“LEARNING ASSURANCE”
FOR EMBEDDED SAFETY-CRITICAL APPLICATIONS

DSTI – [DS] PROJECT PRESENTATION – JUNE 2021

Adeline VIEUSSE – 15-June-2021

PRESENTATION OUTLINE

- Project Overview
- Data Management
- Learning Process Management
- Model Training
- Learning Process Verification
- Model Implementation
- Inference Model Verification
- Conclusion
- Annex

PROJECT OVERVIEW
OUTLINE

- Project Overview
  - Project Objectives
  - What is the CoDANN?
  - Use case scenario
- Data Management
- Learning Process Management
- Model Training
- Learning Process Verification
- Model Implementation
- Inference Model Verification
- Conclusion
- Annex

PROJECT OVERVIEW
PROJECT OBJECTIVES

TO "MIMIC" THE DESIGN, DEVELOPMENT AND VERIFICATION PHASES OF A SAFETY-CRITICAL EMBEDDED MACHINE LEARNING SOFTWARE BASED ON THE PRELIMINARY GUIDANCE MATERIAL ISSUED BY THE EUROPEAN CIVIL AVIATION AUTHORITIES (CODANN).

1. To review the **design assurance requirements** for AI-based embedded safety-critical applications

2. To explore the various **techniques** and **tools** available to satisfy these design assurance requirements.

3. To **implement** some of these techniques on a **simple use case scenario**.

PROJECT OVERVIEW
WHAT IS THE CODANN ? – 1/4

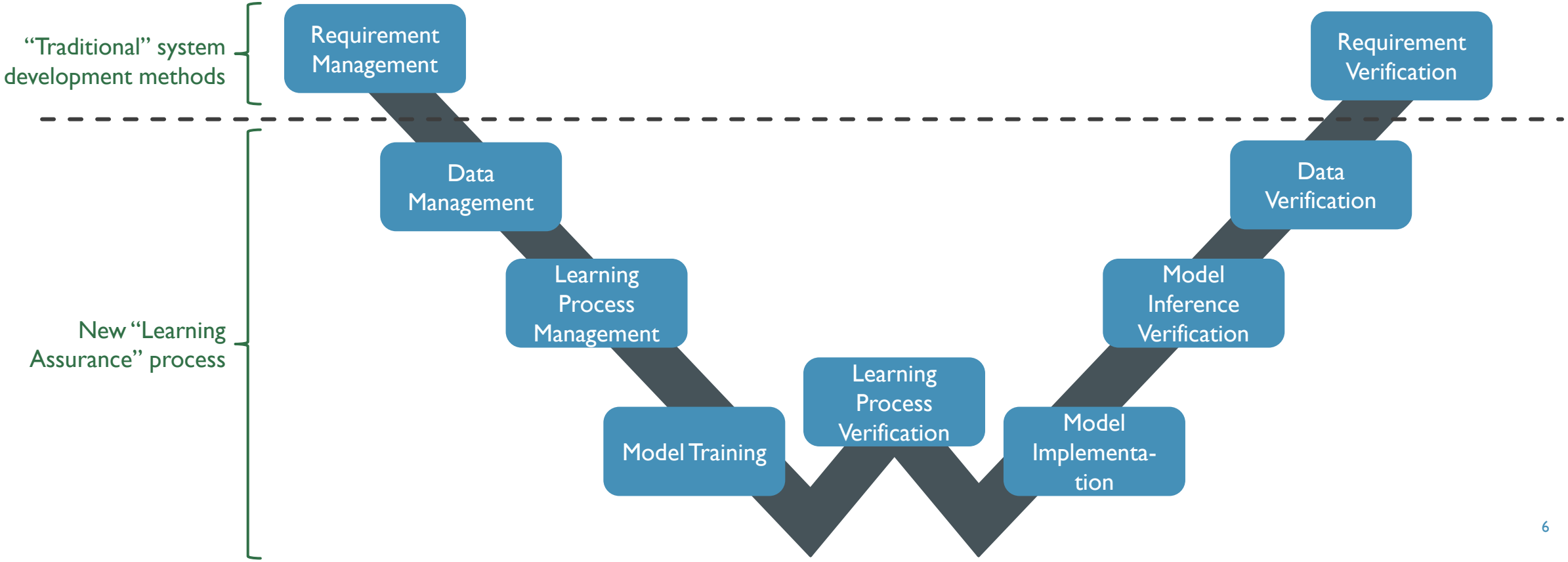
- **CoDANN** = Concept of Design Assurance for Neural Networks [PO1].
- CoDANN Issue 1.0 issued jointly by **EASA** (European Union Aviation Safety Agency) and **Daedalean AG** in March 2020 as part of the EASA AI Roadmap 1.0 [PO2].
- Objective:

  *"Examining the challenges posed by the use of **neural networks in aviation**, in the broader context of allowing machine learning and more generally **artificial intelligence on-board aircraft for safety-critical applications**."*

- Includes:
  - Extensive **theoretical background**,
  - **Application to a use case scenario** - Runway presence detection and corner coordinates computation.
- Focuses on "**Learning Assurance**" and introduces the **W-shaped Learning Assurance Life Cycle**.

PROJECT OVERVIEW
WHAT IS THE CODANN ? – 2/4

W-shaped Learning Assurance Process



PROJECT OVERVIEW
WHAT IS THE CODANN ? – 3/4

- **W-shaped Learning Assurance Process**
  - **Data Management (& Data Verification)** – Identification of the various datasets, preparation of the datasets taking into account requirements such as independence, quality, correctness, completeness…
  - **Learning Process Management** – Preparatory step of the formal training phase (selection and validation of training algorithm, hyperparameters, activation and loss functions, performance metrics…).
  - **Model Training** – Model training and performance assessment on the validation dataset.
  - **Learning Process Verification** – Evaluation of the model performance on the test dataset.
  - **Model Implementation** – Optimisation of the model for transfer to inference hardware.
  - **Inference Model Verification** – Evaluation of the performance of the inference model on the test dataset after transfer to the inference hardware.

PROJECT OVERVIEW
WHAT IS THE CODANN ? – 4/4

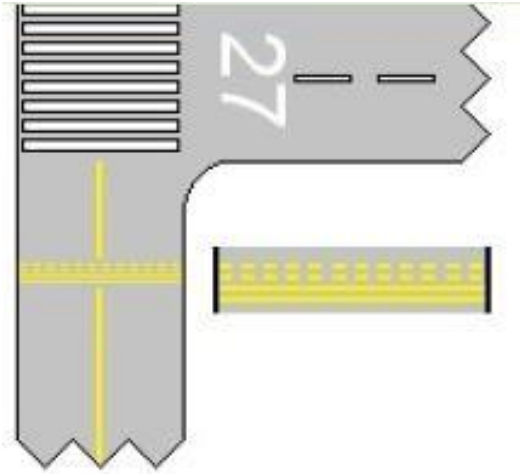
- **CoDANN Scope**
  - **In-scope**
    - Software development
    - Design assurance for machine learning software
    - Machine learning for runway detection and corner coordinates computation
    - Non-adaptative deterministic system
    - Safety Assessment
  - **Out-of-scope**
    - Hardware development
    - Design assurance for "classical" software
    - Machine Learning for end-to-end guidance
    - Adaptative non-deterministic system
    - Security concerns
    - Tools and development environment (hardware and software) qualification

PROJECT OVERVIEW
USE CASE SCENARIO – 1/3

- Runway entry from taxiway is materialised by markings on the ground called "**holding points**".
- An aircraft must obtain clearance from the Air Traffic Control (ATC) to cross the holding point and enter the runway.
- **Runway incursion definition:**

  "Any occurrence at an aerodrome involving the incorrect presence of an aircraft […] on the protected area of a surface designated for the landing and take off of aircraft".

- In 2019 in the US alone, there were 324 reported cases of runway incursion (source).
- Most often with no serious consequences but a runway incursion is the cause of the deadliest accident in aviation history at Tenerife airport in 1977 (source).

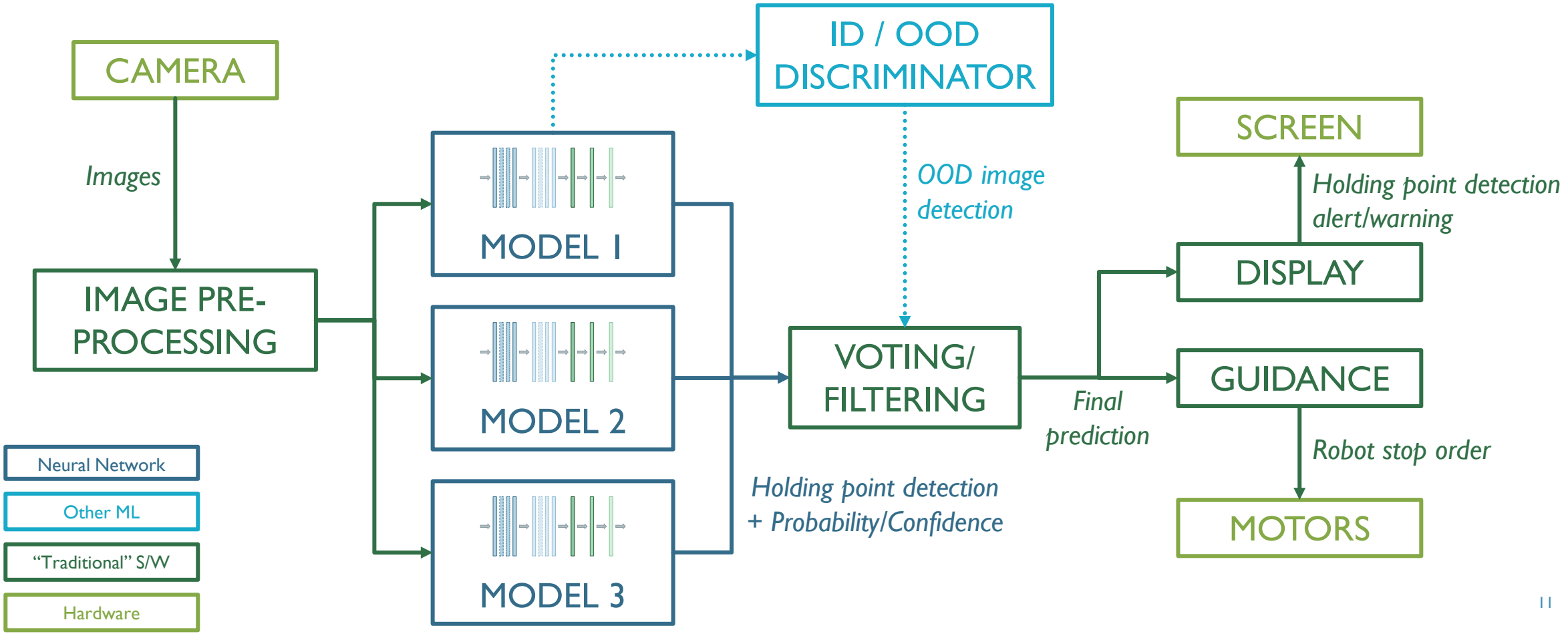


source

PROJECT OVERVIEW
USE CASE SCENARIO – 2/3

- **Proposed use case scenario –** Develop a simplified Holding Point Detection and Runway Incursion Prevention System.
- **Detailed description**
  - **Vehicle** – Waveshare Jetbot (detailed spec)
    - Two-wheeled robot kit,
    - NVIDIA Jetson Nano GPU card,
    - Camera + WiFi + Gamepad controller.
  - **Environment** – Simplified environment, no complex background (cardboard enclosure) and black flooring.
  - "**Mission requirements**" - To detect the presence of a "holding point" (i.e. simple marking on ground), raise an alert and stop the vehicle if no confirmation of clearance is given.

PROJECT OVERVIEW
USE CASE SCENARIO – 3/3

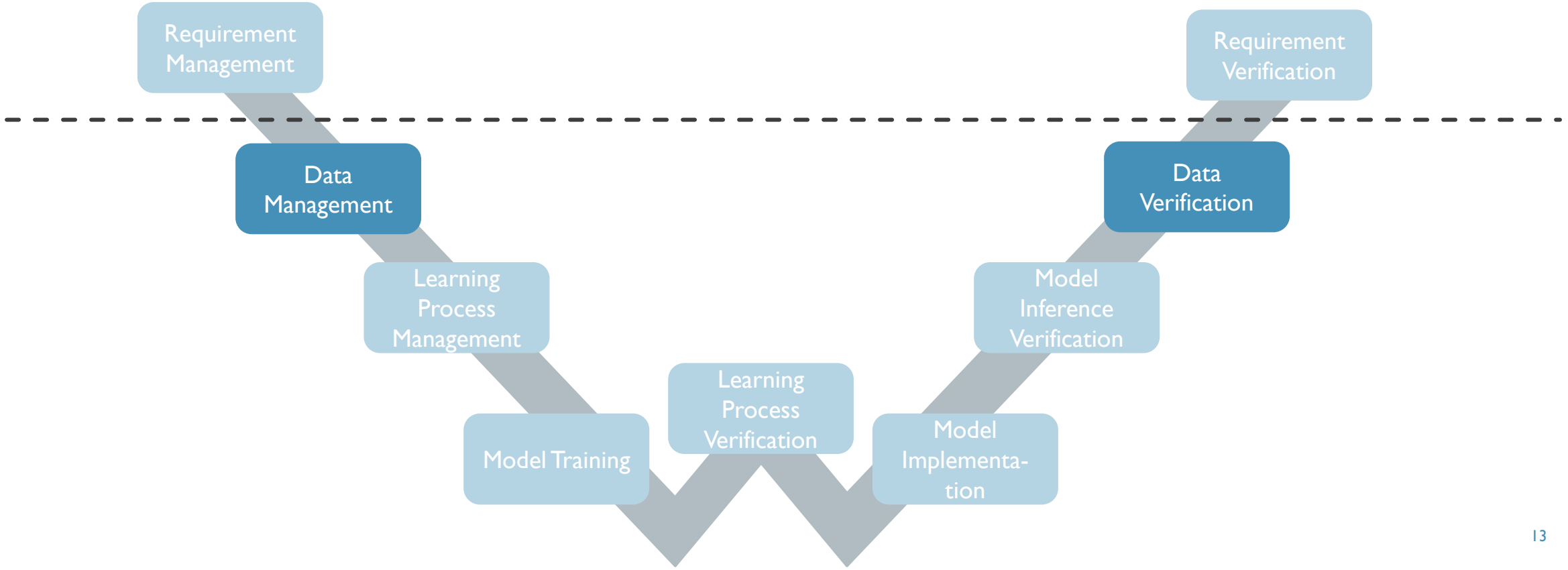


DATA MANAGEMENT
OUTLINE

- Project Overview
- Data Management
  - CoDANN Requirements
  - Application to use case scenario
  - Dataset Completeness
- Learning Process Management
- Model Training
- Learning Process Verification
- Model Implementation
- Inference Model Verification
- Conclusion
- Annex

DATA MANAGEMENT
CODANN REQUIREMENTS – 1/4

W-shaped Leaning Assurance Process



DATA MANAGEMENT
CODANN REQUIREMENTS – 2/4

- The performance of the system in operational conditions depends on the generalization performance of the model resulting from the model's design and development phases.
- But not only… The CoDANN also emphasizes the fact that the **performance of the model "crucially depend on hypotheses on the design dataset"**.
- Consequently, the Data Management phase is very extensively developed in the CoDANN.
- The Data Management requirements are organised into 3 main categories:
  - **Operational Domain Identification,**
  - **Independence,**
  - **Data Quality.**

DATA MANAGEMENT
CODANN REQUIREMENTS – 3/4

- **Operational Domain Identification**
  - Define a list of explicit and interpretable operating parameters.

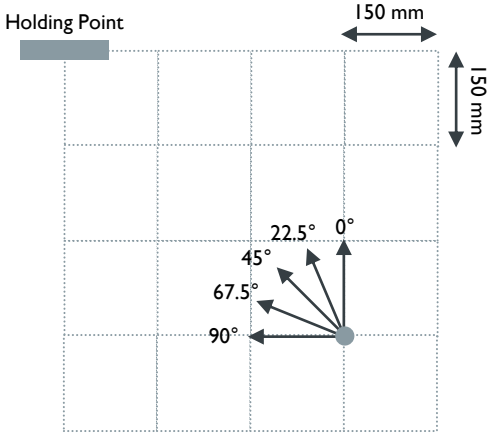
- **Independence**
  - Prepare 3 separate datasets: training, validation and test datasets.
  - Ensure that the training/validation and test datasets are prepared independently.
  - Ensure that the test dataset is not accessed during design phase until the Learning Process Verification phase.

DATA MANAGEMENT
CODANN REQUIREMENTS – 4/4

- **Data Quality**
  - **Accuracy** – Ensure that capture, single source and labelling errors are minimized.
  - **Format and Resolution** – Ensure that the format and resolution are consistent with the intended use.
  - **Digital error protection** – Ensure that the data is not corrupted while stored or in transit.
  - **Traceability** – Ensure ability to determine the origin of each data item during the data collection, annotation and transformation phases:
    - Description of collection protocols,
    - Recording information,
    - Annotation information,
    - Description of data transformation steps.
  - **Completeness** – To be addressed separately.

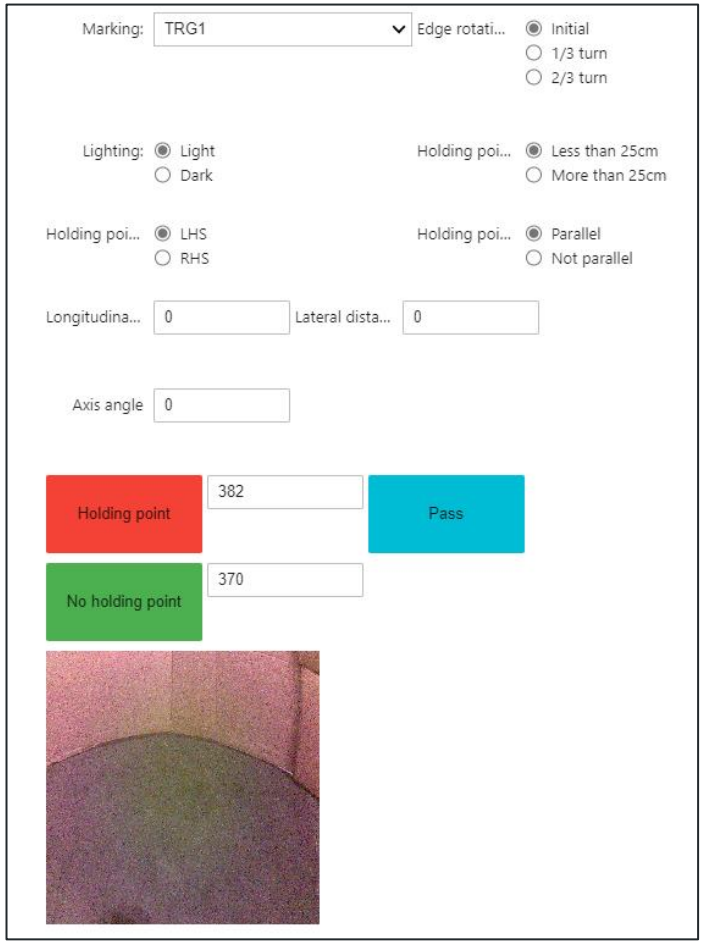
DATA MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 1/4

- **Data Collection**
  - **Environment**
    - Cardboard enclosure + Black flooring
    - Paper strips of different shades (from white to brown) to represent the holding point.
  - **Data Collection Protocol**
    - Grid pattern from 0 to 600 mm (every 150 mm) laterally and longitudinally (i.e. 25 points).
    - One picture taken with angle 0, 22.5, 45, 67.5 and 90 degrees at each point of the grid pattern.



DATA MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 2/4

- **Data Collection (continued)**
  - **Data Collection User Interface** – User interface developed in a Jupyter notebook to record the images and the operating parameters and annotations (in a .csv file) and guide the operator to apply the data collection protocol.
- **Data Augmentation**
  - Approx. 1500 images were collected manually (approx. 750 with holding point and 750 without holding point).
  - Data augmentation (rotation, titling, brightening, darkening…) performed using the PyTorch Augmentor package (documentation).
  - All transformations recorded in a .csv file.



DATA MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 3/4

- **Operational Domain Identification**
  - The following operating parameters have been identified and recorded during the data collection phase:
    - Ambient lighting: bright / dark,
    - Holding point marking colour/contrast with flooring: white / off-white / light brown / dark brown,
    - Holding point side with respect to vehicle: RHS / LHS,
    - Holding point distance from enclosure edge: less than 250 mm / more than 250 mm,
    - Holding point angle with enclosure edge: parallel / not parallel,
    - Longitudinal distance between holding point and vehicle: 0 / 150 / 300 / 450 / 600 mm,
    - Lateral distance between holding point and vehicle: 0 / 150 / 300 / 450 / 600 mm,
    - Angle between holding point and vehicle axis: 0 / 22.5 / 45 / 67.5 / 90 degrees.
- **Independence**
  - Datasets prepared on different days, in different lighting conditions with different markings and enclosure rotation but by the same operator and using the same equipment – Independence not achieved.
  - Test dataset only used during Learning Process Verification phase.

DATA MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 4/4

- **Data Quality**
  - **Accuracy**
    - Capture errors – Data collection module user interface designed to reduce risk of capture errors.
    - Single source error – Only one source (Jetbot) used during data collection (training, validation and test datasets).
    - Labelling errors – Data verified and corrected manually after data collection (but verification performed by operator who did the data collection).
  - **Format and Resolution** – Data captured using the Jetbot camera, i.e. the same that will be used in operation.
  - **Digital error protection** – Hash computed for all .jpg, .csv and .zip files generated during data collection. Integrity of files verified before use.
  - **Traceability** – Excel table generated during data collection and augmentation and containing the following information:
    - Image file name + date and time of collection,
    - Operating parameters (e.g. lighting condition, position on grid pattern…),
    - Hash,
    - Transformations applied during data augmentation.

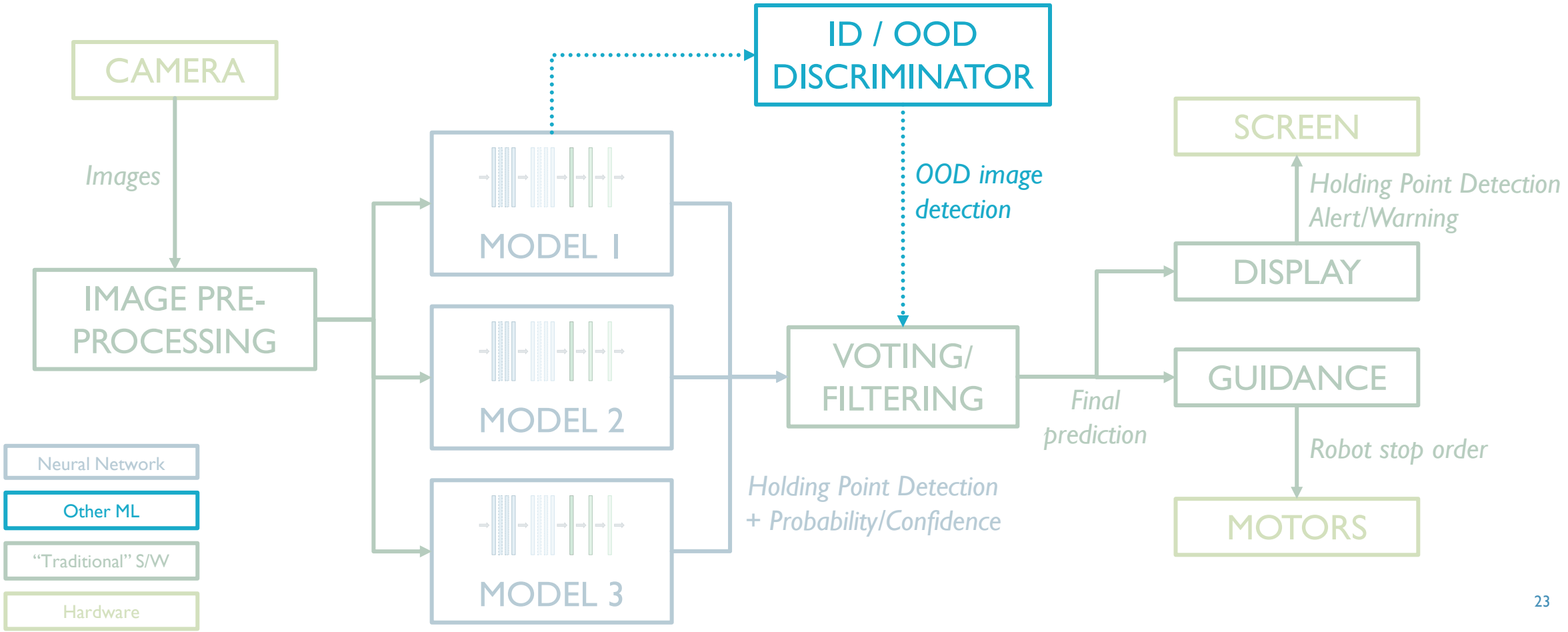
DATA MANAGEMENT
DATASET COMPLETENESS – 1/5

- **CoDANN Completeness Requirement**
  - For a given dataset to be "complete", "most elements in the Input Space should be close to a datapoint".
  - But even when considering numerous independently distributed operating parameters, "**the operating parameters will invariably fail to describe subtleties from the Input Space**".
  - To circumvent this issue, the CoDANN recommends to:
    - Define an **input distribution discriminator**,
    - Define an out-of-distribution dataset,
    - Demonstrate that the Discriminator is close to 1 (one) for the training, validation and test datasets and close to 0 for the out-of-distribution dataset.

DATA MANAGEMENT
DATASET COMPLETENESS – 2/5

- **Out-Of-Distribution Detection techniques** – There are several families of techniques [DM1]:
  - **Apply a threshold to the prediction confidence –** Maximum Softmax probability [DM2] and derived techniques such as ODIN [DM3].
  - **Train two separate networks** – One that performs the classification/regression task and one that performs the OOD Detection task.
  - **Train a single network to perform both tasks at the same time** (i.e. classification/regression and OOD Detection).
  - **Train a separate SVM model** to perform the OOD Detection.
- Some techniques require a **dedicated OOD training dataset** (e.g. ODIN) whereas others do not require any OOD training dataset (e.g. Generalized ODIN [DM4], Early Layer SVM [DM5]), only samples for testing.
- **Selected solution – Early Layer SVM:**
  - No OOD training dataset required, only some OOD samples for performance evaluation,
  - No impact on network architecture.

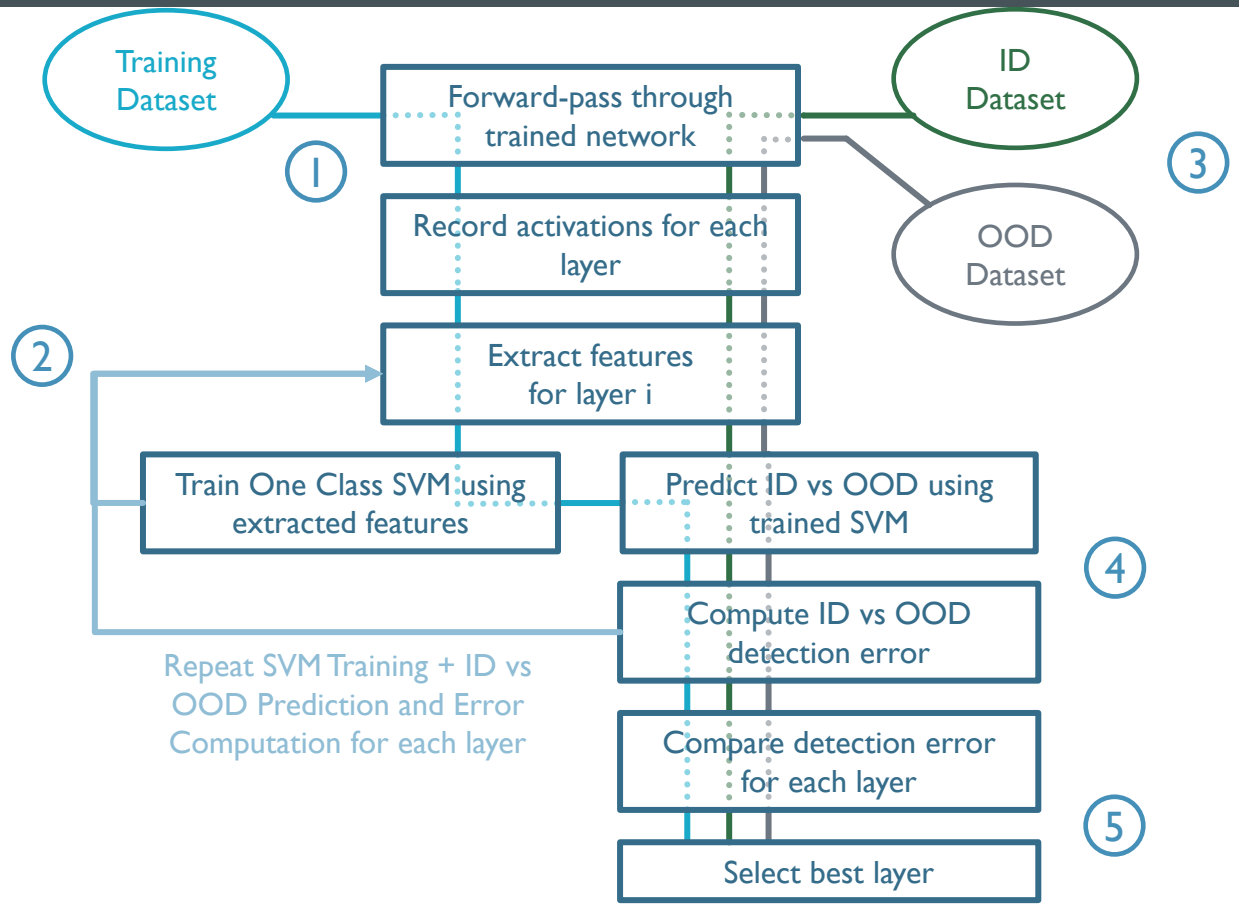
DATA MANAGEMENT
DATASET COMPLETENESS – 3/5



DATA MANAGEMENT
DATASET COMPLETENESS – 4/5

- **Early Layer Selection Algorithm**

1. Forward pass of the training dataset through the trained network and recording of the activations for each layer.
2. Training of a One Class SVM using the extracted features (as many SVM trained as there are layers in the network).
3. Feed the test ID and test OOD datasets through the network and record the activations.
4. Test the trained SVM using the generated activations.
5. Select the SVM and corresponding layer that performs the best.



DATA MANAGEMENT
DATASET COMPLETENESS – 5/5

- Early Layer Selection Algorithm applied to model 1 (see next sections) using:
  - For the **ID Samples Dataset** – Pictures extracted from the training dataset (figures on the left in the table) and from the test dataset (figures on the right).
  - For the **OOD Samples Dataset** – Pictures taken by the robot outside of its simplified environment + other pictures from other datasets (birds, vehicles…) found on Kaggle.
- Overall, the best candidate layer appears to be the **Conv2 Layer**:

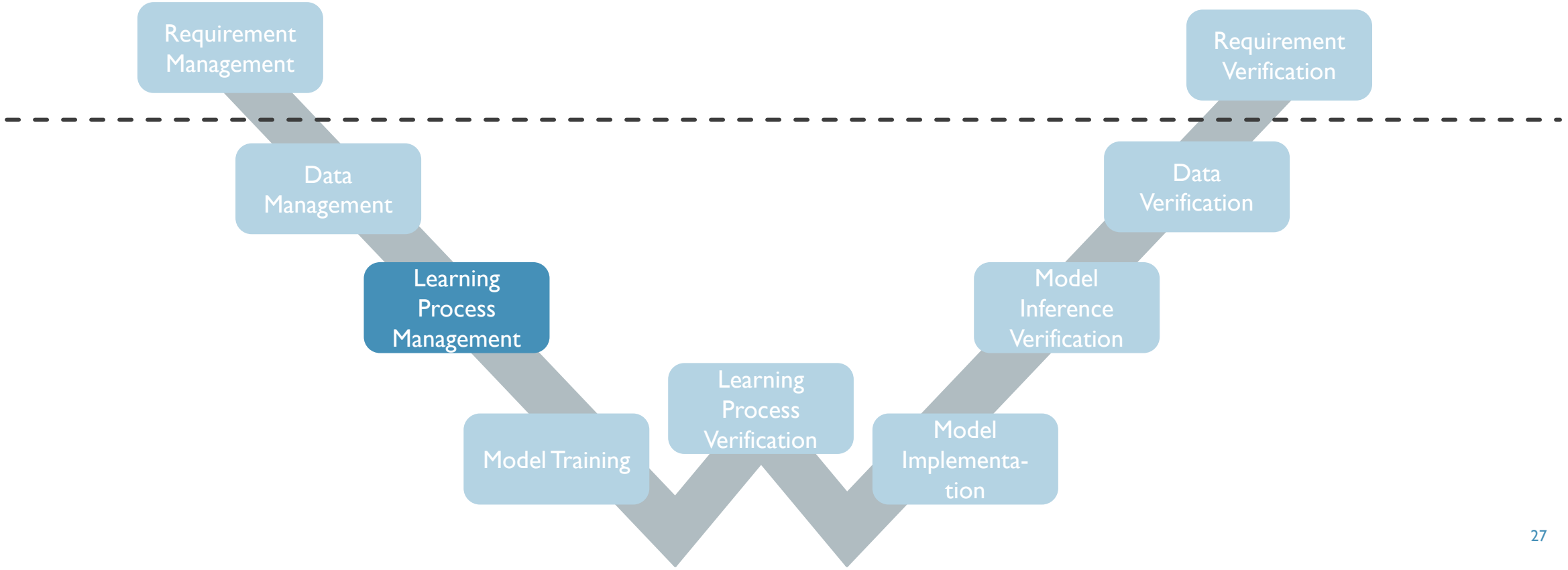
LAYER	ID ERROR	OOD ERROR	OVERALL
Conv 1	0.01 / 0.08	0.22 / 0.22	0.12 / 0.15
BatchNorm 1	0.01 / 0.07	0.26 / 0.26	0.13 / 0.17
Conv 2	**0.02 / 0.18**	**0.04 / 0.04**	**0.03 / 0.11**
BatchNorm 2	0.02 / 0.14	0.16 / 0.16	0.09 / 0.15
FC 1	0.99 / 1.0	0.0 / 0.0	0.5 / 0.5
FC 2	0.97 / 1.0	0.0 / 0.0	0.48 / 0.5

LEARNING PROCESS MANAGEMENT
OUTLINE

- Project Overview
- Data Management
- Learning Process Management
  - CoDANN Requirements
  - Application to use case scenario
- Model Training
- Learning Process Verification
- Model Implementation
- Inference Model Verification
- Conclusion
- Annex

LEARNING PROCESS MANAGEMENT
CODANN REQUIREMENTS – 1/2

W-shaped Leaning Assurance Process



LEARNING PROCESS MANAGEMENT
CODANN REQUIREMENTS – 2/2

- **Training Environment**
  - Define/record training environment configuration.
- **Validation & Verification metrics**
  - Select metrics to be used for the validation and verification phases.
- **Training algorithm definition**
  - Select key elements of the training algorithm:
    - Activation function,
    - Loss function,
    - Initialization strategy,
    - Training hyperparameters.

LEARNING PROCESS MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 1/5

- **Training Environment**
  - Training performed in COLAB PRO environment:
    - Python version – 3.7.10
    - PyTorch version – 1.8.1+cu101
    - Torchvision version – 0.9.1+cu101
    - GPU – Tesla P100-PCIE-16GB
    - CUDA – Version 11.2
    - CPU – Intel(R) Xeon(R) CPU @ 2.30GHz

LEARNING PROCESS MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 2/5

- **Validation & Verification metrics**
  - Main metrics for model selection:
    - **Training Loss,**
    - **Validation Loss,**
    - **Accuracy**.
  - Additional metrics:
    - True Positive (TP) / True Negative (TN) / False Positive (FP) / False Negative (FN),
    - True Positive Rate (TPR) / Sensitivity,
    - True Negative Rate (TNR) / Specificity,
    - False Positive Rate (FPR) = Nuisance Alerts,
    - False Negative Rate (FNR) = Safety Hazard.
  - Metrics recorded and visualized using **TensorBoard.**

		PREDICTED	
		HP	**No HP**
ACTUAL	**HP**	TP	FN
	No HP	FP	TN

Accuracy	(TP+TN)/(TP+TN+FP+FN)
TPR (Sensitivity)	TP/(TP+FN)
TNR (Specificity)	TN/(TN+FP)
FPR	FP/(FP+TN)
FNR	FN/(FN+TP)

LEARNING PROCESS MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 3/5

- **Training algorithm definition**
  - **Activation function – ReLu**

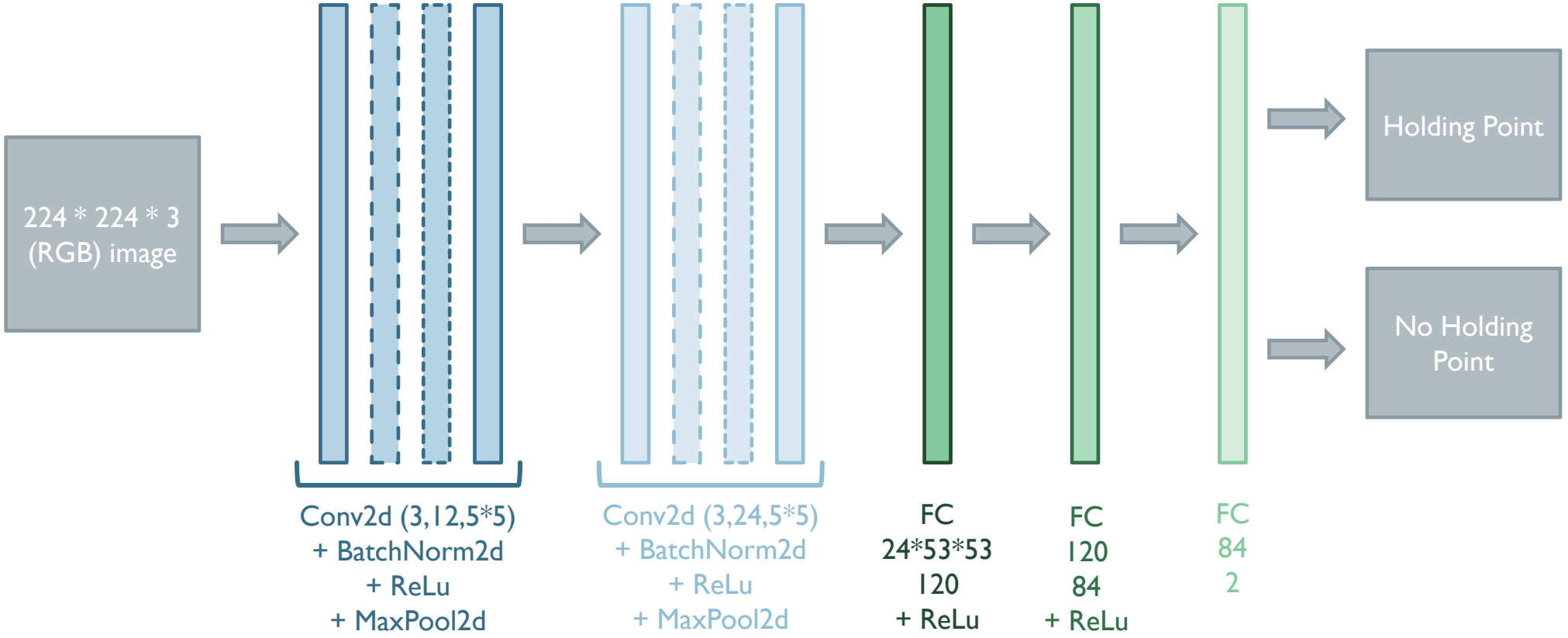
    Rationale – Commonly used activation function + Less susceptible to vanishing gradient issue
  - **Loss function – Adam Optimizer**

    Rationale – Converges rapidly + Widely used optimizer
  - **Initialization strategy**
    - Exploratory phase (e.g. hyperparameters selection…): Uniform (bounded by 1/sqrt(in_features) = default PyTorch initialization),
    - Final models training – Different initialization strategy for each model: Uniform (bounded by 1/sqrt(in_features)) / Xavier Uniform / Xavier Normal

    Rationale – If several models are trained and used with a voting system, the CoDANN requires that different initialization strategy (+ different datasets) be used for the building of the different models.

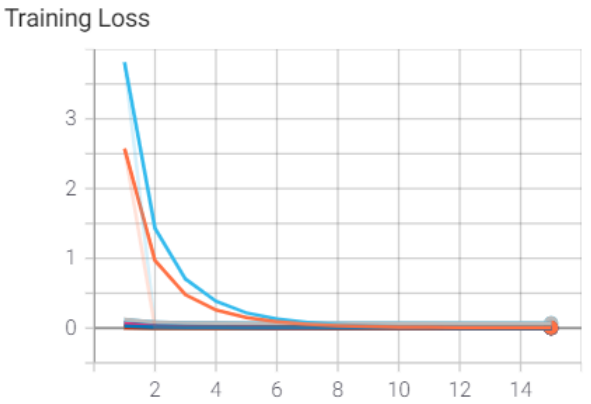
LEARNING PROCESS MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 4/5

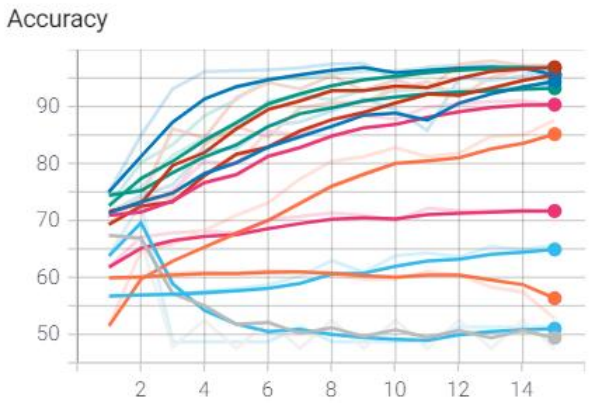
Network Architecture



LEARNING PROCESS MANAGEMENT
APPLICATION TO USE CASE SCENARIO – 5/5

- **Training algorithm definition (continued)**
  - **Training hyperparameters** – Compared the performance of various combinations of hyperparameters:
    - Batch Size = 10 / 100 / 500
    - Learning Rate = 0.1 / 0.01 / 0.001
    - Scheduler Gamma 0.1 / 1 (with Scheduler Step Size = 5)
  - Based on Training and Validation Loss and Accuracy metrics, the following hyperparameters were selected:
    - Batch Size = 100
    - Learning Rate = 0.001
    - Scheduler Gamma = 1



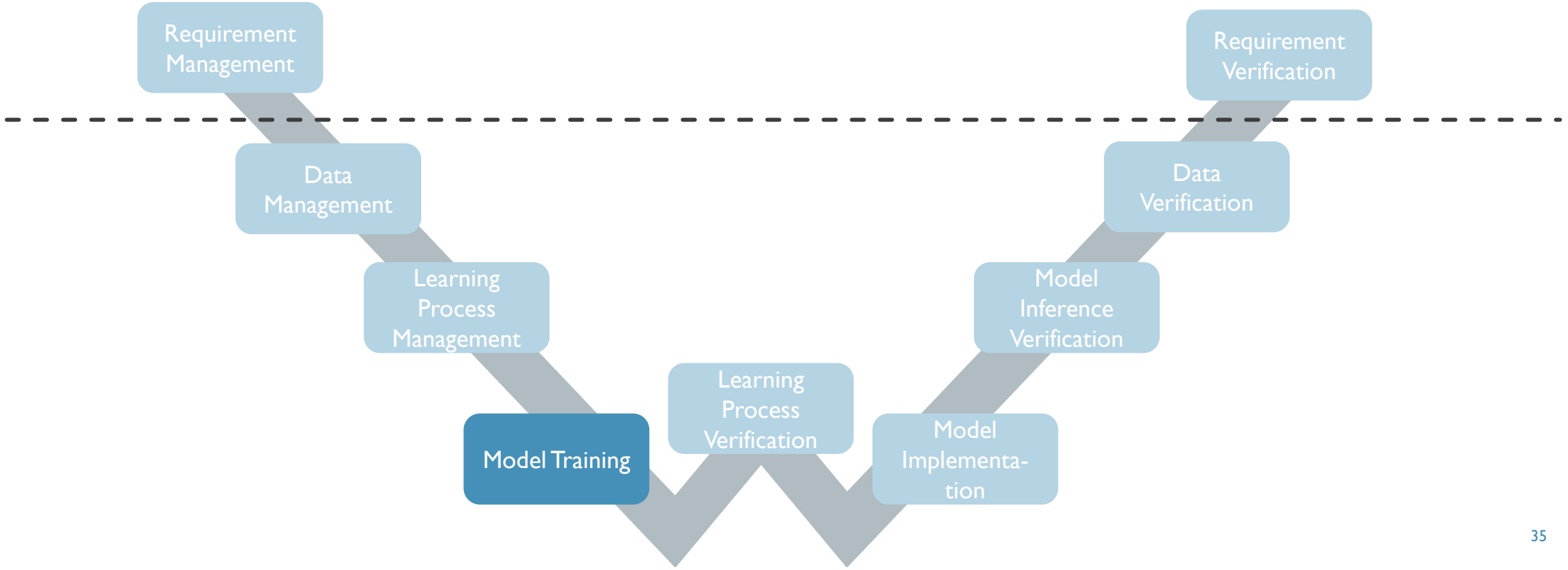


MODEL TRAINING
OUTLINE

- Project Overview
- Data Management
- Learning Process Management
- Model Training
  - CoDANN Requirements
  - Application to use case scenario
  - Training Algorithm Stability
- Learning Process Verification
- Model Implementation
- Inference Model Verification
- Conclusion
- Annex

MODEL TRAINING
CODANN REQUIREMENTS – 1/2

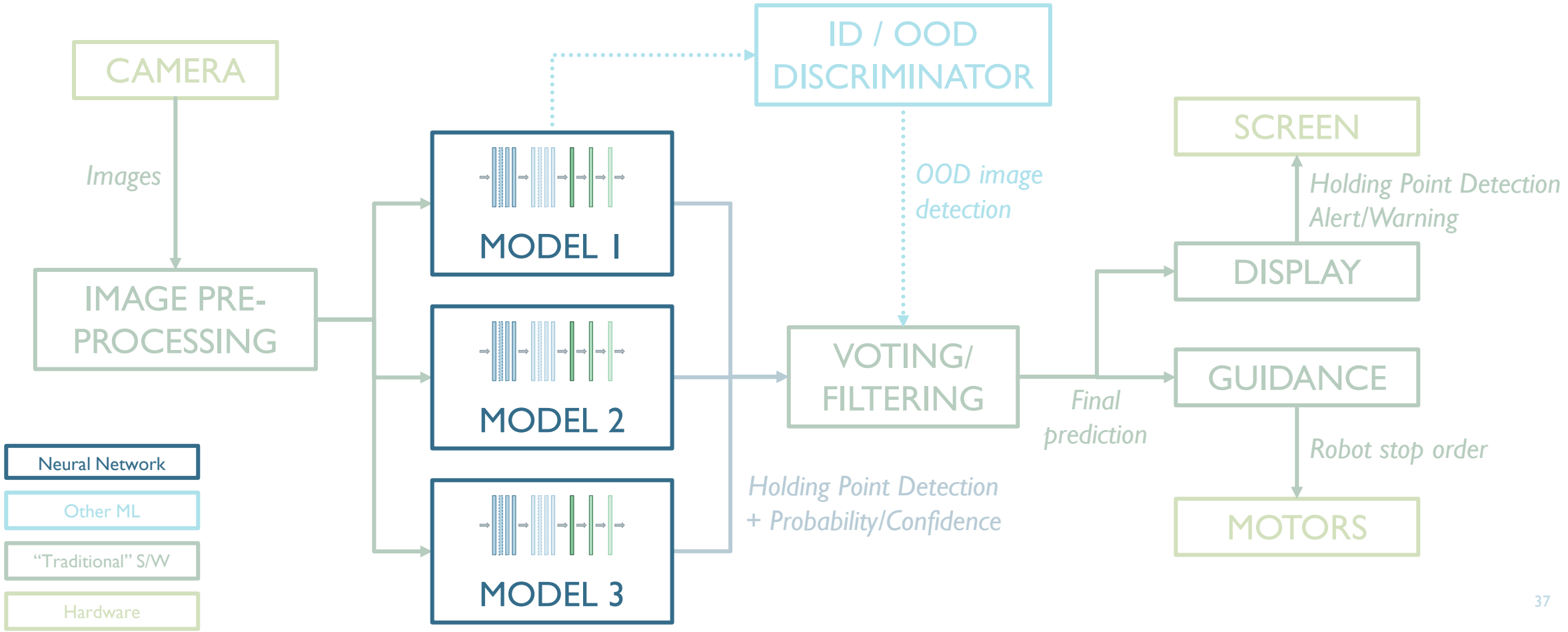
W-shaped Leaning Assurance Process



MODEL TRAINING
CODANN REQUIREMENTS – 2/2

- **Model Training**
  - Execute the training algorithm in the conditions defined previously, using the training dataset.
- **Model validation**
  - Validate the performance of the model using the validation dataset.
- **Training phase verification**
  - Convergence - Demonstrate adequate converge using training curves.
  - Reproducibility and replicability - Demonstrate training algorithm stability.

MODEL TRAINING
APPLICATION TO USE CASE SCENARIO – 1/4

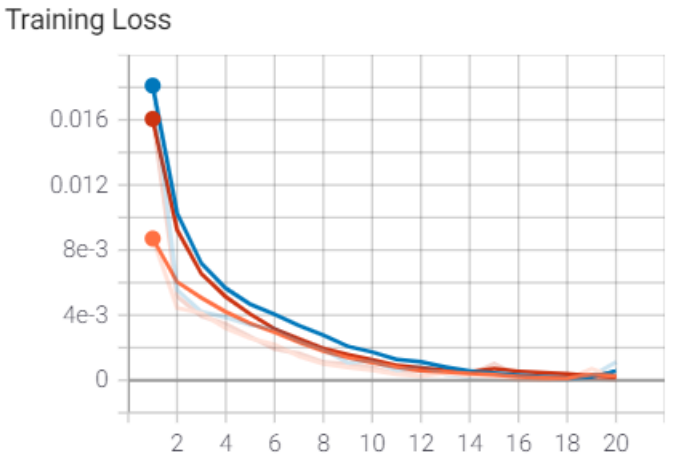


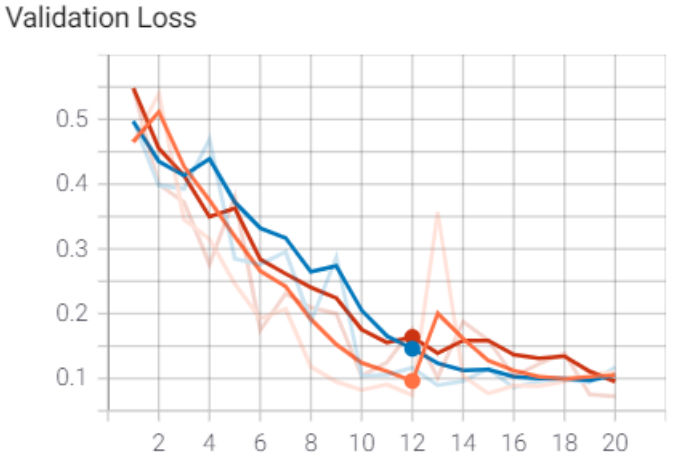
MODEL TRAINING
APPLICATION TO USE CASE SCENARIO – 2/4

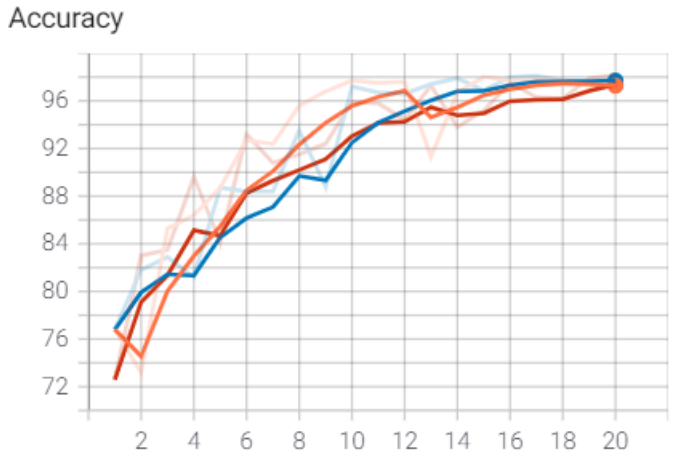
- **Model Training**
  - 3 different models trained using the selected hyperparameters:
    - Batch Size = 100
    - Learning Rate = 0.001
    - Scheduler Gamma = 1
  - Each model was trained:
    - Using a different dataset = Same original data but different augmentation strategies (seed, order of application of the transformations…).
    - Using different weight initialization techniques:
      - Model 1 – Uniform (bounded by 1/sqrt(in_features))
      - Model 2 – Xavier Uniform
      - Model 3 – Xavier Normal

MODEL TRAINING
APPLICATION TO USE CASE SCENARIO – 3/4

- **Model validation**
  - Model performance assessed using:
    - Training Loss,
    - Validation Loss,
    - Accuracy.
- **Training phase verification**
  - Convergence – See Training Curves.
  - Reproducibility and replicability – To be addressed separately.







MODEL TRAINING
APPLICATION TO USE CASE SCENARIO – 4/4

- **Model validation** – Best models additional metrics:

Model 1		PREDICTED	
		HP	No HP
ACTUAL	HP	502	3
	No HP	2	493

Accuracy	99.5%
TPR (Sensitivity)	0.99
TNR (Specificity)	1.00
FPR	0.00
FNR	0.01

Model 2		PREDICTED	
		HP	No HP
ACTUAL	HP	481	25
	No HP	8	486

Accuracy	96.7%
TPR (Sensitivity)	0.95
TNR (Specificity)	0.98
FPR	0.02
FNR	0.05

Model 3		PREDICTED	
		HP	No HP
ACTUAL	HP	483	34
	No HP	30	453

Accuracy	93.6%
TPR (Sensitivity)	0.93
TNR (Specificity)	0.94
FPR	0.06
FNR	0.07

Note – The above accuracy and FNR measurements would not be satisfactory for a safety-critical application such as a holding point detection system but the focus of this project being the application of the design assurance requirements these models will be used for the next phases of the project.

MODEL TRAINING
TRAINING ALGORITHM STABILITY – 1/4

- **Training phase verification** – Reproducibility and replicability
  - Algorithm Robustness vs Model Robustness:
    - Algorithm robustness – To demonstrate the **stability of the training algorithm** means ensuring "that the produced model does not change a lot under perturbations of the training dataset".
    - Model robustness – "Keeping input-output relations of a trained model".
  - In this section, the CoDANN requests a demonstration of the robustness of the Training Algorithm, i.e. that the selected training algorithm produces similar models when the training datasets differ slightly.
  - The stability of the training algorithm can be assessed by comparing:
    - Either the **outputs** (i.e. classification results),
    - or the **activations** of the various layers (techniques include Canonical Correlation Analysis (CCA) [MT1], Centered Kernel Alignment (CKA) [MT2]…)

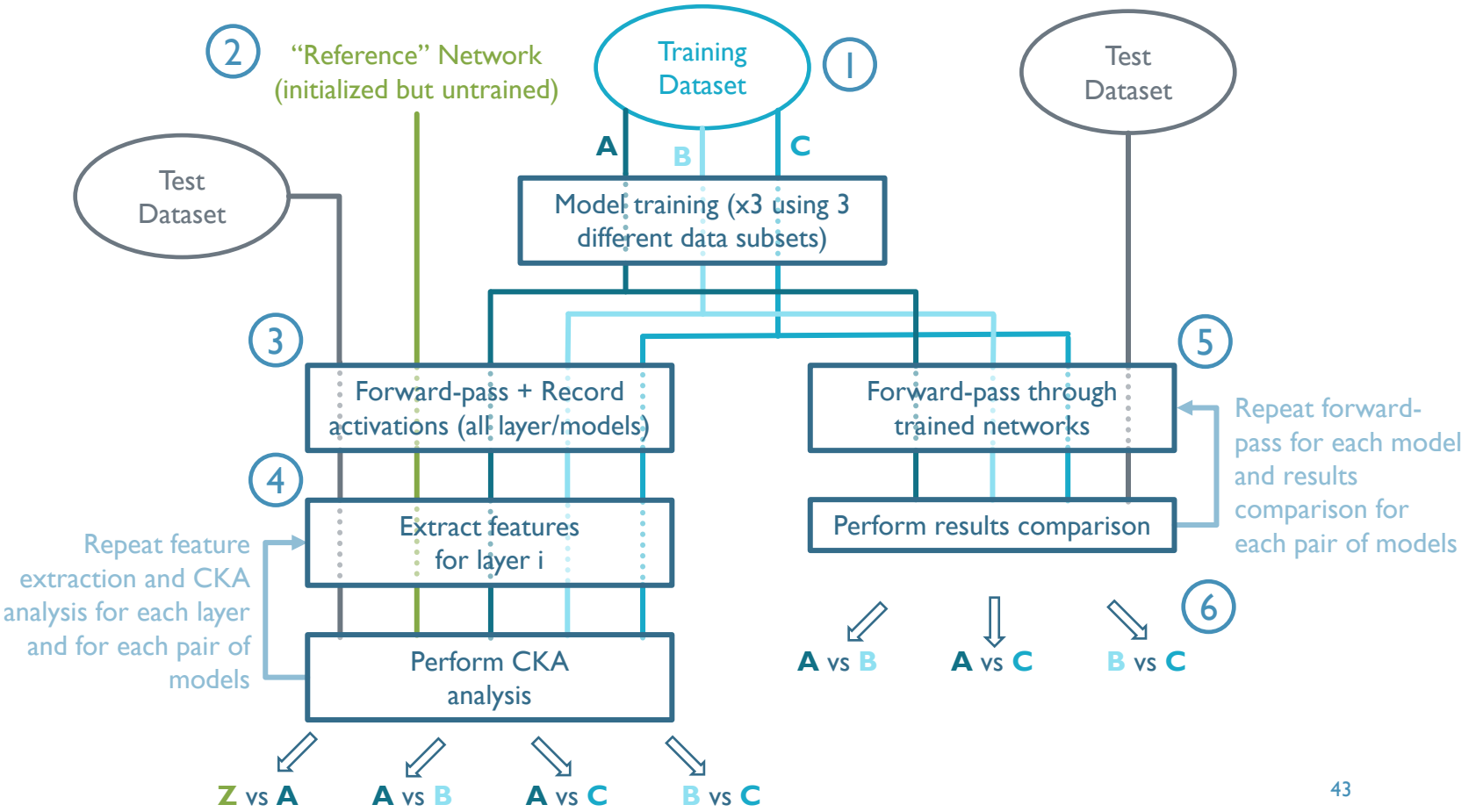
  of multiple networks (trained using slightly different training datasets) when fed with the same test data.

MODEL TRAINING
TRAINING ALGORITHM STABILITY – 2/4

- **Selected solution**
  - 3 new models (A, B and C) trained using:
    - 3 slightly different subsets of Dataset "Training 1" (i.e. the dataset used to train Model 1),
    - Uniform (bounded by 1/sqrt(in_features)) Initialization,
    - Training strategy (hyperparameters…) defined previously.
  - Models compared in terms of:
    - Centered Kernel Analysis of the activations,
    - Output.
  - See stability verification algorithm next slide.

MODEL TRAINING
TRAINING ALGORITHM STABILITY – 3/4

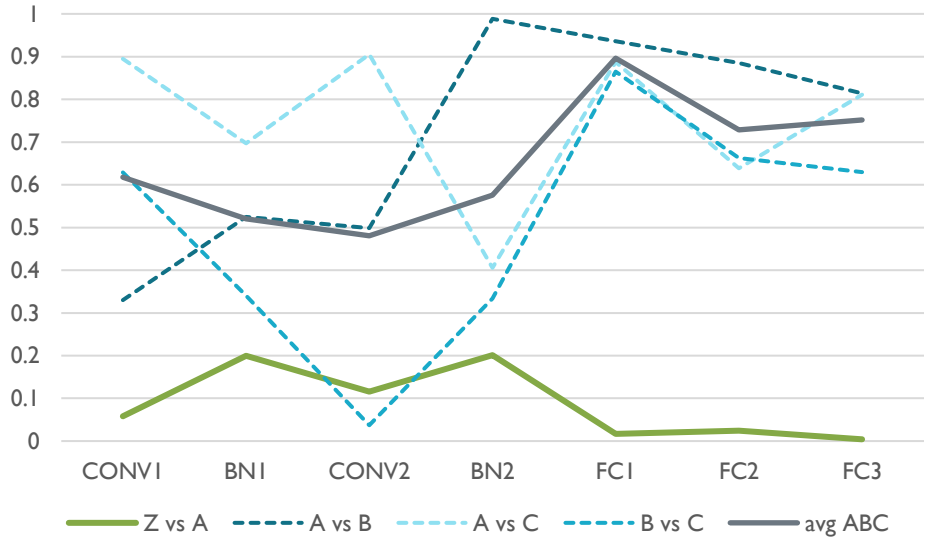
1. Train 3 different models using the same algorithm but different subsets of the training dataset (A/B/C).
2. Prepare an untrained network (Z).
3. Perform a forward pass of the test dataset through all 4 networks and record the activations for each layer.
4. For each layer, retrieve the activations for all 4 networks and perform the CKA Analysis.
5. Perform a forward pass of the test dataset through all 3 trained networks and record the outputs.
6. Compare the outputs of the 3 trained networks.



MODEL TRAINING
TRAINING ALGORITHM STABILITY – 4/4

- **Outcome of CKA Analysis and Result Comparison**
  - Overall, the 3 trained models are closer to each other than to the untrained model when comparing their activations layer-by-layer.
  - They also produce similar results (~83% similar).

LAYER	Z vs A	A vs B	A vs C	B vs C
CKA - Conv 1	0.06	0.033	0.89	0.63
CKA - BatchNorm 1	0.20	0.52	0.69	0.34
CKA - Conv 2	0.12	0.50	0.90	0.04
CKA - BatchNorm 2	0.20	0.98	0.40	0.33
CKA - FC 1	0.02	0.93	0.89	0.86
CKA - FC 2	0.02	0.88	0.64	0.66
CKA - FC3	0.00	0.81	0.81	0.63
Results	N.A.	85%	83%	82%



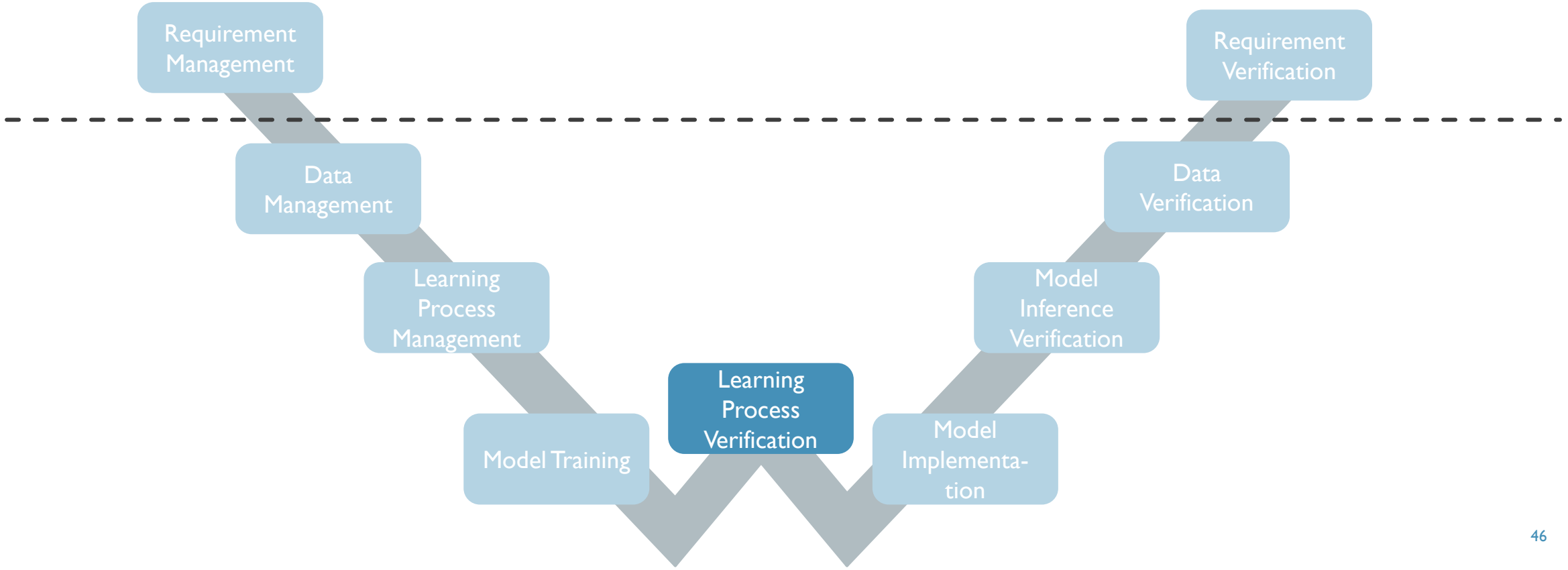
Note – Even though no minimum acceptable index of similarity is defined in the CoDANN (it would depend on the type of application and outcome of the safety assessment), the index of similarity demonstrated here would probably not be satisfactory to comply with the design assurance requirements.

LEARNING PROCESS VERIFICATION
OUTLINE

- Project Overview
- Data Management
- Learning Process Management
- Model Training
- Learning Process Verification
  - CoDANN Requirements
  - Application to use case scenario
  - Model Properties Verification
- Model Implementation
- Inference Model Verification
- Conclusion
- Annex

DATA MANAGEMENT
CODANN REQUIREMENTS – 1/2

W-shaped Leaning Assurance Process



LEARNING PROCESS VERIFICATION
CODANN REQUIREMENTS – 2/2

- **Model verification**
  - Validate the performance of the models using the test dataset.
  - Verify the properties of the models.

LEARNING PROCESS VERIFICATION
APPLICATION TO USE CASE SCENARIO – 1/3

- Performance metrics of the 3 trained models using the test dataset:

Model 1		PREDICTED	
		HP	No HP
ACTUAL	HP	288	66
	No HP	74	291

Accuracy	80.5%
TPR (Sensitivity)	0.81
TNR (Specificity)	0.80
FPR	0.20
FNR	0.19

Model 2		PREDICTED	
		HP	No HP
ACTUAL	HP	282	72
	No HP	82	283

Accuracy	78.6%
TPR (Sensitivity)	0.80
TNR (Specificity)	0.78
FPR	0.22
FNR	0.20

Model 3		PREDICTED	
		HP	No HP
ACTUAL	HP	234	120
	No HP	43	322

Accuracy	77.3%
TPR (Sensitivity)	0.66
TNR (Specificity)	0.88
FPR	0.12
FNR	0.34

LEARNING PROCESS VERIFICATION
APPLICATION TO USE CASE SCENARIO – 2/3

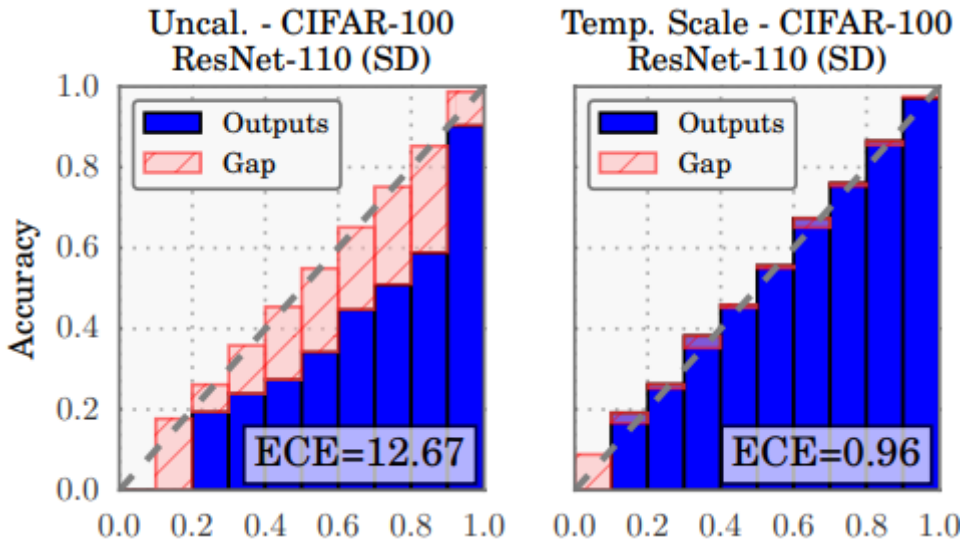
- We notice a significant drop in accuracy when processing the Test Dataset:
  - From upward of **95%** with the **Validation Dataset,**
  - To around **78%** with the **Test Dataset.**
- Accuracy can be improved by filtering out non-confident predictions, i.e. comparing:
  - SoftMax probability,
  - Or Calibrated SoftMax probability using Temperature Scaling [LPV1]

to a given threshold (if the probability is below the threshold, the prediction is excluded).

Temperature Scaling = Post-processing technique used to calibrate neural networks.

softmax = e^(z/T) / sum_i e^(z_i/T)

where T (Temperature Scaling coefficient) is a learned parameter.



source

LEARNING PROCESS VERIFICATION
APPLICATION TO USE CASE SCENARIO – 3/3

- Both techniques (SoftMax and SoftMax with Temperature Scaling) tested on the trained models.
- Optimal confidence threshold (excluding around 30% of datapoints or less for all models) found to be:
  - 0.97 for SoftMax probability,
  - 0.73 for SoftMax with Temperature Scaling probability.
- Both techniques allow to improve the accuracy of the predictions:

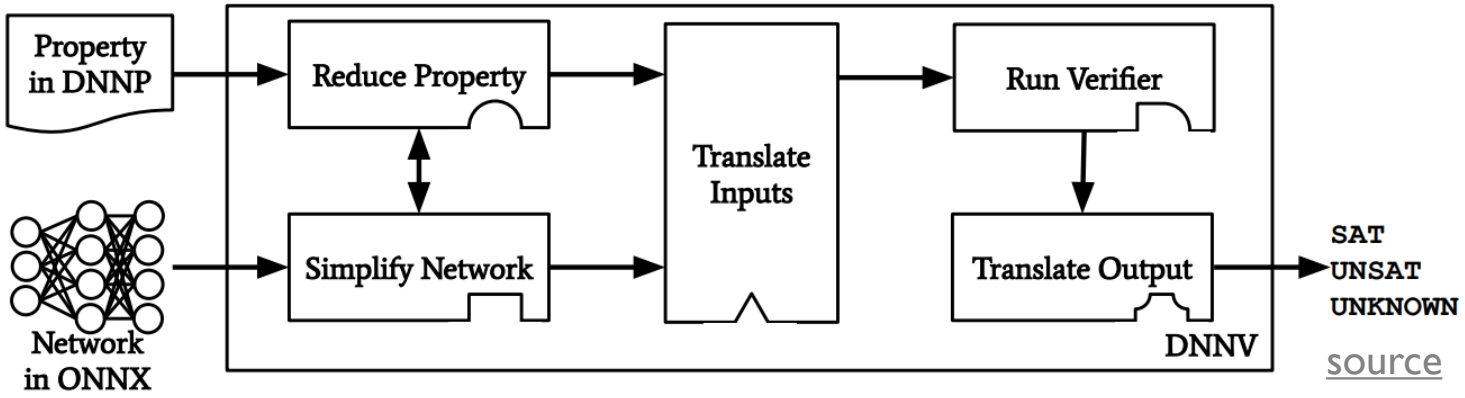
	Model 1	Model 2	Model 3
Raw accuracy	**80.5%**	**78.6%**	**77.3%**
Accuracy with 0.97 threshold	**86.2%**	**85.5%**	**81.7%**
Datapoints excluded with 0.97 threshold	*27.6%*	*24.1%*	*19.5%*
Accuracy with Temperature Scaling and 0.73 threshold	**86.3%**	**87.0%**	**82.0%**
Datapoints excluded with Temperature Scaling and 0.73 threshold	*28.6%*	*31.7%*	*21.0%*

LEARNING PROCESS VERIFICATION
MODEL PROPERTIES VERIFICATION – 1/3

- Numerous methods exist to verify the properties of neural network models:
  - **Coverage-based white-box testing** – Systematic testing of the outputs and internal parameters of the model.
  - **Falsification** – Adversial attacks aimed at generating corner-cases for the trained model.
  - **Formal verification** – Formal approach aiming at obtaining worst case robustness bounds.
- Main issues:
  - Computational cost,
  - Trade-off between runtime of a verification algorithm and the completeness of its results it returns,
  - Applicability limited to certain types of activation function or network architecture,
  - Infeasibility to exhaustively verify all possible computation branches.

LEARNING PROCESS VERIFICATION
MODEL PROPERTIES VERIFICATION – 2/3

- Additional implementation issue – Many methodologies are available but each methodology expects the model to be tested and the properties to be verified in a specific format.
- **Selected solution – DNNV Package** = "Framework" for Deep Neural Network verification [LPV2]
  - Various verification methodologies available (Reluplex, ERAN, Planet…).
  - Network to be tested – Single widely used input format (.onnx).
  - Property to be verified – Single easy-to-generate input format (domain-specific language DNNP).



source

LEARNING PROCESS VERIFICATION
MODEL PROPERTIES VERIFICATION – 3/3

- All 3 trained models verified using the Reluplex [LPV3] and ERAN [LPV4] methodologies.
- Property to be verified - Variations of input image (in-distribution image, with or without holding point) provide same output
- Results:

	Model 1	Model 2	Model 3
RELUPLEX – Picture with Holding Point	PASS	PASS	PASS
RELUPLEX – Picture with no Holding Point	PASS	PASS	PASS
ERAN – Picture with Holding Point	PASS	PASS	PASS
ERAN – Picture with no Holding Point	PASS	PASS	PASS

Note - Multiple variations of property tested (values of epsilon) – All checks always passed (even when true_class was incorrect… Reliability of the tool questionable).

Example of property to be verified in domain-specific language (DNNP):

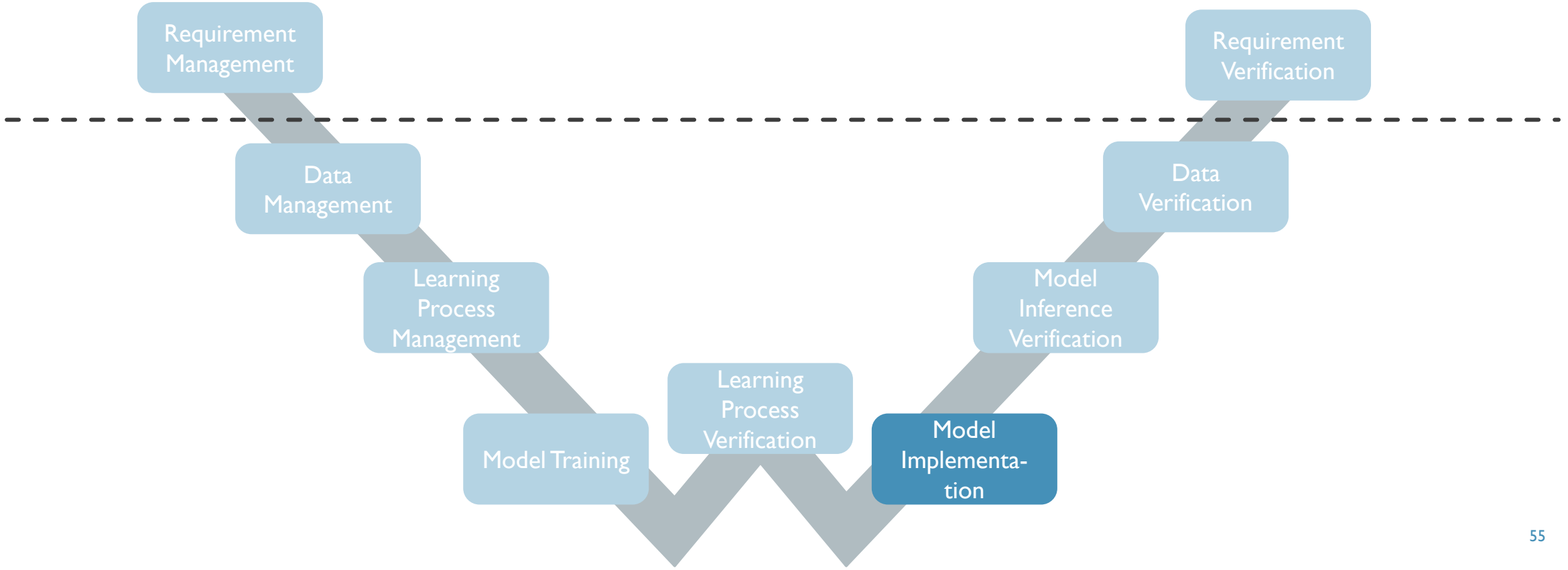
```
from dnnv.properties import *
import numpy as np

N = Network("N")
x =
Image("/content/drive/MyDrive/KASHIKO/DATASET/VERIF/holdingpoint.npy")

epsilon = 2.0/255
true_class = 0

Forall(
    x_,
    Implies(
        ((x - epsilon) < x_ < (x + epsilon))
& (0 < x_ < 1),
        argmax(N(x_)) == argmax(N(x)),
    ),
)
```

MODEL IMPLEMENTATION
OUTLINE

- Project Overview
- Data Management
- Learning Process Management
- Model Training
- Learning Process Verification
- Model Implementation
  - CoDANN Requirements
  - Application to use case scenario
  - Optimisation Techniques and Tools
- Inference Model Verification
- Conclusion
- Annex

MODEL IMPLEMENTATION
CODANN REQUIREMENTS – 1/2

W-shaped Leaning Assurance Process



MODEL IMPLEMENTATION
CODANN REQUIREMENTS – 2/2

- **Inference Hardware**
  - Identify inference hardware features that could affect model behaviour/performance.

- **Model Optimisation**
  - Identify any transformation/optimisation required to run the model on the inference hardware.
  - Assess the impact of these transformations on the model properties.
  - Record transformation/optimisation environnement configuration (i.e. software tools used…).

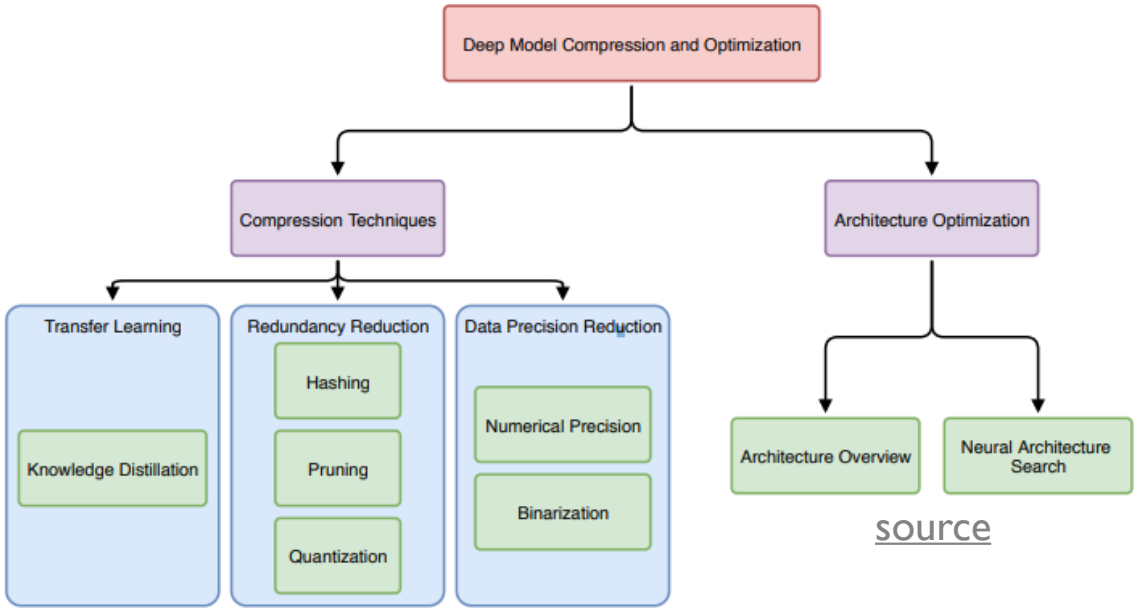
MODEL IMPLEMENTATION
APPLICATION TO USE CASE SCENARIO

- **Inference Hardware**
  - NVIDIA 128-Core Maxwell GPU
  - Quad-Core ARM Cortex A-57 Processor (1.43 GHz)
  - 4GB LPDDR4 Memory (25.6 GB/s)
- The inference hardware is greatly limited compared to the training environment (GPU - Tesla V100-SXM2-16GB = 5120 Cores).
- Optimisation of the models may be required to be able to run all 3 models in parallel on the inference hardware.

MODEL IMPLEMENTATION
OPTIMISATION TECHNIQUES AND TOOLS – 1/2

- There exists multiple compression and optimisation techniques [MI1] [MI2]:
  - Pruning and Quantisation are two techniques for which tools are readily available (e.g. NNI package, PyTorch built-in functionalities).
  - Quantisation can be performed post-training and may not require retraining.

    PyTorch Limitation – Optimisation with quantisation is only available for models running on CPU, not for models running on GPU.
  - Pruning requires retraining of the network.
- **Selected solution – Pruning** (coefficient 0.7) with **retraining** using the NNI package (documentation).



source

MODEL IMPLEMENTATION
OPTIMISATION TECHNIQUES AND TOOLS – 2/2

- Performance of the models (averaged for all 3 models) before pruning, after pruning but before retraining and after pruning and retraining:

TRAINING DATASET	PRE-PRUNING	POST-PRUNING / PRE-RETRAINING	POST-PRUNING / POST-RETRAINING
Accuracy	97%	72%	99%
Time	35s	25s	25s

TEST DATASET	PRE-PRUNING	POST-PRUNING / PRE-RETRAINING	POST-PRUNING / POST-RETRAINING
Accuracy	79%	64%	78%
Time	9.5s	7s	7s

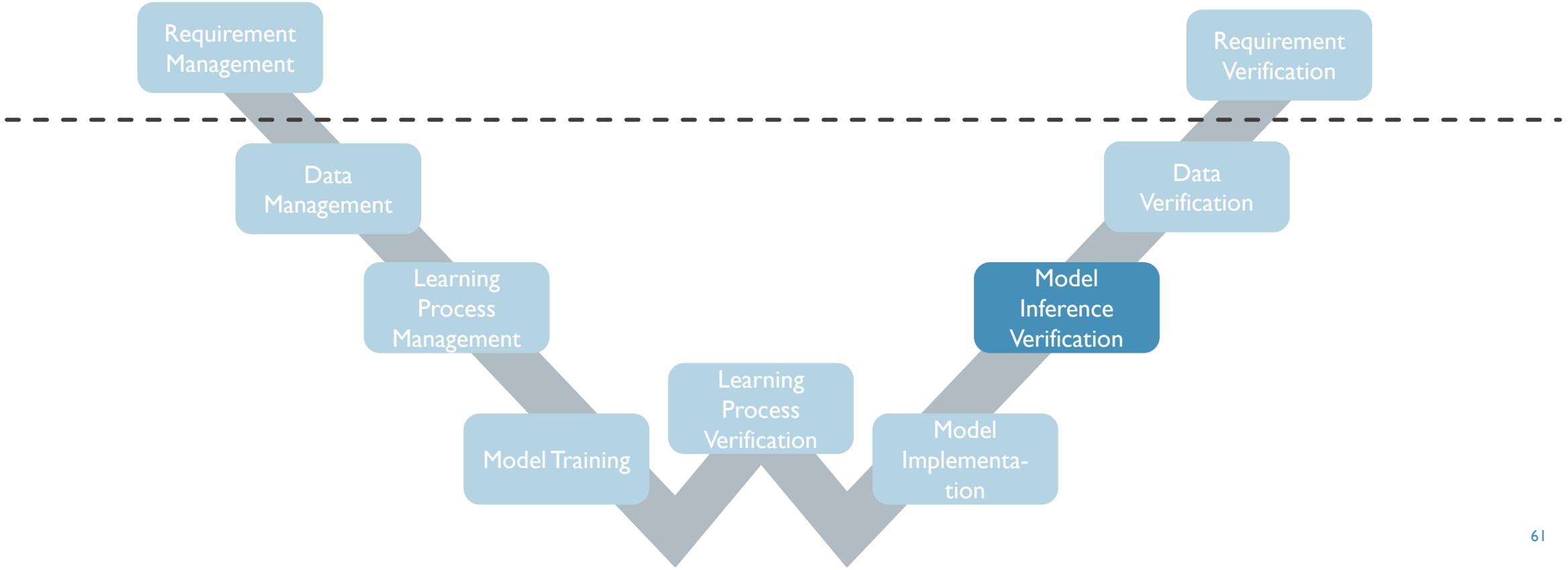
- Accuracy performance of pruned models after retraining equivalent to initial model.
- Slight improvement in execution time.
- Further improvement could be achieved by performing iterative pruning and retraining.

INFERENCE MODEL VERIFICATION
OUTLINE

- Project Overview
- Data Management
- Learning Process Management
- Model Training
- Learning Process Verification
- Model Implementation
- Inference Model Verification
  - CoDANN Requirements
  - Application to use case scenario
- Conclusion
- Annex

INFERENCE MODEL VERIFICATION
CODANN REQUIREMENTS – 1/2

W-shaped Leaning Assurance Process



INFERENCE MODEL VERIFICATION
CODANN REQUIREMENTS – 2/2

- **Inference Model Performance**
  - Evaluate the inference model using the test dataset.

- **Model properties preservation**
  - Verify (through implementation analysis or formal method) that the model properties have been preserved.

INFERENCE MODEL VERIFICATION
APPLICATION TO USE CASE SCENARIO – 1/3

- **Inference Model Performance**
  - Forward Pass of test dataset on pruned models on inference hardware – Individual model performance:

Model 1		PREDICTED	
		HP	No HP
ACTUAL	HP	277	77
	No HP	70	295

Raw Accuracy	79.5%
TPR (Sensitivity)	0.78
TNR (Specificity)	0.81
FPR	0.19
FNR	0.22

Model 2		PREDICTED	
		HP	No HP
ACTUAL	HP	280	74
	No HP	85	280

Raw Accuracy	77.9%
TPR (Sensitivity)	0.79
TNR (Specificity)	0.77
FPR	0.23
FNR	0.21

Model 3		PREDICTED	
		HP	No HP
ACTUAL	HP	236	118
	No HP	45	320

Raw Accuracy	77.3%
TPR (Sensitivity)	0.67
TNR (Specificity)	0.88
FPR	0.12
FNR	0.33

INFERENCE MODEL VERIFICATION
APPLICATION TO USE CASE SCENARIO – 2/3

- **Inference Model Performance (continued)**
  - Forward Pass of Test Dataset on pruned models on inference hardware – Overall Performance (including Confidence Filtering + Voting):

Confidence Filtering	Model 1	Model 2	Model 3
Raw accuracy	**79.5%**	**77.9%**	**77.3%**
Accuracy with 0.97 threshold	**85.5%**	**88.4%**	**85.9%**
Datapoints excluded with 0.97 threshold	*19.6%*	*29.4%*	*27.3%*

Note – The above accuracy and FNR measurements would not be satisfactory for a safety-critical application such as a holding point detection system but the focus of this project being the application of the design assurance requirements these models will be used for the next phases of the project.

Final Prediction		PREDICTED	
		HP	No HP
ACTUAL	HP	253	28
	No HP	20	269

Accuracy	**91.6%**
Datapoints Excluded	*20.7%*
TPR (Sensitivity)	**0.90**
TNR (Specificity)	**0.93**
FPR	**0.07**
FNR	**0.10**

INFERENCE MODEL VERIFICATION
APPLICATION TO USE CASE SCENARIO – 3/3

- **Model properties preservation**
  - Pruned models analysed using DNNV:

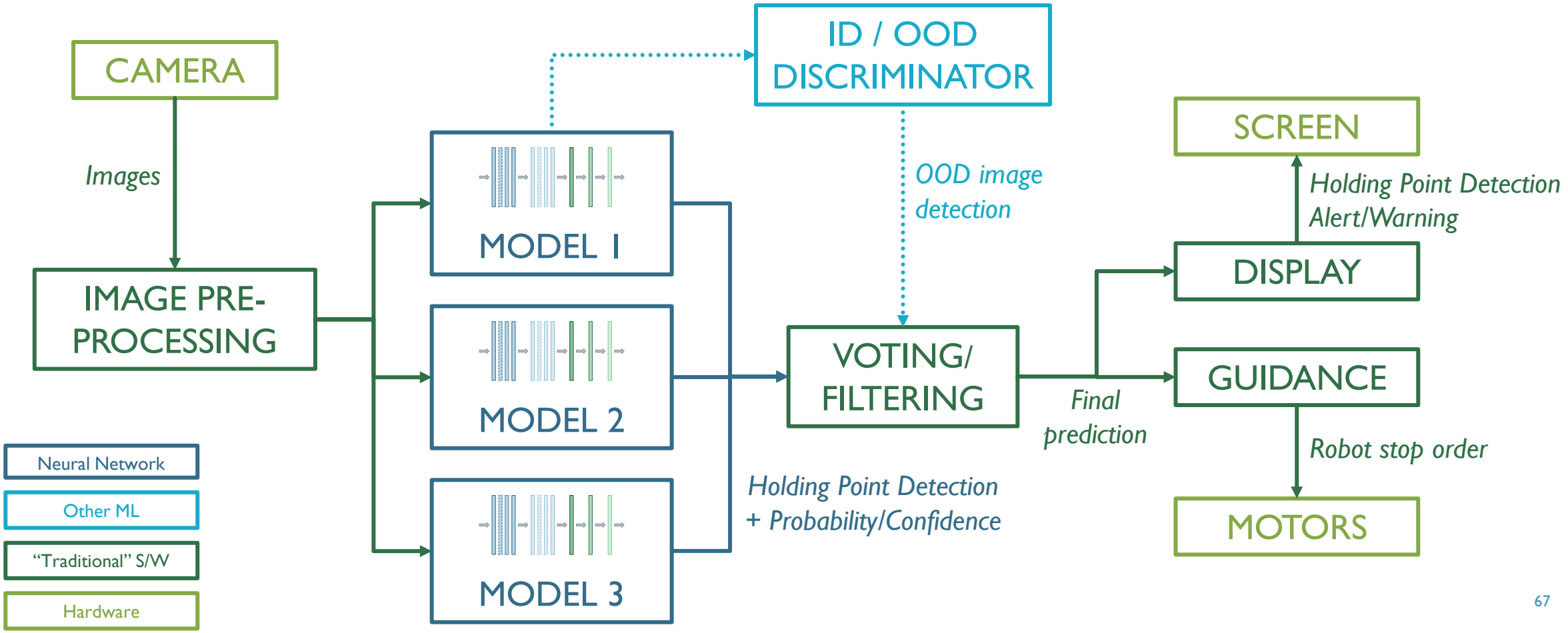
	Model 1	Model 2	Model 3
RELUPLEX – Picture with Holding Point	PASS	PASS	PASS
RELUPLEX – Picture with no Holding Point	PASS	PASS	PASS
ERAN – Picture with Holding Point	PASS	PASS	PASS
ERAN – Picture with no Holding Point	PASS	PASS	PASS

Note – As stated in the Learning Process Verification section, the reliability of the tool is questionable.

CONCLUSION
OUTLINE

- Project Overview
- Data Management
- Learning Process Management
- Model Training
- Learning Process Verification
- Model Implementation
- Inference Model Verification
- Conclusion
  - End-to-end Solution
  - Project Objectives Review
  - Discussion
- Annex

CONCLUSION
END-TO-END SOLUTION – 1/5



CONCLUSION
END-TO-END SOLUTION – 2/5

- **Image pre-processing**
  - Image pre-processing adapted to process camera's direct live feed compared to images in dataset.
- **Model transfer to Jetbot**
  - Jetbot runs PyTorch 1.3.0 whereas models training was performed using Pytroch 1.8.1.
  - Models saved using torch.save() function with setting `_use_new_zipfile_serialization=False` to ensure backward compatibility.
- **Discriminator**
  - Jetbot runs scikit-learn 0.19.1 whereas SVM model training was performed using scikit-learn 0.22.2.
  - SVM model trained using a more recent version of scikit-learn cannot be used by a previous version of scikit-learn.
  - SVM model had to be retrained locally in the Jetbot's environment.

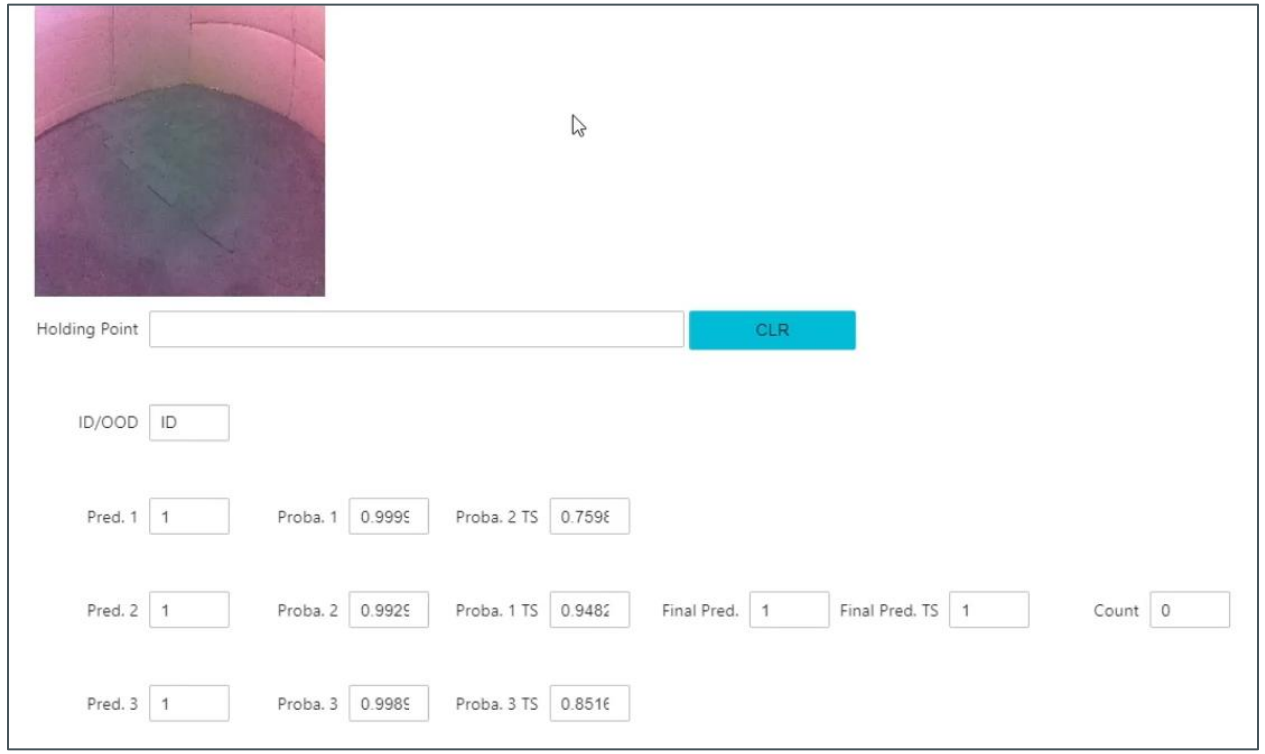
CONCLUSION
END-TO-END SOLUTION – 3/5

- **Voting/Filtering**
  - **Objective** – To improve the accuracy of the final prediction.
  - **Filtering logics** – SoftMax probabilities lower than 0.97 are not taken into account by the voting function.

    Note – SoftMax with Temperature Scaling probabilities are computed and assessed in the end-to-end solution as an experimental feature but are not used for the final predictions.
  - **Voting logics**
    - If all 3 models are "confident" (i.e. SoftMax probabilities > 0.97), two-out-of-three voting logics applied to compute the final prediction.
    - If only 2 models are "confident", both models must agree for a final prediction to be computed. If the models disagree, no final prediction is computed (set to -1).
    - If less than 2 models are "confident", no final prediction is computed (set to -1).

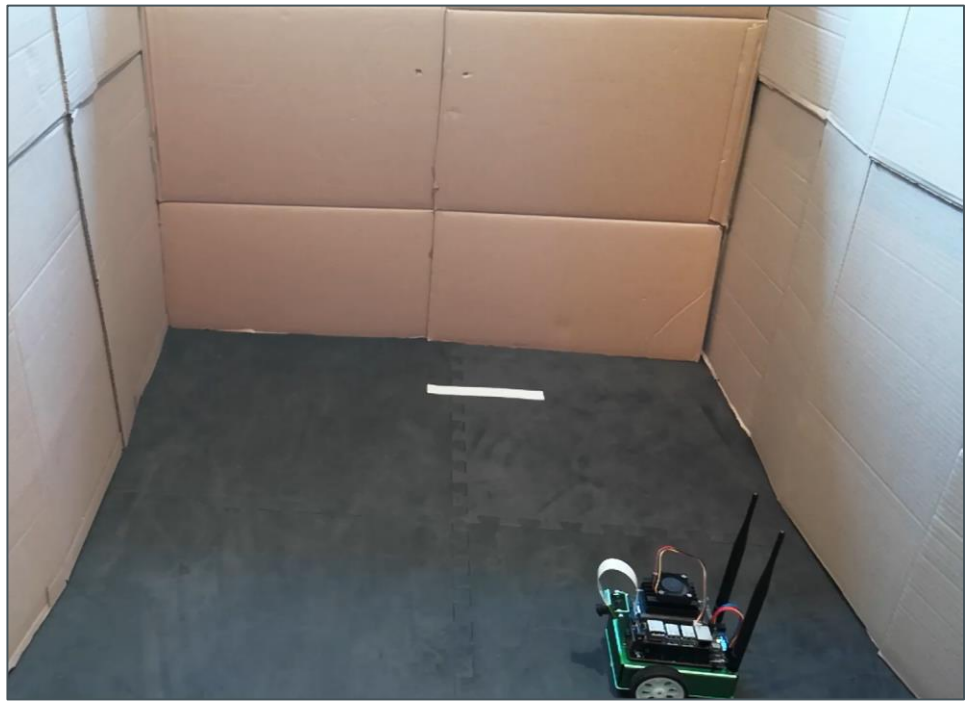
CONCLUSION
END-TO-END SOLUTION – 4/5

- **Demo – Out-Of-Distribution Detection**



CONCLUSION
END-TO-END SOLUTION – 5/5

- **Demo – Holding Point Detection + Warning**



CONCLUSION
PROJECT OBJECTIVES REVIEW – 1/2

TO "MIMIC" THE DESIGN, DEVELOPMENT AND VERIFICATION PHASES OF A SAFETY-CRITICAL EMBEDDED MACHINE LEARNING SOFTWARE BASED ON THE PRELIMINARY GUIDANCE MATERIAL ISSUED BY THE EUROPEAN CIVIL AVIATION AUTHORITIES (CODANN).

1. To review the **design assurance requirements** for AI-based embedded safety-critical applications

2. To explore the various **techniques** and **tools** available to satisfy these design assurance requirements.

3. To **implement** some of these techniques on a **simple use case scenario**.

CONCLUSION
PROJECT OBJECTIVES REVIEW – 2/2

- **Achievements**
  - 3 CNNs developed and trained on a simple use case scenario (to detect the presence of a holding point).
  - CoDANN design assurance recommendations applied during CNN design, development and verification phases.
  - End-to-end solution (including real-time image processing, OOD detection, warning display…) implemented on Jetbot.
  - Overall accuracy of the end-to-end solution > 90%.
  - In addition to CNN development, various techniques implemented in end-to-end solution:

REQUIREMENTS	TECHNIQUES IMPLEMENTED ON USE CASE SCENARIO
Out-Of-Distribution Detection	Early Layer SVM
Training Algorithm Stability	CKA Analysis
Model Calibration	Temperature Scaling
Model Properties Verification	Reluplex & Eran using DNNV
Model Optimisation	Pruning + Retraining

CONCLUSION
DISCUSSION

- CoDANN Issue 1.0 aims at addressing some of the **major challenges and risks associated to the implementation of neural networks in safety-critical applications**.
- It is an "**enabler** towards the certification and approval of machine learning […] in safety-critical applications."
- But much **many more challenges** are yet to be overcome:
  - The field of deep neural networks development and verification is in constant evolution. New methods are continuously being developed. There are still very few stable reference methodologies/techniques.
  - Limited scope of CoDANN Issue 1.0 :
    - Strong emphasis on data management and concept of "generalizability" but does not yet cover "explanability" issues.
    - Focuses only on non-recurrent CNN and on a specific use case (runway detection and corner coordinates computation). Not easily "generalizable" to other applications or neural network technologies.
    - Does not yet address:
      - Methodologies and software tools qualification, training environment and hardware qualification, inference hardware qualification,
      - Possible model evolution after initial certification,
      - Relationship between metrics (e.g. accuracy, similarity index…) and outcome of safety assessment.

ANNEX
OUTLINE

- Project Overview
- Data Management
- Learning Process Management
- Model Training
- Learning Process Verification
- Model Implementation
- Inference Model Verification
- Conclusion
- Annex
  - Software Modules Architecture
  - References

ANNEX
SOFTWARE MODULES ARCHITECTURE – 1/6

- **Data Management**
  - **data_collection.ipynb**
    - Objective – To collect the pictures and record associated information (date/time, lighting condition, distance/angle Jetbot/holding point…).
    - Output – .zip file containing pictures (.jpg files) and .csv file recording the metadata + hashing information for all files.
    - Platform – Jetbot
  - **data_augmentation.ipynb**
    - Objective – To check the integrity of the data collected with the robot and perform data augmentation.
    - Output – 3 different Training/Validation Datasets (based on the same original data but using different data augmentation strategies) + .csv file recording the metadata .
    - Platform – Local PC
  - **training_dataset_normalization_parameters.ipynb**
    - Objective – To compute the mean and standard deviation of the 3 Training/Validation datasets.
    - Output – .csv file containing the mean and standard deviation for each channel (R/G/B) to be used to normalize each training dataset as well as the average for each parameter to be used with the test dataset and in service.
    - Platform – Colab

ANNEX
SOFTWARE MODULES ARCHITECTURE – 2/6

- **Data Management (continued)**
  - **discriminator.ipynb**
    - Objective – To find the best layer for Early Layer OOD Discriminator and build the SVM Model.
    - Output – Best candidate layer for early layer SVM + SVM Model.
    - Platform – Colab
  - **Other files**
    - TRG_DATASET_NORM_PARAM.csv – Output of training_dataset_normalization_parameters.ipynb.
    - augmented_csv.csv – Sample traceability file for Training Dataset 1.

- **Learning Process Management**
  - **learning_process_management.ipynb**
    - Objective – To select the hyperparameters (e.g. batch size, learning rate…) to be used for the training of the final models.
    - Output – Training strategy (hyperparameters…).
    - Platform – Colab

ANNEX
SOFTWARE MODULES ARCHITECTURE – 3/6

- **Model Training**
  - **model_training.ipynb**
    - Objective – To train the 3 final models in accordance with the training strategy defined previously.
    - Output – 3 trained models.
    - Platform – Colab
  - **stability_analysis_cca_cka_results.ipynb**
    - Objective – To verify the stability of the training algorithm.
    - Output – CKA Similarity index for each layer and each pair of models + Result similarity metric for each pair of models.
    - Platform – Colab

ANNEX
SOFTWARE MODULES ARCHITECTURE – 4/6

- **Learning Process Verification**
  - **learning_process_verification.ipynb**
    - Objective – To verify the performance of the trained networks using the test dataset.
    - Output – Performance metrics for each model.
    - Platform – Colab
  - **temperature_scaling.ipynb**
    - Objective – To compute the Temperature Scaling Coefficient for each model and calibrate the trained models.
    - Output – Temperature Scaling Coefficient for each model.
    - Platform – Colab
  - **model_verification_DNNV.ipynb**
    - Objective – To verify the properties of the trained models using the Reluplex and Eran methodologies (through the DNNV package).
    - Output – Properties verification pass/fail for each model and each methodology.
    - Platform – Colab

ANNEX
SOFTWARE MODULES ARCHITECTURE – 5/6

- **Model Implementation**
  - **model_optimization.ipynb**
    - Objective – To prune the trained models and retrain them.
    - Output – 3 pruned models and associated performance metrics.
    - Platform – Colab
  - **SVM_model_training.ipynb**
    - Objective – To recompute the SVM model used to detect out of distribution images.
    - Output – SVM model.
    - Platform – Jetbot
  - **inference_model.ipynb**
    - Objective – To implement the final system on the Jetbot platform, i.e. live camera feed image pre-processing + pruned models + discriminator (SVM model) + voting/filtering + display + guidance.
    - Output – End-to-end solution capable of processing camera feed in real-time and raise alerts if a holding point is detected.
    - Platform – Jetbot

ANNEX
SOFTWARE MODULES ARCHITECTURE – 6/6

- **Model Implementation (continued)**
  - **Other files**
    - SVM_model.sav – Locally retrained SVM model, output of SVM_model_training.ipynb

- **Inference Model Verification**
  - **inference_model_test.ipynb**
    - Objective – To test the performance of the pruned models once embedded on the Jetbot platform using the test dataset.
    - Output – Performance metrics.
    - Platform – Jetbot

ANNEX
REFERENCES – 1/3

- **Project Overview**
  - [PO1] "Artificial Intelligence Roadmap – A human-centric approach to Ai in Aviation", EASA Report (document)
  - [PO2] "Concepts of Design Assurance for Neural Networks (CoDANN)", EASA AI Task Force and Daedalean AG (document)
- **Data Management**
  - [DM1] "Out-of-Distribution Detection in Deep Neural Networks", Neeraj Varshney (article)
  - [DM2] "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks", Dan Hendrycks, Kevin Gimpel (article)
  - [DM3] "Enhancing the reliability of out-of-distribution image detection in neural networks", Shiyu Liang, Yixuan Li, R. Srikant (article)
  - [DM4] "Generalized ODIN: Detecting Out-of-distribution Image without Learning from Out-of-distribution Data", Yen-Chang Hsu, Yilin Shen, Hongxia Jin, Zsolt Kira (article)

ANNEX
REFERENCES – 2/3

- **Data Management (continued)**
  - [DM5] "Detecting Out-of-Distribution Inputs in Deep Neural Networks Using an Early-Layer Output", Vahdat Abdelzad, Krzysztof Czarnecki, Rick Salay, Taylor Denounden, Sachin Vernekar, Buu Phan (article + implementation)
- **Model Training**
  - [MT1] "Insights on representational similarity in neural networks with canonical correlation", Ari S. Morcos, Maithra Raghu, Samy Bengio (article + implementation)
  - [MT2] "Similarity of Neural Network Representations Revisited", Simon Kornblith, Mohammad Norouzi, Honglak Lee, Geoffrey Hinton (article + implementation)
- **Learning Process Verification**
  - [LPV1] "On Calibration of Modern Neural Networks", Chuan Guo, Geoff Pleiss, Yu Sun, Kilian Q. Weinberger (article + implementation)
  - [LPV2] "DNNV: A Framework for Deep Neural Network Verification", David Shriver, Sebastian Elbaum, Matthew B. Dwyer (article + implementation + documentation + video)

ANNEX
REFERENCES – 3/3

- **Learning Process Verification (continued)**
  - [LPV3] "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks", Guy Katz, Clark Barrett, David Dill, Kyle Julian, Mykel Kochenderfer (article + implementation)
  - [LPV4] "ETH Robustness Analyzer for Neural Networks (ERAN)" (implementation)
- **Model Implementation**
  - [MI1] "Deep Model Compression and Architecture Optimization for Embedded Systems: A Survey", Anthony Berthelier, Thierry Chateau, Stefan Duffner, Christophe Garcia, Christophe Blanc (article)
  - [MI2] "A Survey of Model Compression and Acceleration for Deep Neural Networks", Yu Cheng, Duo Wang, Pan Zhou, Tao Zhang (article)