick 2 Product Specifications", Ark.intel.com. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html. [Accessed: 12- Oct- 2020].
[11]. "Technology | Coral", Coral. [Online]. Available: https://coral.ai/technology/. [Accessed: Oct- 2020].
[12]. Coral.ai. [Online]. Available: https://coral.ai/products/accelerator/. [Accessed: Oct- 2020].
[13]. "Mini PCIe Accelerator | Coral", Coral. [Online]. Available: https://coral.ai/products/pcie-accelerator/. [Accessed: Oct- 2020].
[14]. "M.2 Accelerator A+E key | Coral", Coral. [Online]. Available: https://coral.ai/products/m2-accelerator-ae. [Accessed: Oct- 2020].
[15]. "Dev Board | Coral", Coral. [Online]. Available: https://coral.ai/products/dev-board/. [Accessed: Oct- 2020].
[16]. M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", arXiv.org, vol. 2, 2016. [Accessed 2020].
[17]. K. Team, "Keras: the Python deep learning API", Keras.io. [Online]. Available: https://keras.io/. [Accessed: Oct- 2020].
[18]. "TensorFlow Lite guide", TensorFlow. [Online]. Available: https://www.tensorflow.org/lite/guide. [Accessed: Oct- 2020].
[19]. "Jetson Nano Developer Kit", NVIDIA Developer. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed: Apr- 2021]
[20]. "Jetson AGX Xavier Developer Kit", NVIDIA Developer. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit. [Accessed: Apr- 2021]
[21]. S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, 'Detection of traffic signs in real-world images: The German traffic sign detection benchmark', in The 2013 International Joint Conference on Neural Networks (IJCNN), Aug. 2013, pp. 1–8, doi: 10.1109/IJCNN.2013.6706807.
[22]. A. D. Kumar, K. N. R. Chebrolu, and S. KP, 'A brief survey on autonomous vehicle possible attacks, exploits and vulnerabilities', arXiv preprint arXiv:1810.04144, 2018
[23]. K. Eykholt et al., 'Robust Physical-World Attacks on Deep Learning Visual Classification', 2018, pp. 1625–1634, Accessed: Nov. 14, 2020. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/html/Eykholt_Robust_Physical-World_Attacks_CVPR_2018_paper.
[24]. CARAMEL: D2.1 - Report on Detailed Specification of Use Cases. 2020.
[25]. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, 'MobileNetV2: Inverted Residuals and Linear Bottlenecks', 2018, pp. 4510–4520, Accessed: Nov. 17, 2020. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html.
[26]. He, J. D. (2019). Modulating Image Restoration With Continual Levels via Adaptive Feature Modification Layers. CVPR.

[27]. Chen, L.-C. e. (2017). Rethinking Atrous Convolution for Semantic Image Segmentation. ArXiv.
[28]. S. Shi, X. W. (2019). PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-779.
[29]. A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 2012, pp. 3354-3361, doi: 10.1109/CVPR.2012.6248074.
[30]. J. Petit, B. Stottelaar, and M. Feiri, "Remote Attacks on Automated Vehicles Sensors : Experiments on Camera and LiDAR," Comput. Sci., 2015, Accessed: Sep. 18, 2020. [Online]. Available: /paper/Remote-Attacks-on-Automated-Vehicles-Sensors-%3A-on-Petit-Stottelaar/e06fef73f5bad0489bb033f490d41a046f61878a.
[31]. K. N. Truong, S. N. Patel, J. W. Summet, and G. D. Abowd, "Preventing Camera Recording by Designing a Capture-Resistant Environment," in UbiComp 2005: Ubiquitous Computing, Berlin, Heidelberg, 2005, pp. 73–86, doi: 10.1007/11551201_5.
[32]. Now from Nationwide ®. 2014. A History Of Car Safety Innovations [Infographic]. [online] Available at: https://blog.nationwide.com/car-safety-timeline-infographic/
[33]. A. Boloor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, and X. Zhang, "Attacking vision-based perception in end-to-end autonomous driving models," J. Syst. Archit., vol. 110, p. 101766, Nov. 2020, doi: 10.1016/j.sysarc.2020.101766.
[34]. Q. He, X. Meng, and R. Qu, "Towards a Severity Assessment Method for Potential Cyber Attacks to Connected and Autonomous Vehicles," J. Adv. Transp., vol. 2020, pp. 1–15, Sep. 2020, doi: 10.1155/2020/6873273.
[35]. B. Sheehan, F. Murphy, M. Mullins, and C. Ryan, "Connected and autonomous vehicles: A cyber-risk classification framework," Transp. Res. Part Policy Pract., vol. 124, pp. 523–536, Jun. 2019, doi: 10.1016/j.tra.2018.06.033.
[36]. S. Parkinson, P. Ward, K. Wilson, and J. Miller, "Cyber threats facing autonomous and connected vehicles: Future challenges," IEEE Trans. Intell. Transp. Syst., vol. 18, no. 11, pp. 2898–2915, 2017.
[37]. S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," Proc. IEEE, vol. 107, no. 8, pp. 1697–1716, 2019.
[38]. Y. Man, M. Li, and R. Gerdes, "GhostImage: Remote Perception Attacks against Camera-based Image Classification Systems," p. 16.
[39]. J. Petit and S. E. Shladover, "Potential Cyberattacks on Automated Vehicles," IEEE Trans. Intell. Transp. Syst., vol. 16, no. 2, pp. 546–556, Apr. 2015, doi: 10.1109/TITS.2014.2342271.
[40]. A. Qayyum, M. Usama, J. Qadir, and A. Al-Fuqaha, "Securing connected & autonomous vehicles: Challenges posed by adversarial machine learning and the way forward," IEEE Commun. Surv. Tutor., vol. 22, no. 2, pp. 998–1026, 2020.
[41]. S. Meryem and T. Mazri, "Security study and challenges of connected autonomous vehicles," in Proceedings of the 4th International Conference on Smart City Applications, 2019, pp. 1–4.
[42]. Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang.Image restoration using convolutional auto-encoders with symmetric skip connections, 2016.
[43]. P. Milanfar. A tour of modern image filtering: New insights and methods, both practical and theoretical. IEEE Signal Processing Magazine, 30(1):106–128, 2013.
[44]. J. Snell, K. Ridgeway, R. Liao, B. D. Roads, M. C. Mozer, and R. S. Zemel, "Learning to generate images with perceptual similarity metrics", In 2017 IEEE International Conference on Image Processing (ICIP), pages 4277–4281, 2017.
[45]. Y. Zeng, C. Yu, and W. Chang. Fault detection of surveillance camera using fixed-point arithmetic operations. In 2014 IEEE International Conference on Consumer Electronics - Taiwan, pages 87–88, 2014.
[46]. Matt P Wand and M Chris Jones. Kernel smoothing. Crc Press, 1994.
[47]. Tianjun Zhu and Xiaoxuan Yin. Image shadow detection and removal in autonomous vehicle based on support vector machine. Sensors and Materials, 32, 2020
[48]. A. K. Jain, Fundamentals of Digital Image Processing, Prentice Hall, Englewood Cliffs, NJ, USA, 1989.
[49]. R. Gonzalez and R. Woods, Digital Image Processing, Prentice Hall, NewYork, NY,USA, 2nd edition, 2001.

[50]. K. Nallaperumal, R. K. Selvakumar, S. Arumugaperumal, S.Annam, and N. Vishwanath, "Hansa filter: a fuzzy adaptive nonlinear Gaussian filter: an adaptive non local algorithm," International Journal of Imaging and Engineering, vol. 2, no. 1, 2008.
[51]. F. Guichard, L. Moisan, and J.-M. Morel, "A review of P.D.E. models in image processing and image analysis," Journal de Physique, vol. 4, pp. 137–154, 2001.
[52]. J. Weickert, Anisotropic Diffusion in Image Processing, European Consortium for athematics in Industry, B. G. Teubner, Stuttgart, Germany, 1998.
[53]. L. Alvarez, "Images and PDE's," in ICAOS '96 (Paris, 1996), M.-O. Berger, R. Deriche, I. Herlin, J. Jaffr´e, and J.-M. Morel, Eds., vol. 219 of Lecture Notes in Control and Information Sciences, pp. 3–14, Springer, London, UK, 1996.
[54]. T. Barbu, "A PDE based model for sonar image and video denoising," AnAlele Stiintifice Ale Universitatii Ovidius, Constanta, Seria Matematica, vol. 19, no. 3, pp. 51–58, 2011.
[55]. P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," in Proceedings of IEEE Computer Society Workshop on Computer Vision, pp. 16–27, 1987.
[56]. F. Catte, P. L. Lions, J.-M. Morel, and T. Coll, "Image selective smoothing and edge detection by nonlinear diffusion," SIAM Journal on Numerical Analysis, vol. 29, no. 1, pp. 182–193, 1992.
[57]. L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation-based noise removal algorithms," Physica D, Vol. 60, pp. 259–268, 1992.
[58]. L. Rudin and S. Osher, "Total variation-based image restoration with free local constrais," in Proc. IEEE ICIP, Vol. I, pp. 31–35, Austin, TX, USA, 1994.
[59]. D. Mumford and J. Shah, "Optimal approximation by piecewise smooth functions and associated variational problems," Comm. Pure Appl. Math., Vol. 42, pp. 577–685, 1989.
[60]. M.G.Kuhn, "An Asymmetric Security Mechanism for Navigation Signals". In 6th Information Hiding Workshop,Toronto,Canada, 23-25 May 2004. Lecture Notes in Computer science, Vol.3200 (Springer,2004), pp.239-252
[61]. Tianqi Chen Carlos Guestrin "XGBoost: A Scalable Tree Boosting System" KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data MiningAugust 2016 Pages 785–794
[62]. CARAMEL: D4.4 Report on the Fallback Actions for Minimal Risk Condition. 2021.