D3.2 Interim report of DL components and library

Version 1.0

Documentation Information

Contract Number	101069595
Project Website	www.safexplain.eu
Contractual Deadline	31.03.2024
Dissemination Level	PU
Nature	R
Author	Robert Lowe (RISE), Thanh Bui (RISE)
Contributors	Axel Brando (BSC), Jaume Abella (BSC)
Reviewer	Shruthi Gowda (NavInfo)
Keywords	Artificial Intelligence, safety, Explainability, Software engineering

This project has received funding from the European Union's Horizon Europe programme under grant agreement number 101069595.

Change Log

Version	Description Change
V0.1	First draft
V0.2	Reviewed version
V1.0	Final version

Table of Contents

Executive Summary

This document outlines our investigation into supporting algorithms and techniques that permit the DL components developed within the AI-FSM safety lifecycle to be" dependable", i.e. to be compliant with the process and guidelines provided by deliverable D2.1[1], using the proposed approaches from D3.1[2]. By providing realization examples of DL components, including the algorithmic means to make these components explainable and traceable, we will evaluate and develop these supporting algorithms within the context of our DL development project. There are two library components being developed: i) EXPLib (Explainable Deep Learning Library) – a project library interfacing with the AI-FSM lifecycle management process as part of the inter WP2-WP3 works, relevant to task T3.4, and ii) DLLib (Deep Learning Deployment Library) - a project library, relevant to the requirements of task T3.5, which is designed to support deployment, interfacing with the platform provided by WP4 and supporting the three use cases. DLLib receives a trained DL model (model definition, parameters) and supporting components (supervisor, explainers) from EXPLib, which can then be ported into low-level c-libraries (suitable for usage on the project-selected NVIDIA Jetson AGX Orin platform). These libraries contain a number of python files, and descriptions. They are currently being developed and stored in a shared google Drive repository but will ultimately be developed as a repository hosted by GitHub/GitLab. Within our Drive shareable repository, we provide a simple colab walkthrough script that accesses available functions/scripts within the EXPLib project library (providing loadable models and supporting components for "DLLib"). EXPLib and DLLib are also the names of the root folders that contain hierarchically structured sub-folders that contain related functions/classes/data). The hierarchical folder structures of EXPLib and DLLib are realized through python libraries. For the purpose of providing a simple mnemonic, this relationship is illustrated in Figure 1.

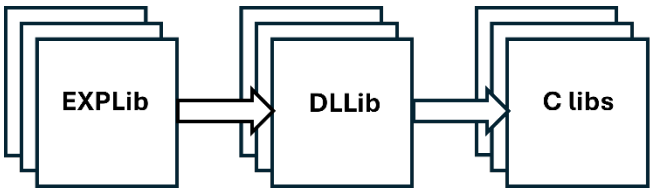


Figure 1: The three libraries relevant to D3.2. EXPLib provides supporting functions relevant to the phases of AI-FSM, which utilizes explainability/traceability core functions to ensure the dependable Deep Learning component to be developed/verified. The DLLib receives the result artifacts from the EXPLib in the form of verified DL model (both for the DL component and for the trainable explainer models, where relevant). The C libs receive input from the DLLib in the form of specifications for conversion (in relation to the models and supervisor components of those models) for real-time (NVIDIA Jetson AGX Orin platform) deployment.

1 Introduction

The D3.2 Interim report describes in-progress implementation and workings of DL components and supporting XAI libraries (T3.4, T3.5), following the approaches described in D3.1. Specifically, this document reports on the progress undertaken on the implementation of the libraries to support dependable DL component engineering in compliance with AI-FSM process guidelines (D2.1) and safety patterns (D2.2[3]). It also covers the interim report on the low-level optimization libraries for the selected deployment components.

The main purpose of this interim report is to provide a proposed library structure, with initial content, for supporting various phases of AI-FSM Safety Lifecycle (as reported on in D2.1). We refer to this structure as the **EXPLib** (task T3.4) - *Explainable Deep Learning Library*, which supports various phases of the AI-FSM Lifecycle illustrated again for convenience in Figure 2.

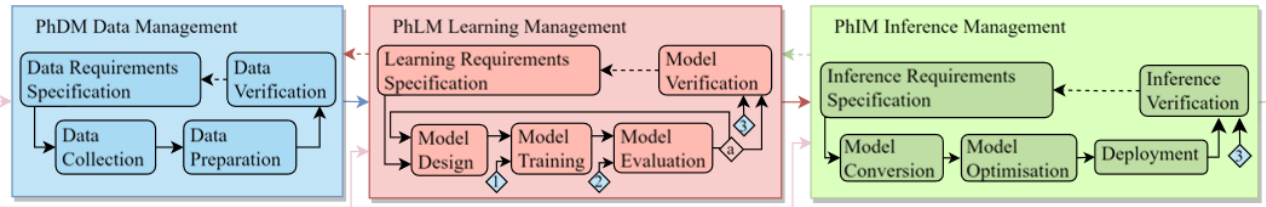


Figure 2: Phases of the AI-FSM (taken from D2.1). EXPLib provides algorithms for supporting the above phases core to AI-FSM, PhDM (Data Management phase), PhLM (Learning Management phase), PhIM (Inference Management phase). When requirements are met in PhIM, The DLLib is able to load the trained DL models (model definition and parameters) and supporting components (explainers, supervisors) that have met the PhIM requirements specification.

This term (EXPLib) allows for differentiation from **DLLib** (task T3.5) - *Deep Learning Deployment Library* - the focus of which is on 'porting' the EXPLib developed artifacts (verified DL components, supervisory monitor components, algorithms for decision function) into the SAFEXPLAIN selected deployment platform (NVIDIA Jetson AGX Orin platform). This device utilized in the Deployment/Inference phase uses a model, converted into low-level C libraries in integration with SAFEXPLAIN ORIN platform software stack (based on NVIDIA Jetpack), and potentially quantized to allow for faster real-time inference. EXPLib uses python-based AI frameworks for implementation and is currently focused on pytorch, but will also include tensorflow/keras and ONNX implementations to interface between python deep learning AI frameworks. We report here on an initial selection and implementation of a number of DL components, 'XAI' supporting tools and Supervisor component candidates that have been listed in deliverable D3.1.

The library structures of both EXPLib and DLLib is intended to be reusable; that is, there should be minimal requirements of the user when importing data and models (loading or training) for testing different AI-FSM phases with different explainer-based algorithms.

Notwithstanding, devising libraries that allow for simple reuse is not trivial. State-of-the-art based DL components, e.g. Object Detection or Image Classification algorithms are implemented using different AI frameworks (with focus on tensorflow, keras, Pytorch, ONNX) with different library structures and using different external libraries or dependencies (some of which, in turn, may be designed for a specific operating system). Selected deep learning models may utilize post hoc explainer algorithms or may have built in (intrinsic) explainable properties. The task of composing a DL library, therefore, is to adopt available state of the art DL components, so that they can be adapted to follow the proposed AI-FSM safety lifecycle, using a standardized library structure for *two different perspectives*:

- *AI-FSM perspective*: data management, learning management, inference management and operational XAI. The last item "Operational XAI" is not within the AI-FSM phases but it provides realization code related to Section 4 - Improved DL component robustness and online monitorability (also see D3.1[2]). Operational XAI serves as the integration point with DLLib where all required components for the deployment are to be found.
- *Explainable AI core technique perspective*: Data explainers, model explainers, metric extractor and supervisor algorithms are utilized and combined for providing mobility systems-based application solutions (explanation and transparency of deep learning model predictions).

In summary, the goal of producing the EXPLib is to:

- *Facilitate AI-FSM activities in the development* process of deep learning-based algorithms (models and complementary explainable components), and
- *Support generating evidence, as required throughout AI-FSM lifecycle phases, for each AI-FSM phase meeting requirements specifications*: Data management phase, Learning management phase and Inference Management phase.
- *Support the development of explainable components* for use in Operation and Monitoring phase (deployment phase), represented as Operational XAI.

The goal of the DLLib is to:

- *Provide tools required for ORIN platform*: Port to the inference platform (ORIN platform with middleware libraries) the AI-FSM verified DL components and XAI components for final verification in the inference environment, the verified models are then ready for conversion to the C libraries to be used with the ORIN platform (connecting to T3.5), i.e. deployment/inference environment.
- *Integration with the deployment platform*: Monitoring mechanism, deployment requirements, and other requirements from the middleware safety control mechanism.

It is worth noting that besides the above set goal, a very important role of the two libraries within the project lifetime is to baseline and support the constantly evolving alignment and understanding of project partners, who are multi-disciplinary researchers, from different industrial domains. The interactive work and continuous technical debates, leading to the realization examples from the libraries, will help to gradually identify and close the gaps between different perspectives: strict functional safety, opaque nature of deep learning and practicality of industrial requirements.

Furthermore, the temporary repository environment of the EXPLib library in a shared google Drive has also allowed for the use of a runnable EXPLib Colab script (namely aifsm_eval.ipynb) to test-run portions of the code within the libraries. Notwithstanding the limitation of using Colab for development for large software systems (limitation of allocated computing resources and real-time requirement support), Colab was elected on the basis that it is not OS specific, allows for cloud access of accelerators (GPUs) and thereby facilitates co-development within the project. The tool allows for a walkthrough of tested/testable components for the different AI-FSM phases/activities, e.g. for a specific dataset and specific DL component (model, e.g. Yolo7), and cell by cell documentation relevant to the phases of AI-FSM. With appropriate instruction (See README.txt file in Appendix B) a subset of the cells in the colab script can be run depending on

the particular requirement being tested, e.g. individual AI-FSM phases/activities, or a combination thereof.

In summary, the goal of the use of an EXPLib colab demo script is to:

- allow for platform (and hardware)-independent evaluations of functionality of (python) implementation of the library in supporting AI-FSM phases/activities,
- allow intuitive guidance for the audience to walk through phases of the AI-FSM with respect to the target DL components and related XAI algorithms.

Note, we do not provide a DLLib colab demo script because unlike EXPLib the focus of this library is not on testing multiple, partially independent parts of a process (AI-FSM phases) but rather on providing the means for verified components to be translated to c libraries.

2 Objectives and Report Breakdown

This deliverable reports on the progress achieved at M18 on the realization of the AI-FSM aware DL components and the usage of **EXPLib** (Section 3) to support the development process of deep neural networks. It also reports on a **DLLib** (Section 4), which provides realization of support for the deployment phase.

In Section 3, the focus is on describing the EXPLib: DL models investigated XAI library relevant to each phase of the AI-FSM, the AI-FSM activities/artifacts themselves and how they leverage the available supporting functions from XAI library. This section is completed with a detailed description of the Colab walkthrough demo of usage of the EXPLib in relation to the phases of the AI-FSM cycles (3.5), and an initial plan to build the Minimum Viable Product (or MVP) that will showcase an end-to-end example of how all components shall work together throughout the development lifecycle through to the deployed operation (3.6).

The DL components populated in the EXPLib include the DL models/networks (e.g. Yolo7[4]) taken as examples of how they will be processed and evaluated through the AI-FSM process. The DL component undergoes performance and explainability checks culminating in potentially pruned and quantized form specific to the Operational Design Domain (ODD). The DL components also include trainable models that can be used together in the safety architecture for deployment/inference, e.g. supervisory monitors such as anomaly detectors, that are trained on the same datasets as that of the DL model and are thereafter able to evaluate whether incoming data in the operation mode deviates significantly from that which it was originally trained/verified on.

In relation to the AI-FSM phases of PhDM (Data Management), PhLM (Learning Management) and PhIM (Inference Management) there is a need to compare the required performance and robustness of elected models (using safety and goal driven performance metrics) as well as explainability of these models. Such comparison will benefit from using a common interface framework (library for these phases, which will be a part of EXPLib) since many algorithms use different frameworks with different requirements and libraries.

Various intrinsic explainability and posthoc XAI techniques are required for different usage/audience types at each of the AI-FSM phases, which we will refer to as XAI algorithms.

In the PhDM -Data Management phase - we include supporting scripts using available explainable AI (XAI) algorithms designed to summarize, describe and verify the quality of datasets in line with

specifications outlined in the AI-FSM *PhDMT0001_Data_Requirements_Specifications_template* and *PhDMG0001_Data_Management_guideline*. The applicable library components thus include methods for data profiling (summary report describing statistical properties of a dataset), data analysis plots (e.g. feature-based data distributions analyses), data prototyping (e.g. use of clustering methods to extract representative data prototypes), and data descriptors (e.g. using generative models to describe the dataset space). Within the EXPLib we also make use of more traditional (as opposed to more modern XAI) methods for data analysis (e.g. class distribution, labelling/annotation consistency, dimension reduction analyses). At this phase, the boundary conditions of inlier/outlier will also be logged for later usage with data anomaly detectors in Operation phase.

In the PhLM – Learning Management phase - we include XAI-enabled tools to support model election, training, validation and verification processes in line with the AI-FSM *PhLMD0001_Learning_Requirements_Specifications* and *PhLMG0002_Learning_Management_Guidelines*. The components include methods for evaluation of training settings (hyperparameters), application of intrinsic/posthoc/antehoc explainability to the target DL model, disentanglement w.r.t. concepts and/or input conditions, evaluation of test result validity, and provisioning baselines of known model behaviors via logs of activation patterns or using intrinsic surrogate models as representative model behaviors.

In the PhIM – Inference Management phase - we include XAI-enabled tools for evaluating the effects of using conversion/optimization techniques for adapting to the inference environment and pre-deployment testing. In line with *PhIMG0003_Inference_Management_Guideline* (D2.1) and the proposed approaches in D3.1, we explore XAI mechanisms to support assessment of potential performance differences between the original and converted/optimized model, both in terms of accuracy and explainability consistency. Here approaches include comparing performance (e.g. mAP and inference speed, FPS) as well as explainability consistency (e.g. in reference to feature representations, via L2, and cosine, distance measures, e.g. ShapGAP[5]).

In the EXPLib and DLLib we also include an initial set of implemented metrics for extracting performance measures and explainability measures to be logged and compared against the requirement specifications. A special class of metrics, i.e. runtime metrics, are also implemented for monitoring DL performance and uncertainty during Operation and used for “L1 diagnosis and monitoring” as in the D2.2 reference architecture. Monitoring metrics are to be extracted from the C-code converted model when deployed in Inference/Operation environments (Orin platform). Furthermore, what monitoring metrics and explanations are to be made visible to humans at runtime for decision making, and what are to be automatically dealt with by the Supervisory monitors (i.e. leading to rejection or acceptance of the Deep Learning component prediction) will also be updated following the inference phase using the C libraries and deployment on the Orin platform.

Finally, work towards the Minimum Viable Product (MVP) will be exemplified via a walkthrough example of usage of the EXPLib and DLLib as implemented in python using the pytorch deep learning framework and facilitated by the use of a colab runnable script for cross-WP/partner integration. This walkthrough currently utilizes the Yolo7 model (to be replaced by a toy model), as an example of a near-to-state-of-the-art object detector DL model, trained on the PASCAL-VOC dataset, and with reference to selected approaches (callable using a runnable colab notebook) for

each of the AI-FSM phases: PhDM, PhLM, PhIM. The trained models (including AI-based models trained for the purposes of supervisory monitors and decision function usage in the Operation phase, e.g. via anomaly detectors) are saved and loadable to the DLLib files for evaluating Operational XAI.

In Section 4 of this document, the focus is on describing the DLLib, i.e. verified DL component together with supplementary runtime components to support the reference architecture patterns, e.g. supervisor components, decision function and metric extractors. This is designed to provide monitoring of the trained DL models (and explainers), i.e. the DL component, so that it is ready for conversion into the C libraries for deployment on the final deployment platform NVIDIA Jetson AGX Orin.

3 EXPLib – A library for explainable and dependable DL components

Implementation of the DL components constitutes a realization of explainable AI architectures as described in D3.1. As part of the specification, design, and improvements of DL components, we will apply a safety assurance process AI-FSM to train, validate and deploy the proposed designs. Realizations will be based on popular AI frameworks such as TensorFlow, Keras, Pytorch, and will engineer the most popular and representative neural network types (MLP, CNN, RNN, LSTM, Transformers). The Minimum Viable Product (MVP) will be based on a generalization of the case study requirements into a toy model (D5.1[6]).

We can identify five phases of development of the EXPLib and DLLib libraries, which will continue to inform further development (beyond M18):

- *Pre-insertion phase*: This is a test phase that was carried out by several of the partners in relation to different XAI algorithms (as listed in D3.1). This phase allowed us to assess the suitability of the algorithms in terms of their function, relevance, and simply whether they work as expected.
- *Insertion phase*: In this phase, functions/scripts are inserted into appropriate folders within the EXPLib directory that concern the different AI-FSM phases or otherwise should be considered part of the XAI library.
- *Integration into colab (or main runnable) program*: Here, for purposes of providing platform independent interaction with SAFEXPLAIN partners, we have implemented a colab script - aifsm_eval.ipynb. A subset of functions/scripts inserted into the EXPLib library/directory are currently runnable from this script including the subset that contribute to the walkthrough example. Others are presently not runnable through this script but are put in backlog for implementation.
- *Testing of phases within the runnable program*: By evaluating usability of various algorithms relevant to the different AI-FSM phases/activities it is possible to produce output artifacts, e.g. logs, filled AI-FSM templates, trained explainer models, deep learning components/models. The optimal model (via metric-based evaluation such as those belonging to performance and explainability criteria) can then be selected.

- *Utilization of saved models in DLLib*: Following i)-iv) it is possible to load trained models (including explainer models) and a set of other variables (based on logged reports of relevant metrics, e.g. for monitoring DL performance and explainability) in the DLLib. DLLib provides the implementation of the deployable (verified in EXPLib) DL component that includes the original Deep learning model (e.g. Yolo7), the explainer models necessary for supervisory monitor and the set relevant metrics for supervisor monitoring.

The *pre-insertion* phase has largely been described in D3.1, i.e. description of tested XAI algorithms that are relevant to the different AI-FSM phases. D3.2, therefore, will instead focus on phases bullet points 2-5. A description of the entire process will be described in relation to our walkthrough ("toy") example for evaluating the dependability of the DL component (see Section 3.5). It should also be noted that in later instantiations of EXPLib, we will use non-colab scripts (i.e. .py scripts) within the main SAFEXPLAIN GitLab repository for further development and finetuning. However, we will still seek to allow for a colab runnable script for providing alternative means of access.

3.1 EXPLib library structure

Figure 3: First and second levels of the EXPLib directory (with third level for CNN also visible)

The structure of the libraries containing the DL components (including explainable components) are subject to the safety assurance under AI-FSM Lifecyle and referenced in D3.1; they utilize the supported phases: (i) PhDM-Data Management, (ii) PhLM-Learning Management, and (iii) PhIM-Inference Management. An extra, special, phase Operational XAI, that is not an official part of the AI-FSM phases, will serve the transition from the development/test environment to the deployment environment. Here, a focus is on incorporation of XAI and (pre-deployment) supervisory components for ensuring the dependability of the DL components in operation.

The fuller visualization of the implementation of the EXPLib (and DLLib) about which this interim report concerns can be visualized in Appendix A (see Figure 38).

Since the library structure of EXPLib is somewhat complex we will summarize and visualize it according to its constituent sub-parts. In Figure 3, the structure of the EXPLib is broken down into the first two levels.

- **models** (Dependable DL components/neural networks): a number of DL components are incorporated within the DL libraries at this stage using the Pytorch Deep Learning framework whilst other frameworks, e.g. Tensorflow, ONNX, will also be realized in forthcoming work (beyond M18)
- **xai_library** (core XAI algorithms): this sub-library contains all candidate XAI algorithms that can be used throughout the safety development lifecycle and the Operation and Monitoring stage.
    - *data_explainers*: Contains python classes and functions that can be used to explain a data sample or a dataset. Examples include: Data profiling, data prototype extractor, data plots
    - *model_explainers*: Contains python classes and functions to explain models. Examples include: saliency map, LRP, feature importance, etc
    - *metric_extractors*: Contains python functions/classes to compute different metrics for model, explainability, coverage.
    - *supervisor*: Contains generic python functions/classes that can be used to build supervisory monitors. This sub-library also includes decision function related algorithms to analyse outputs from DL component and supervisor monitors into a prediction and trustworthiness level.
- **aifsm_phases** (AI-FSM phase wise perspective): stakeholder scripts that call functions contained in the *xai_library* to support different steps within each AI-FSM lifecycle phase perform their tasks (Figure 4) in connection of the target DL components (including complementary explainable components). The systematic approaches are described in D3.1. The following phases are supported:
    - *PhDM-Data Management*: supports data management process and steps, such as evaluating a dataset, summarizing a dataset, extracting representative prototypes.
    - *PhLM-Learning Management*: supports adding explainability into DL model (either by modifying the network or posthoc explainability).
    - *PhIM-Inference Management*: the key supporting tools for this phase are meta-heuristic search tools to support V&V activities, and metric extractors.
    - *Operational XAI*: An extra phase that is not an official part of AI-FSM phases, will serve the transition from the development/test environment to the deployment environment. Here, a focus is on incorporation of XAI and (pre-deployment) supervisory components for ensuring the dependable DL components in operation.
- **datasets**: Containing related datasets that will be used within the SAFEXPLAIN experiments. The datasets are collected from public datasets, webcam captured dataset, customized datasets (e.g. railsem19) and a toy model related dataset.
- **UCs**: the three use cases - ADAS, railway, space.
- **configs**: general configuration files to support the configurable structure of the library
- **utils**: general utilities functions.

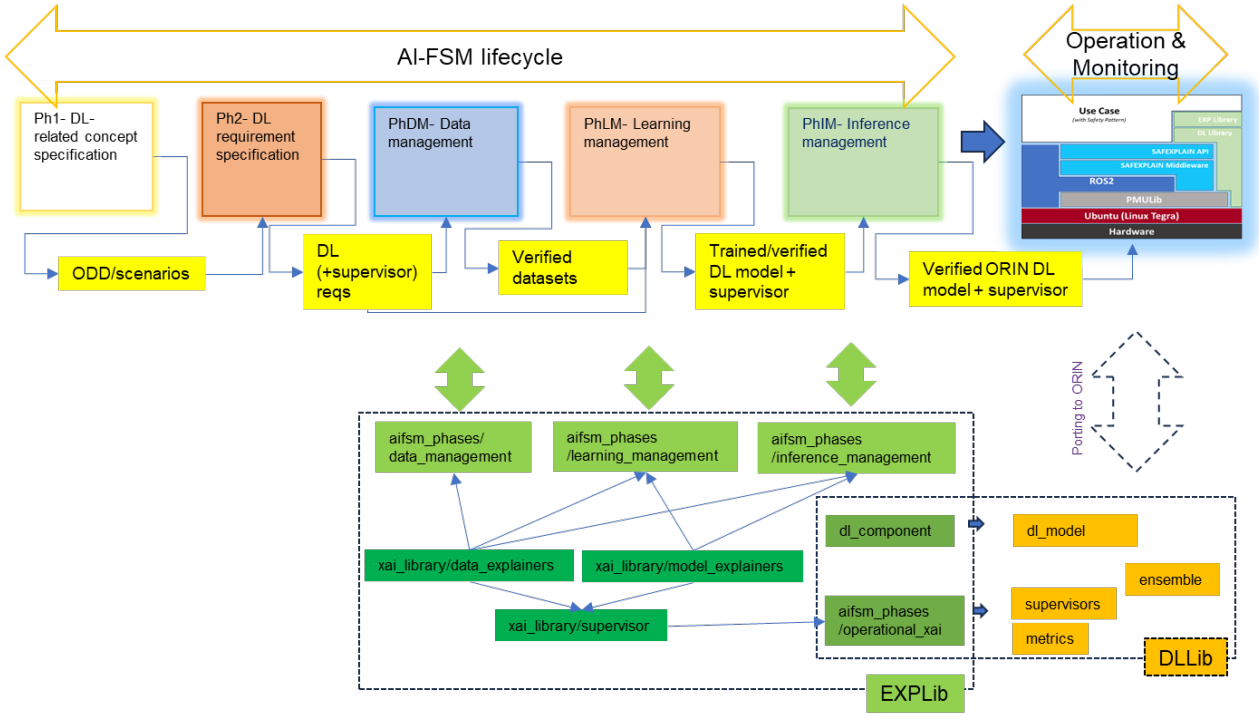


Figure 4: The utilization of EXPLib and DLLib are depicted in relation to the AI-FSM phases developed in WP2. The green pages represent input/output artifacts of these phases that together build up the DL specification.

In the following sub-sections, we will provide some details of EXPLib sub-components.

3.2 DL models

In the path *EXPLib->models* exist a number of folders labelled by the neural network types/models. In Figure 5 these are visible as MLPs, RNNs (incl. LSTMs), Transformers, CNNs and AEs.

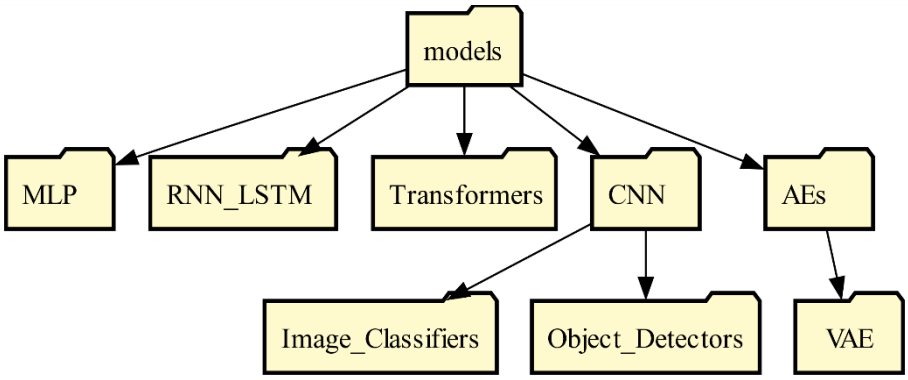


Figure 5: Sub-directory for the Deep learning/Neural Networks based models that we have investigated/will be investigating.

We have been reviewing (reported in D3.1) and primarily evaluating several explainer algorithms designed to reduce the black box nature of DL based models for all of the model types labelled here, e.g. DeepRED[7] (used for extracting logic rules, e.g. XOR, typically for densely connected networks, therefore related to MLPs). The models we have focused our attention on at this stage, in relation to evaluating within the AI-FSM process are CNN-based: Image Classifiers (e.g. VGG-based neural networks, or ResNet-based), or Object Detectors (e.g. Yolo-based models) - see

Figure 6. We have also investigated AEs (Autoencoders), above all Variational Autoencoders (VAEs), as means for evaluating data distributions (PhDM) and for anomaly detection (PhLM, PhIM).

The files that are stored in the folders specifying the model types are those necessary for loading trained models for testing in different phases of the AI-FSM process, e.g. in the Inference phase (PhIM) whereby a model architecture definition file and trained model checkpoints (for obtaining weights) can be loaded.

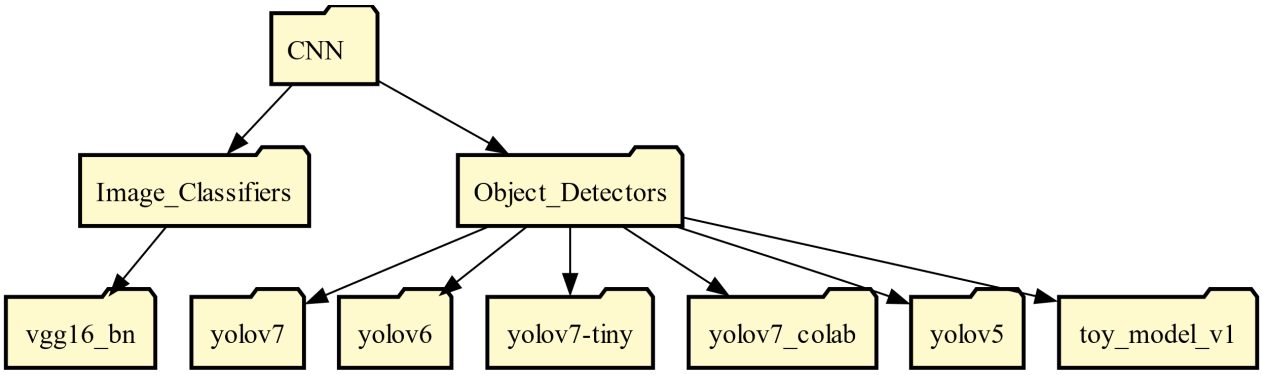


Figure 6: Path directory to CNN model types. Image Classifiers contain model implementation code (architecture definition) and model checkpoint files (for obtaining the trained model weights).

The currently selected target DL models are based on the visual based CNN[8] architecture, or "backbone" for object detectors, currently included in the EXPLib are models for:

- Object Detection:
  - Candidate models for Railway UC: Yolo5, Yolo6, Yolo7 [4];
  - Toy model: SSD MobileNetV3[9], [10]
  - *Candidate models for Automotive UC to be included: YoloX, NanoDet*
- Classification:
  - VGG16[11]: batch normalized standard deep Convolutional Neural Network (CNN) with 16 convolutional layers.
  - *Prototree[12]: intrinsic explainable ResNet[13] with hierarchical prototype decision tree. To be included.*

A generative DL model (Variational Autoencoder), intended to be used as a candidate for Operational XAI supervisor function, is also included in the DL models. The motivation for its inclusion is that this supervisor function itself is also a DL-based model, and thus also needs to go through the safety assurance process similar to the main target DL model.

3.2.1 CNN: Image Classifiers

Image classifier models are stored in the folder in Figure 6 of the same name. We also provide python notebook (.ipynb) files for testing on the cloud (via Colab), which reduces requirements for platform-dependent installation. The selected CNN models are placed/will be placed within this sub-directory path – i) script for architecture and functions (e.g. yolo7.yaml and yolo.py), ii) .pt files for weights (trained model parameters). Since the focus of SAFEXPLAIN's Railway Use Cases (Railway, Automotive, and Space) investigation is on Object Detector models such as YOLO models, we have so far only studied VGG16 as an Image Classifier model (specifically the batch normalized version thereof available at pytorch: https://pytorch.org/hub/pytorch_vision_vgg/). The models

to be studied will include popular versions of CNNs, particularly those that are relevant to (e.g. share computational functions with) Object Detectors: VGG19 (batch normalized), ResNet[13]/ResNeXt[14] models, RepVGG[15]-based models. Since Darknet[16] is implemented in C libraries we may not include this within EXPLib. These CNNs also often provide full "backbones" for versions of YOLO used for feature extraction and, therefore, in-depth study in relation to application of XAI algorithms of such algorithms serves a practical purpose for our Use Cases.

3.2.2 CNN: Object Detectors

The Object detector algorithms included so far are versions of YOLO, specifically YOLOv5/6/7 all of which have been released since 2020 but have been well studied providing us with foundations for deriving insights in relation to applications of XAI algorithms. These three algorithms have also been utilized in relation to XAI algorithms designed for providing explanations of predictions of object detectors L-CRP (Localized-Concept Relevance Propagation)[17], Kernel Shap[18].

More models will be included in the near future: SSD based Toy model(D5.1[6]), Yolov7-tiny, Yolov8. We will consider later versions of Yolo as they are released, e.g. Yolo9 (recently released), depending on factors such as peer review.

3.2.3 AE: Autoencoders

The Autoencoder[19] is a type of DL model for unsupervised learning. The key usage of this type of model is to encode high dimensional input data (e.g. image datasets) into a smaller dimensional feature space (also referred to as latent space), and then use the extracted features to reconstruct, as close as possible, the original input data space.

Within SAFEXPLAIN we leverage these special properties of the Autoencoder to use for the following key purposes:

- *Dataset descriptor*: we use a special class of AE, namely Variational AE (VAE) where the reconstruction quality is not gauged via a single data sample but rather with respect to how to represent the entire dataset. A well-designed VAE (number of layers, latent space dimensions, convolution layers, kernel sizes, etc.) trained on a dataset, thus can be considered an advanced data descriptor to describe that dataset. We make the assumption that the trained VAE is considered as a *valid* dataset descriptor, where a data sample is represented by a specific latent vector in the low dimensional feature space. The data distribution will then be described by the statistical properties of the set of features describing the training/validation/test datasets.
- *Data anomaly detector*: Once the decision to use VAE as data descriptor is made and accepted by the certifier during the AI-FSM process, we can then assume that the VAE can be used to detect in operation mode which data can be classified as "known" or "unknown" data by comparing the distance from a specific input data point to the logged distributions of known datasets during the AI-FSM development lifecycle.
- *Neuron activation pattern anomaly detector*: The above approach can then also be used to apply to a special type of data, i.e. interim features represented by activation patterns of a specific layer of the DL network. By that, we assume that the anomalous neuron activation patterns in operation represent anomalous model behaviors in response to a specific situation in the real world. The reason is that, for the V&V that is based on neural coverage, the anomalous patterns are not included and thus cannot be verified as safe in runtime.

3.3 AI-FSM lifecycle tools

These tools can be broken down into the AI-FSM phases for Data Management (PhDM), Learning Management (PhLM), Inference Management (PhIM). We describe in this sub-section the tools used in EXPLib to support the activities in these phases including the XAI techniques relevant to the activities.

3.3.1 Data Management

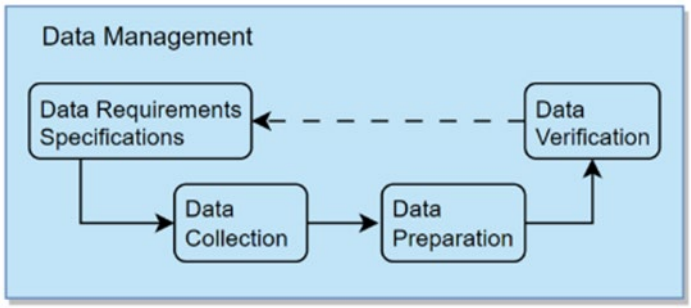


Figure 7: Data Management phase (PhDM) (originally from D2.1). The activity "Data Verification" is relying on the support of "XAI" based algorithms (xai_library) in order to evaluate facets of explainability. Data Collection and Preparation sub-phases are also represented within EXPLib as a series of supporting functions.

Data Management (Figure 7, originally from D2.1) uses PhDM_controller.py that calls several other functions to verify the quality of the datasets or describe them at different steps/activities, e.g. for data profiling and prototyping: in terms of class distribution, average pixel intensity, cluster analysis, etc.

- **Data Collection ->** a number of analyses can be carried out that assess the quality of the raw dataset(s), with respect to the Data Requirements Specifications:
  - *data format and consistency* – e.g. are all files .png files, .jpg files or a mixture?
  - *data volume* – dataset size
  - *dataset characteristics* - e.g. amount of data samples per input data dimension (data balance and representativeness properties)
  - *data type* - whether the data is real or synthetic (generated)
  - *data plots and profiling* – statistical reports and human understandable plots (showing a lower dimensional projection of the dataset or data points)
- **Data Preparation ->** the same analyses as in data collection are carried out but now comparing the pre-processed (prepared) data to the original dataset and in relation to the data requirements. Noted that the data balance properties are described by a desired distribution per specific dimension(s) of input data or per annotated class. Such pre-processing includes assessment for promoting:
  - *data augmentation* – adding more samples to the dataset using different transformations such as scale, rotation, flip to achieve the desired distributions.
  - *data normalization*:
    - *numerical stability*: Normalization of pixel intensities per class of data
    - *scale uniformity*: input features should be of equivalent or desired scale to avoid model bias to specific features.

        - *object characteristics*: normalization of annotation distributions, sufficient data samples (desired balance distribution) per class of objects.
    - *data cleaning/labelling*:
        - *denoising corrupt images*: use of denoiser model (e.g. autoencoder)
        - *manual re-labelling*: where labels are missing/incorrect.
- **Data Verification ->** Data Profiling and Data Prototyping:
    - *data profiling –*
        - *class/bounding box parameter distributions*: are classes of data sufficiently represented/evenly distributed? Do bounding boxes sufficiently cover the range of sizes and locations within the image space according to expected (e.g. normal) distributions?
        - *feature distributions and separability*: are classes of data potentially separable based on the feature characteristics and distributions within the dataset?
        - *Data profile summary report*: generate summary report of the dataset.
    - *data prototyping –*
        - *Prototype extraction via clustering methods*: to evaluate prototypical examples of image features that provide statistical representations of objects/concepts,
        - *Prototype patching*: do smaller patches of images demonstrate the existence of prototypical image parts?
    - *data descriptors –*
        - *VAE based descriptors: Using a trained VAE to describe the dataset.*
        - *Distribution descriptors: Logging the data distributions per dimension or per several dimensions. Dimensions can either be ODD/Operational parameters or annotations, or descriptor-based features (such as using VAE or prototypes).*

The key usages at the Data Management phase (or PhDM) are for: (i) dataset profiling to generate explainable data reports and (ii) dataset descriptors, and (iii) data analysis visualization tools. Figure 8, above, visualizes the current structure.

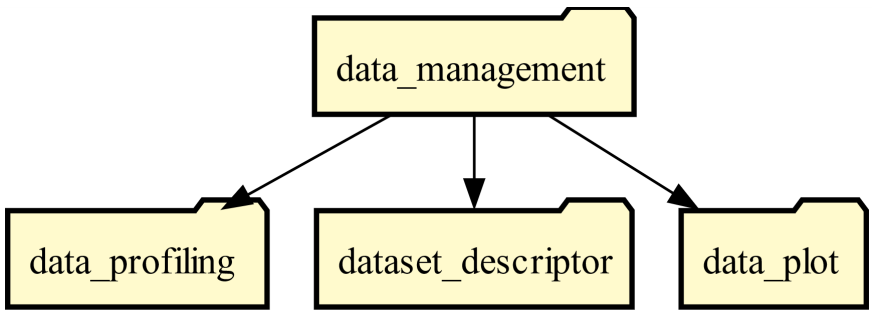


Figure 8: data_management subfolder

PhDM entails calling of *data_explainer* functions from *xai_library* in order to assess and understand the quality of the data and addresses the domain uncertainties. Within the EXPLib the *aifsm_phases->data_management* sub-directory it is possible to locate the PhDM_controller.py function that allows for a number of analyses to be run on the data, including but not limited to XAI based methods when calling the data_explainers.py script from within the *xai_library*. Note,

many data explanation methods might not be considered “Explainable AI” on the basis that they concern the use of analytic approaches that are not novel but could be applicable to any ML model engineering with regards to the datasets.

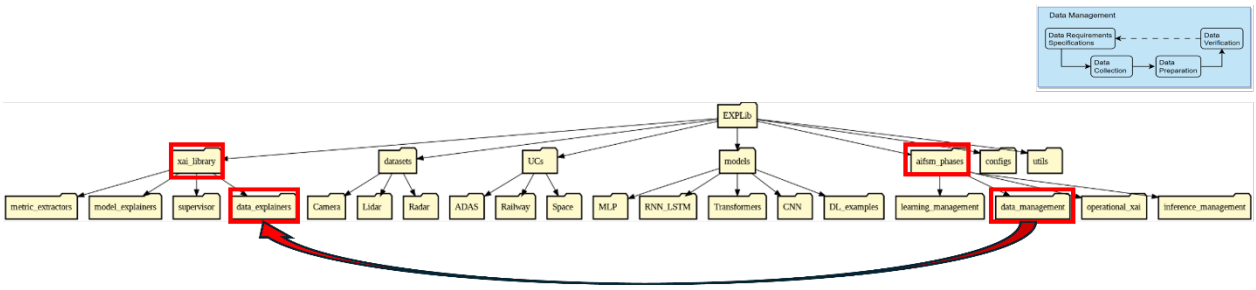
Whilst datasets contain folders categorized by sensor types of Camera, Lidar and Radar, Camera (image based datasets) has received until now the most focus on SAFEXPLAIN (relating to T3.1-T3.4 activities). Nevertheless, the Lidar sub-directory contains an example use of data. This is found in the Lidar->WADS path, script of data profiling (using yData profiling library) on this Lidar point cloud data is also provided.

The example datasets stored in the current version of the **EXPLib** include coco2017[20], CUB_200_2011[21], PASCAL-Parts[22] (labelled PASCAL-PART.v5i.yolov7pytorch; from Roboflow, which provides object part annotation files), rs19_val (which is a cropped version of the railsem19 dataset for the Railway use case – see Wp5). The current datasets stored within EXPLib, are mainly benchmark public datasets in the Deep Learning community for testing image classifiers and Object detector algorithms and already provide well-balanced datasets in terms of generic application. These datasets, however, can only be used as examples of how the libraries can be used. For the purposes of safety critical systems, the dataset should be collected/prepared in conjunction with the required ODD/operational scenarios and intended functionalities.

Our customized datasets for the 3 use cases (Railway, ADAS, Space) also require preparation techniques: Labelling annotation – data must be correctly annotated; Data augmentation – new data distribution must fit the expected real world distribution that may occur in the ODD; Data cleaning – anomalies, corrupt instances, must be removed from the dataset, cropping procedures must be appropriately standardized; Data pre-processing – normalization of data values, e.g. pixel intensities, feature selection, class distribution (from PhDMT0001_Data_Requirements_Specifications_template).

To undergo data preparation (for storage in the *prepared_data* folder), it is necessary to access scripts and functions in the *data_management* folder within *aifsm_phases* directory.

The called XAI algorithms from the *PhDM_controller.py* file within the *data_management* folder concern the Data Verification sub-phase depicted in Figure 7 and relate to *data profiling*, *data prototyping* (see section 3.4.1). Figure 9 depicts the relationship between the paths leading to *data_management* and *data_explainer*.



PhDM_controller.py

```
def datasetReqCheck(datadir, traindir, validdir, testdir, outputdir):
  cat_df, image_df = data_explainers.eval_class_dist(datadir, traindir, validdir, testdir, outputdir)
    # RL: probably better to retun a matrix for the pixel intensity info ...
  mean,median,mode,std_dev,range_of_data,skewness,kurtosis,iqr = data_explainers.pixel_intensity_analysis(traindir,outputdir)
  data_explainers.PCA_analysis(traindir,outputdir)
  data_explainers.UMAP_analysis(traindir,outputdir)
  data_explainers.tSNE_analysis(traindir,outputdir)
  return cat_df,image_df,mean,median,mode,std_dev,range_of_data,skewness,kurtosis,iqr
```

Figure 9: Top. Depiction of relationship between data_management and data_explainers folders. The PhDM_controller.py script from within data_management calls a number of data analysis functions from within the data_explainers folder. Bottom. A subset of functions from PhDM_controller.py is shown in the datSetReqCheck() function (that pertains to dataset requirements evaluations, D2.1). Top right. The PhDM plot from Figure 7.

3.3.2 Learning Management

Learning Management (Figure 10, originally from D2.1), within the *aifsm_phases->learning_management* folder the PhLM_controller.py script calls a number of functions to support steps/activities within this phase, e.g. to evaluate: i) model training performance: as a function of timing, accuracy (and other measures, e.g. mAP for object detectors), explainability. Application of XAI relates to:

- **Model Design** ->
  - *Explainability by design*: support to modify the DL network with specific XAI techniques. Currently supported techniques are: GradCAM, eigenCAM, activation pattern extractor, part/object decision tree surrogate model. Planned next steps include: utilizing techiques for disentanglement of latent space w.r.t. concepts or input parameters (from ODD/scenarios).
  - *Metaheuristic search* using feedback from model training and evaluation allowing for decisions on: i) what is the best model, e.g. yolo7, yolo8, ii) which is the best hyperparameterization of that model.
- **Model Training** ->
  - *Feature importance*: LIME[23]/SHAP[18]
  - *Saliency maps*: GradCAM[24], EigenCAM[25]
  - *Layerwise Backpropagation*: LRP[26], CRP[27] (Concept Relevance Propagation)
  - *Metaheuristic search*: see above
  - *Prototype/Concept based surrogate models*: Decision tree used for evaluating model predictions w.r.t. labelled prototypes and concepts, generated by XAI algorithms.
  - *Neuron activation pattern extractor*: to evaluate properties of specific layers, e.g for neural coverage or relevance to model predictions.
  - *Anomaly detection*: VAE[28], GANomaly[29], trained on activation patterns.

- **Model Evaluation** -> XAI is used in relation to hyperparameter selection (learning rate, etc) - design - should be made that best combine timing, performance/accuracy, explainability - in relation to the training data for the trained model. Explainability techniques are used with respect to the trained models.
- **Model Verification** -> XAI is used on test/verification data to evaluate timing, accuracy, explainability.

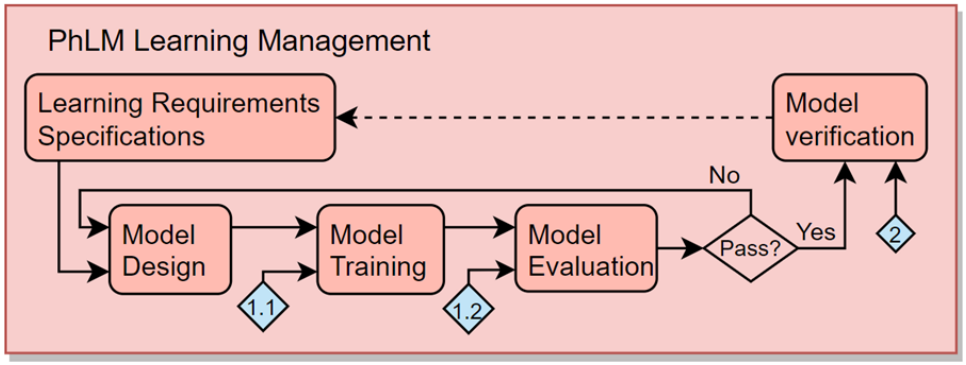


Figure 10: Learning Management phase (PhLM) (from D2.1). The sub-processes "Model Design", "Model Training", "Model Evaluation" and "Model Verification" are all (potentially) subjected to traditional evaluation techniques (e.g. convergence performance during training) or modern "XAI" based algorithms/methods for evaluating explainability. The usage of XAI may depend on model used and user preference, e.g. "Model Training" may not always utilize explainer algorithms to evaluate a model that hasn't been "fully" (to stopping criteria) trained.

Consistent with the sub-phases depicted in Figure 10, the implementation of PhLM - Learning Management within EXPLib entails model design, training/evaluation, and verification. The runnable colab script aifsm_eval.ipynb allows for model design in reference to a user selecting a model architecture of choice (not necessarily informed by any design criteria or iterative verification from this phase, at the current stage of development). Within the script it is also possible to exemplify actual training execution of models: (i) the target DL model, and (ii) explainer models, in the current state of development, this concerns the use of a variational autoencoder (VAE) that can be trained on the same data as the dependable DL component and used as data descriptors (PhDM) and data/model anomaly detection (OM). The reader should consult section 3.4.1 for a more detailed discussion.

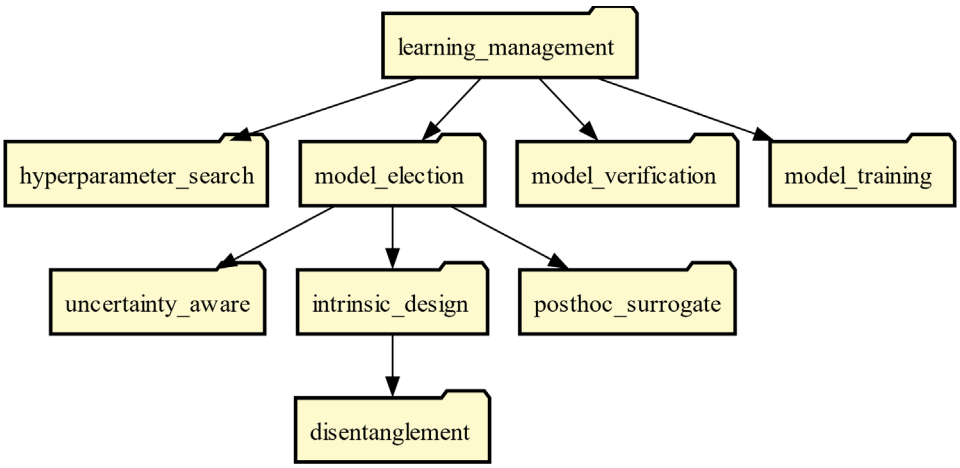


Figure 11: learning_management sub-folder.

Figure 11 depicts the directory sub-structure for the *learning_management* folder.

The XAI functions relevant to these phases are callable within the PhLM_controller.py script accessible from within the *aifsm_phases->learning_management* path. Figure 12 captures the relationship within EXPLib between the paths to *learning_management* and *model_explainers*.

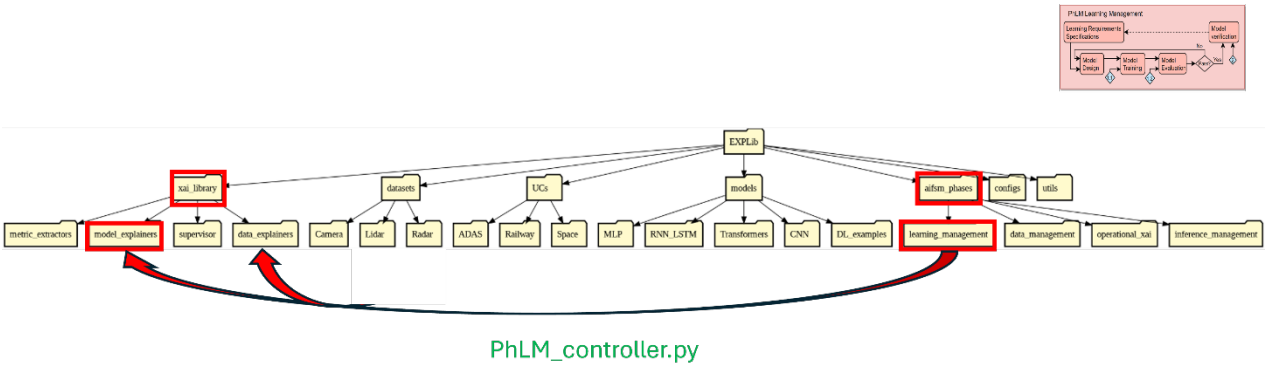


Figure 12: Depiction of relationship between learning_management and model_explainers folders. The PhLM_controller.py script from within learning_management calls a number of data analyses functions from within the model_explainers (and also data_explainers) folder. Top right. The PhLM plot from Figure 10.

Model explainers and data explainers (through reference to the explainable properties of the data with which the models were trained) are used to support this phase's activities as following (see Figure 12) – more description provided in section 3.4.1:

- *Model design*: Different techniques to add explainability to the target DL model. Available techniques that can be leveraged include: Saliency maps, surrogate models, feature disentanglement methods.
- *Model training/validation*: Decomposition approach (neural components, e.g. layers, frozen to assign different sub-problems to different parts of the model), disentanglement approach (semantic latent space), using data explainers to support dataset mixtures and correlation with model performance w.r.t. different performance metrics (e.g. mixture between real world data, synthetic data, and rejected data from the runtime inference)
- *Model verification*: Measure the performance degradation of the model when working with generalized datasets. Report the correlation between performance degradation and dataset specification.

3.3.3 Inference Management

Inference Management (Figure 13, originally from D2.1), within the *aifsm_phases->inference_management* folder the PhIM_controller.py script calls a number of functions relevant to preparation for deployment:

- **Model conversion**: Evaluation of fidelity of quantized models (parametrically and architecturally simpler models) for assessing potential for use in deployment.
- **Model optimization**: Use of techniques such as pruning, layerwise ablation, to reduce redundant architectural complexity.

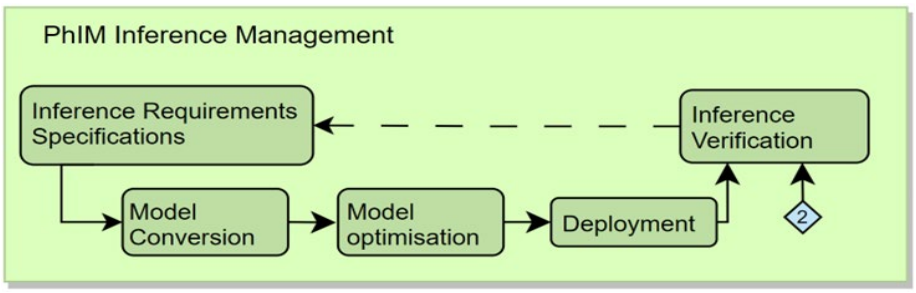


Figure 13: PhIM (from "PhIMG0003_Inference_Management_guideline" –D2.1). The Data Management phase

The implementation of Inference Management phase (or PhIM) within EXPLib, consistent with Figure 13, entails implementation of algorithms for the following applicable sub-phases:

- *Model conversion*: requires evaluation of performance of a simpler (quantized) model with respect to the originally trained model, e.g. evaluation of Yolo7-tiny as compared to Yolo7 (standard) to assess the effects of architectural simplification (and parameter reduction) on performance. This is an important pre-deployment step as smaller models allow for greater inference (real time) speeds necessary for use on the Use Cases.
- *Model optimisation*: entails use of post hoc techniques such as pruning, enabled through algorithms concerning neural coverage, such that redundant parts of the model can be ablated.
- *Inference verification*: At this stage model performance and explainability can be referenced with respect to the requirements relevant to the Use Cases.

It can be argued that model conversion and optimisation require recourse to XAI approaches and so the details of EXPLib implementation are provided in section 3.4. The PhIM_controller.py script contained within *aifsm_phases->inference_management* calls XAI functions from within the *model_explainers* folder (as used for PhLM). The current folder structure within EXPLib is depicted in Figure 14. The *scenario_generator* folder will be populated by search algorithms that permit a large variety of relevant scenarios to be tested. The *tradeoff* folder will contain supporting tools to facilitate implementations of tradeoff mechanism in Operation and Monitoring stage. The "trade-off" will be with computed respect to three dimensions: i) model performance (accuracy), ii) execution time, and iii) explainability.

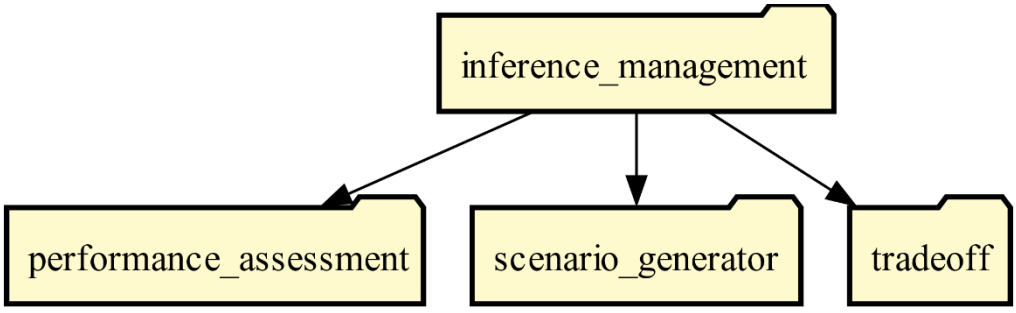


Figure 14: Inference_management sub-folder

3.3.4 Operational XAI

Operational XAI (Figure 15) is the interface between EXPLib and DLLib where relevant algorithm candidates required to build the reference architecture patterns (D2.2) shall be provided.

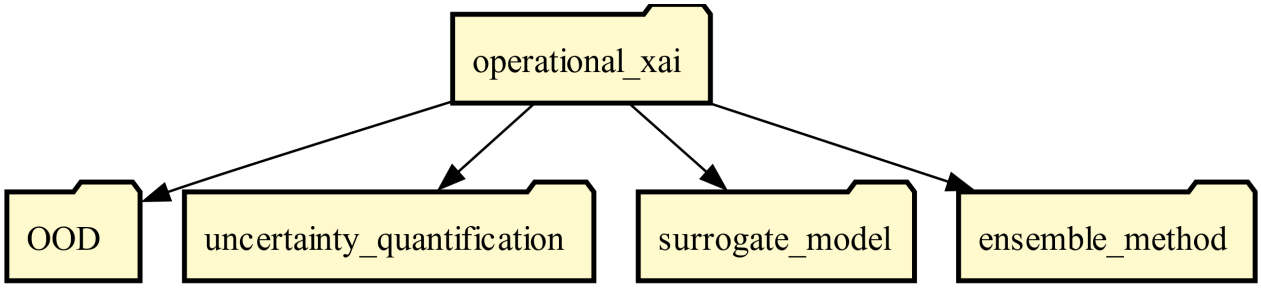


Figure 15: Operational XAI sublibrary

With reference to D3.1, Section 4, we have stored within the EXPLib methods for OOD detection, including measures of distance between models' attention maps that have been subjected to variations of input patterns (heatmapdistancefolder.py). Other folders contained within this path are to contain saved models for anomaly detection such as the VAE trained within the *learning_management* sub-directory. The use of trained surrogate models and uncertainty estimators is planned for work beyond M18. The key supervisory monitory components are summarized thus:

- *Out of Distribution (OOD) detector*. Here we focus on realizing algorithms that work with input images. The OOD detector can be used for monitoring anomalies directly from input image data or can also be used to monitor anomalies from an interim layer's neuron activation pattern. In the latter case, a function to extract activation patterns from a specific layer of the DL network is used. Scripts pertaining to the following used for OOD detection are contained within the *operational_xai->OOD* path:
  - Autoencoder: where VAE is preferred given its latent distribution modelling properties as compared to "vanilla" autoencoders that model pointwise latent space,
  - Likelihood Ratios[30]: Likehood ratio provides a statistical measure for hypothesis testing. The test will derive likelihood ratio of an observed data point against two hypotheses. In the case of the supervisor, the hypotheses are for normal and anomalous conditions.
- *Uncertainty estimator*: This subfolder contains the modified version of the original DL component to provide uncertainty estimates in addition to the prediction. A simple example of the result of applying dropout to achieve a Bayesian model for an MLP is provided in Figure 16. This is also planned for YOLO/SSD implementation after M18. However, we will have to decide if/how uncertainty techniques shall be used for the ORIN hardware, provided the first observation of time/resource consumption of Bayesian dropout approaches. A promising approach may be to train the target DL model to provide output as distribution parameters (e.g. means and standard deviations of GMM).
- *Surrogate model*: A trained surrogate simple and intrinsic model that approximates the output prediction of the target DL component. Within this library, a simple Decision Tree based model that approximates the object detection (boundingbox, scores) from detected object's parts (boundingbox, scores) is exemplified (Figure 18).
- *Decision function*: Using different ensemble methods. This component will analyse outputs from DL model(s) and supervisors and provide consolidated output of (prediction,

accept/reject, trustworthiness score). As of this point in time, soft voting method and bagging methods (Random Forest) are available but not yet adapted to YOLO/SSD models.

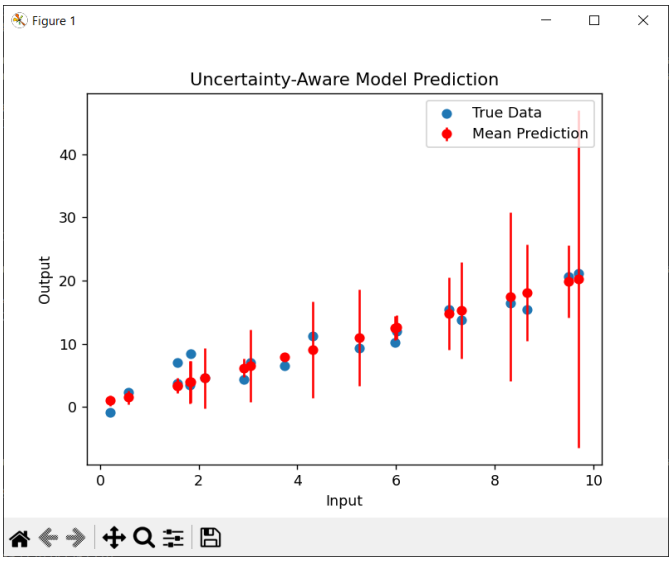


Figure 16: Example results of Bayesian uncertainty aware model. The example model is a simple MLP regression network

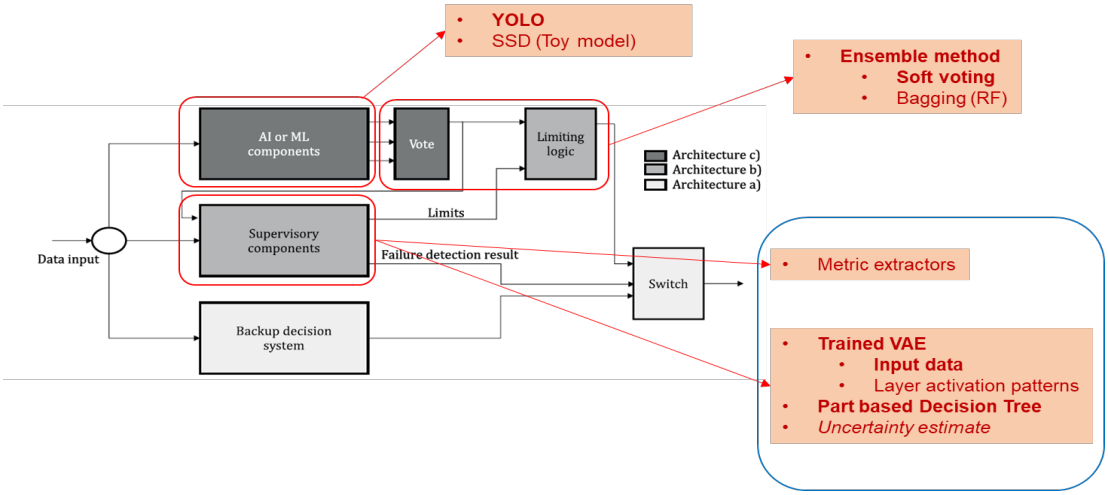


Figure 17: Mapping Operational XAI components in EXPLib and ISO/IEC TR 5469 safety architecture

Figure 17 depicts the mappings between the components within this library folder (where VAE provides the OOD detector) and the reference safety architecture (by ISO/IEC TR5469[31], which is also adopted by D2.2[3])

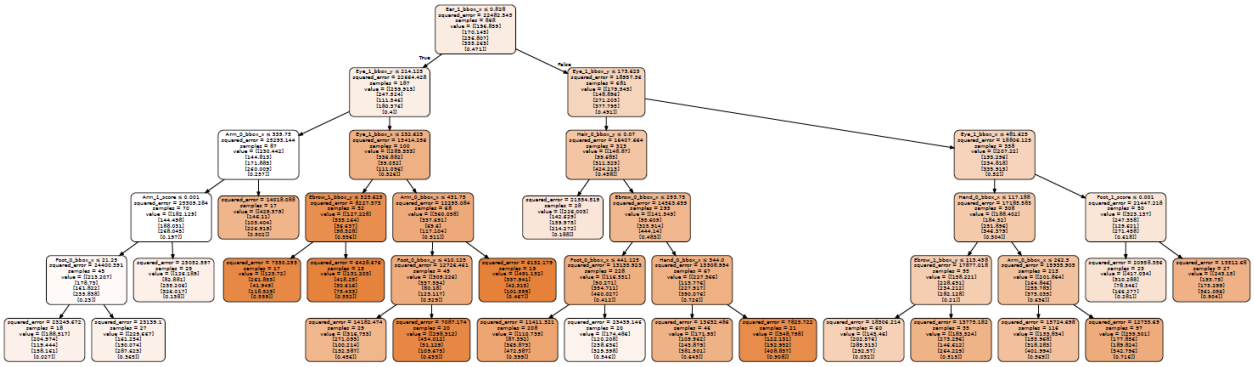


Figure 18: Example decision tree surrogate model approximating a trained Yolov7 model bodyparts/person detection relation on the PASCAL-VOC dataset

3.4 XAI library

In this section we present the rationale behind, and content of, the folders within the EXPLib that comprise the XAI library. Explanation (XAI) is required at each of the AI-FSM phases: PhDM (Data Management phase), PhLM (Learning Management phase), PhIM (Inference Management phase).

In reference to Figure 20 (the relevant sub-part of which being presented in abstraction in Figure 19), in order to assure the dependability of the DL component, explainer algorithms are required to explain: i) quality of the data ("Data explainers"), ii) the quality of the model ("Model explainers"). In turn, a (Safety) supervisor component is required to arbitrate whether the DL component can be relied upon when making its predictions in OM stage. The (multiple) supervisory checks that this component makes concern possible weaknesses of the data or of the model itself.

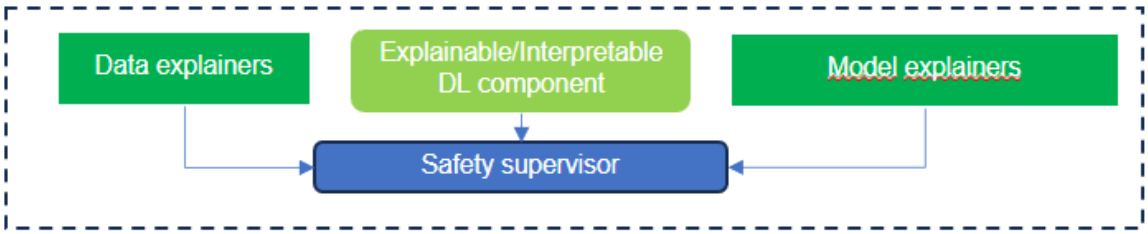


Figure 19: Critical XAI components required for assuring dependability of the DL component.

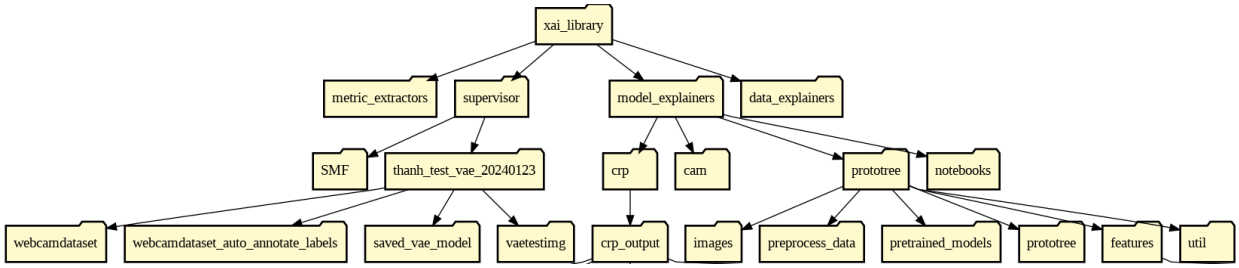


Figure 20: Sub-directory for the xai_library that implements: i) data explainers, ii) model explainers, iii) supervisor component, and additionally iv) metric extractors.

In Figure 20 is visualized the sub-directory of EXPLib that concerns the contents of the XAI library. The contents of the 1st level folders within this sub-directory can be summarized as follows:

- *xai_library->data_explainers* contains a number of scripts for data profiling and data prototyping. The folder also contains a script data_explainers.py that allows to call a number of functions for data profiling and data prototyping.
- *xai_library->model_explainers* contains folders labelled according to a number of XAI algorithms used (shown above are *cam* and *crp* (concept relevance propagation), other algorithms (not shown here due to limitations in space) include:
    - *DT* (decision tree algorithm used as an interpretable surrogate), callable within the colab (aifsm_eval.ipynb) runnable script,
    - *shap* (kernel shap used as a tool for computing shap values with respect to classes of objects contained in selected bounding boxes within a given image), callable within the colab (aifsm_eval.ipynb) runnable script,
    - *torchBNN* (concerns use of bayesian models to evaluate uncertainty in relation to stochastic output of layers – including final layer see D3.1 for more description.
- *xai_library->metric_extractors* currently contains functions for assessing similarity (divergence), distance and correlations between different versions of the same model, e.g. for evaluating consistency of outputs given inputs with respect to changes in the model.
- *xai_library->supervisor* contains a number of functions relevant to detection of corner cases and anomalies regarding OOD data. These functions include: pytorchvae.py and likelihood_ratios.py. Currently, the walkthrough colab script calls the VAE train function in through a call to a function in the *learning_management* folder. The trained model is saved both in *xai_library->supervisor->anomaly_detector->saved_vae_model* path but also the *operational_xai->OOD->saved_vae_model* path that is designed to provide the final model for C library conversion (see Figure 4). Also within *xai_library->supervisor->uncertainty_estimator* is the script inference_dropout.py is contained. Surrogate models will also be contained here. The supervisor component has been tested on successive camera frame data to evaluate anomalous inserted frames within the data stream.

In the following subsections we describe the content investigated within the EXPLib referenced in Figure 19 for each component.

3.4.1 Data explainers

The main role of data explanation is to provide a means to assess whether: i) data requirements specifications have been met, and ii) to provide a summary (log) to a certification authority that provides appropriate explanation of the datasets and assurance of their quality. This can be achieved through *data profiling* and *data prototyping* methods.

Models are only as good as the data they are trained on. Explanation of DL components cannot just be limited to the (deep) learning models. Model analysis also requires accounting for the dataset used to train and validate the model. If the dataset is insufficient, e.g. too small, outside the ODD, insufficiently captures the ODD, filled with corrupted data or unbalanced in relation to classes and feature space covered, then the (deep) learning models will fail and may provide the reasons (uncovered by XAI algorithms at the PhLM stage) why the trained model fails. Therefore, explanation is required not just in the model evaluation stages (learning and inference management) but also in relation to the data. We break down some of the methods we have investigated so far (both those callable within the colab script and those that are currently not callable but reside within the EXPLib and will soon be interfaced with the runnable script) in relation to data profiling and data prototyping methods.

Data profiling: This includes simple methods such as evaluating whether data is corrupted or is missing labels or contains incorrect labels. We have included within the EXPLib simple functions for computing these properties of the dataset and are callable through the data_explainers.py script that in turn is called by the PhDM_controller.py script. We do not consider such profiling methods representative of XAI approaches though. Here, instead we present the use of exploratory data analysis (EDA) that is currently present within the EXPLib.

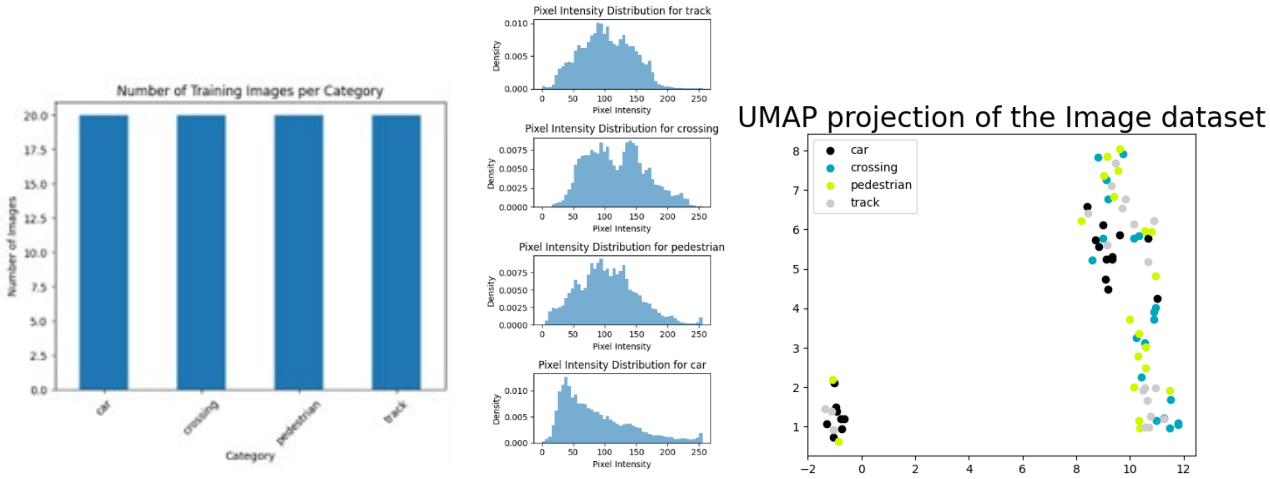


Figure 21: Outputs of EDA for the rs19_short dataset when running PhDM_controller.py script. This script is callable from the aifsm_eval.ipynb script, which calls data profiling functions from within the PhDM_controller.py script. Left. Class distribution analysis. Centre. Pixel Intenstiy Analysis. Right. UMAP Analysis.

EDA functions are callable from within the PhDM_controller.py script. In Figure 21 shows the output of analysis of four classes of data within *rs19_short* dataset: car, crossing, pedestrian, track. The left most figure clarifies that the class distribution is uniform and not biased. The pixel intensity analysis, on the other hand, shows a skewed distribution of pixel intensities for the car class relative to the other classes. The relatively lower pixel intensity values for this class of data might lead to difficulties in model predictions for this class of data in as smaller intensity inputs can lead to smaller gradient values with which to update the weights (feature representations) for these pixels. The developer is therefore informed that normalization techniques would be useful here to guard against bias of class representation. In the right most figure the plot of UMAP (Uniform Manifold Approximation and Projection for Dimension Reduction) [32] (preferable to PCA at it captures better the complex non-linear relationships that can exist in the unstructured data) indicates that car images form distinct clusters of two types and are mostly separated from the other classes of data. Crossing and pedestrian classes overlap, which could be explained by the fact that pedestrian images are often beside or on crossings. Track images overlap somewhat with.

3.4.2 Model explainers

A number of types of model explainers are found in the model_explainers folder some of which are indirectly callable from the aifsm_eval.ipynb colab script other are planned to be callable from March 31st. The approach to XAI is broken down in D3.1. into use of global explainers (where explanations for how models represent features of the dataset are provided in relation to the trained weights of the DL model), local explainers (where explanations for model predictions are provided in reference to single data instances); we further add the notion of glocal explainers (see [17]) concerning XAI algorithms that offer both global and local explanations. Furthering the

description provided in 3.3.5.6 of D3.1. evaluation of the dependability of the DL component should require both global and local explanations.

3.4.2.1 Global explainers

Global explainers provide insights into how models (including Deep Learning based models) represent information (features) of the data as well as the non-linear hierarchical structure that permits nonlinear combinations of represented features. Below we briefly summarize pros and cons of the different approaches before describing how they have been used so far within the EXPLib.

Pros:

- *Identifying vulnerabilities*: Understanding the holistic model structure is rather critical in terms of accounting for potential vulnerabilities of the model as well as the potential effects of data instances that the model has not been trained on/exposed to;
- *Optimization*: Provides insights into the model structure and how features are represented -> can be used to separate classes of data and help understand the extent to which such classes are separable (within the model)
- *Quantifiable*: Feature importance rankings can enable stakeholders (e.g. developers) to assess the quality and relevance of the model's feature representations.
- *Model validation*: Feature representation can be compared against domain expert knowledge to validate whether the model is making predictions based on appropriate feature representation and hierarchical decomposition.

Cons:

- *Complexity*: Explanations may be overly complex (this is also a potential feature of local explanations) and consequently hard to interpret, e.g. if a decision tree output has many branchings it can be challenging to interpret (see [33])
- *Non-specific explanations*: Doesn't provide explanations for individual instances (e.g. DeepDream, kernel activation maximization approaches, may show which types of input can activate parts of the network but these inputs may never have been used in training - hallucinations)

Examples of the global methods that have been contained/investigated within EXPLib:

- *Kernel activation maximization* is a post hoc method and used in relation to the VGG16_BN[11] image classifier utilized in the functions contained within the *xai_libraries->model_explainers->crp* path, it is a gradient method that optimizes pixel values of an initially noisy input image in relation to a selected feature map or layer within a CNN.
- *Feature importance*: Techniques such as input dilation/permutation can be used (SHAP[18], LIME[23]) to provide global importance or input features with regards to a model.
- *Partial Dependence Plots (PDP*[34]): show the dependence between the target response and a set of input features.
- *Accumulated Local Effects (ALE*[35]*):* describe how features influence the prediction of a machine learning model on average.
- *Prototree*[12] (referred to in D3.1., Section 2.4.2) as an intrinsically explainable model that computes predictions based on extractred prototypes through a ResNet based feature extractor. This can be found in the *xai_libraries->model_explainers->prototree* path.

3.4.2.2 Local explainers

Local explainer XAI algorithms are necessary to provide explanations for specific data instances, which might include corner cases, i.e. examples of data that is OOD. Briefly summarizing Local explainers might be broken down into pros and cons as follows:

Pros:

- *Detailed explanations*: for model predictions for specific data inputs -> can be used in "run time" for predictions for a given data instance (e.g. video frame).
- *Optimization*: Developers can utilize explanations for OOD or anomalous data and consider how to better design or train models or otherwise utilize different distributions of datasets, or augment data for allowing for the model to better cater for such data.
- *Can be Model Agnostic*: Unlike global methods that entail making transparent "under-the-hood" representations of features within the black box model (DL component), local explainers can be model agnostic offering explanations for input-output mappings that do not depend on the internal workings of the model.

Cons:

- *Lack global context*: absence of the global explanation for which features are critical for predicting data (including classes of data), holistic context (how the model represents features) may be lost,
- *Interpretability*: XAI approaches that use saliency or heat maps may be subject to interpretation, the visualizations may provide fast intuitive explanations but may come at the expense of accuracy (see [33]).
- *Inconsistency (within and between algorithms)*: When based on saliency/heat maps different runs may yield different results (different saliency/heat maps) for some algorithms. Conversely, different saliency map generating local XAI algorithms might provide different explanations.
- *Overfitting*: Some XAI algorithms, e.g. those based on gradient methods, risk overfitting feature explanations in the images where the features may show activation in relation to the model prediction (correlative) but may not be relevant to the prediction (causal).

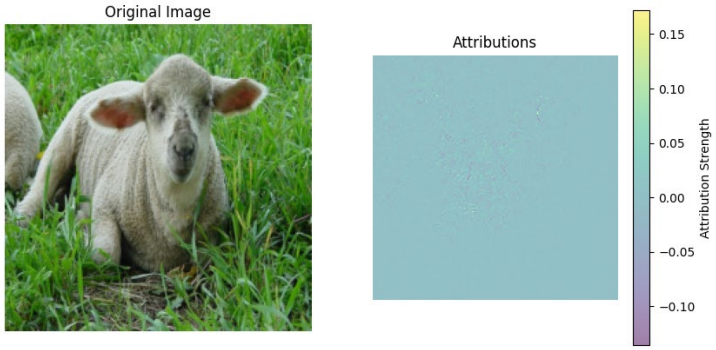


Figure 22: Example attribution map generated from LRP local explainer method (PASCAL VOC sample image)

Figure 22 illustrates an example of a model local explainer (using LRP).The left subfigure is an image from PASCAL_VOC dataset, and the right subfigure shows pixel-wise feature relevance for a classification model prediction (ResNet50).

Examples of the local methods contained/investigated within EXPLlib:

- *LIME* – as a post hoc model agnostic local explainer (referenced in D3.1. Section 2.3.2.2) we have tested this method but currently it is not runnable using the colab script. These files are not yet callable within the aifsm_eval.ipynb script.
- *Grad-CAM* (and other versions of CAM) – another post hoc gradient method described in Section 2.3.2.2 of D3.1. The CAM .py files can be found in xai_library->model_explainers->cam. These files are not yet callable within the aifsm_eval.ipynb script.
- *Anchors*[36]*:* using standard image segmentation methods available from scikit-image lib (felxenszwalb, slic...) for selecting super-pixel segmented area from the input sample data (image). The function takes input parameters as minimum anchor precision threshold (default value 0.95), multi-armed bandit parameter used for searching the best candidate anchor from a list of proposed anchors at each iteration. The function will output anchor rules as if-then rules using absence or presence of each segmented superpixels.

3.4.2.3 Glocal explainers

The term "Glocal" refers to XAI approaches that allow for both local and global explanations (e.g. [17]). This approach is aimed at addressing the limitations or purely global or purely local approaches by permitting: i) understanding of the structure of the learned model that permits understanding of what features are represented; ii) providing explanations for the specific data instances.

Many XAI models that have a "glocal" focus seek to explain model predictions on single data instances (local) through extracting high-level human-interpretable features represented in the model (global). This approach is sometimes referred to as "disentangling" the higher-level features that the model is learning from the artefactual elements that are represented but have no predictive value ("relevance").

Examples contained/investigated within EXPLib:

- *Shap/Kernel Shap*: method for evaluating the value of individual features and their interdependence with other features in model predictions (referred to in D3.1, Section 2.3.2.2). Kernel shap allows for class predictions (shap values) to be made within the bounding boxes of Object Detectors such as Yolo5, Yolo7. The path to the shap functions can be found in *xai_library->model_explainers->shap* in the shap_yolo.ipynb file (using colab functions for accessing and other colab functions. These functions are callable, therefore, within our aifsm_eval.ipynb. An example of usage can be seen in Figure 26. The shap_yolo.ipynb script adapts code from https://www.steadforce.com/how-tos/explainable-object-detection.

- *CRP/L-CRP*: Concept Relevance Propagation and Localized-Concept Relevance Propagation are gradient based approaches based on the Layerwise Relevance Propagation algorithm (LRP). CRP[27] allows for learning the relevance of disentangled "concepts" (supra-threshold relevance values in feature maps) to model predictions; L-CRP[17] takes this concept further by applying concept disentanglement in relation to class predictions within bounded boxes. The path to the CRP functions can be found in *xai_library->model_explainers->crp ,* which also links to a colab/jupyter script crp.ipynb callable from within our aifsm_eval.ipynb script and currently runnable with the VGG16_bn model.

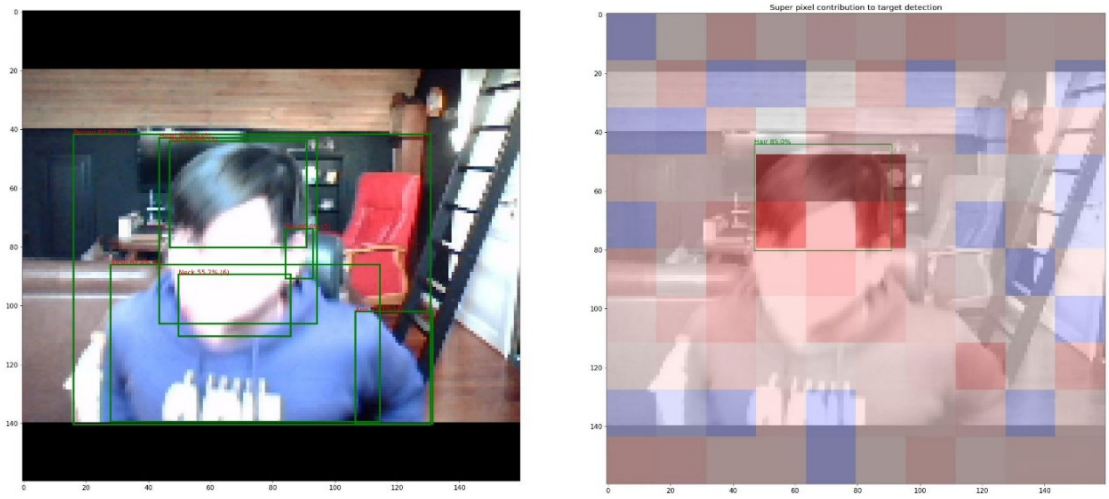


Figure 23: Using Kernel-Shap to provide shap values for selected bounding box with respect to the selected class (in this case "hair"). This example model explainer is used on a trained Yolo7 model with a customized dataset.

From the Kernel Shap example in Figure 23, we can see that shap values (red pixels) are highest for the selected object (hair) that classified by Yolo7 (trained on the coco2017 dataset) within the bounding box and pixels that are not human related are generally negatively values (blue pixels).

An example of CRP working with the VGG16_bn (CNN) model is given in Figure 24. This shows heatmaps for the most relevant concepts for feature map 1 of layer 40 of this CNN. Red pixels

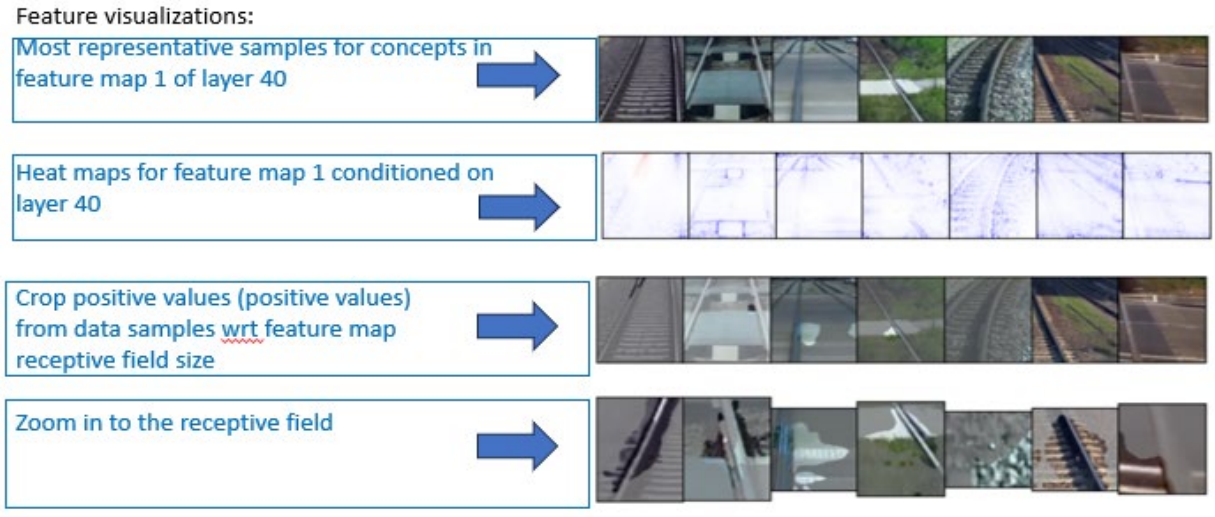


Figure 24: Concept Relevance Propagation example with cropped version of Railsem19 (rs19_short) dataset [34]

indicate positive values with respect to the class prediction “crossing” and blue pixels provide negative values. The values below a certain (positive) threshold are masked and original image segment is overlayed on a faded version of the original image with “zoom in” over the segment for visibility. The combination track-part and rail-part concepts might reflect the fact that the crossing and track features tend to overlap in the dataset (as identified by the UMAP analysis shown in Figure 21). The crp code is adapted from: https://github.com/rachtibat/zennit-crp.

The zoomed in segments of Figure 24 show “what” is being encoded of the image, i.e. the concept, and in this sense considered a global property by the authors. Examples here concern heatmaps showing “where” are the relevant parts of the image for model prediction, and in this sense considered a local property by the authors. This combination of data point specific location of features relevant to prediction and local-independent human interpretable concepts are what make the approach “glocal”.

3.4.3 Metric extractors

3.4.3.1 Explainability and KPI metrics

The list of explainable metrics and KPIs that will finally be implemented and tested within the EXPLib can be found in D3.1., section 3.5.1. Currently XAI metrics are implemented in terms of distance metrics between models using different input (as mentioned in Section 3.4) using the heatmapdistancefolder.py file found in *xai_library->model_explainers->metric_extractors*. We also use VAE for comparing reconstruction error performance (RMSE) for the training (validation) dataset against the verification dataset (used in the DLLib and potentially for PhIM – not currently callable within our aifsm_eval.ipynb script). If the reconstruction error for the verification dataset is lower than for the training dataset beyond a certain threshold, it can be considered anomalous. We will also compare models for explanation performance using distance measures (e.g. euclidean, cosine) in relation to original trained models (e.g. Yolo7) and quantized versions (e.g. Yolo7-tiny) or pruned versions in order to see if shap values (or other such saliency map based metrics) significantly differ with respect to given data points. An algorithm Shap-GAP already exists for this purpose ([5]).

3.4.3.2 DL component metrics

The list of candidate DL component explainability metrics that will finally be implemented and tested within the EXPLib can be found in D3.1., section 3.5.2. The output of Yolo7 trainable using the aifsm_eval.ipynb script (albeit with the requirement for the model to be trained in colab but off google Drive (seemingly for access reasons) and then to save the trained model and KPI/metrics into the relevant folder. This folder contains the saved weights of the model and standard metrics concerning Precision-Recall curves, F1 curve, mAP, confusion matrix (over all 80 data classes). Currently for the VGG16_bn only stored are graphs plotting training performance with respect to training and validation error on loss and accuracy over epochs where stopping criteria was selected for number of epochs rather than successive increases of validation error. Training of the VAE is stopped, however, according to six increasing validation losses (to guard against overfitting to the training data).

3.4.3.3 Structural coverage metrics

The following metric extractors are available (reader should consult D3.1 for more detailed descriptions): NC, NBC, SNAC, KMNC, TKNC, DC.

Structural coverage metrics implemented within the library, EXPLib.xai_library.metric_extractors.structural_coverage include:

NeuronCoverage class (NC): Activated neurons are selected with threshold 0.5, and the ratio of number of activated neurons/total number of neurons in a specific selected layer can be considered as the NC. The metric in this case is specific to a data sample as input. When applying this metric extractors for all data samples in the "known" datasets and combine the activation neuron-wise, we will achieve the overall NC, and can also log the neuron activation patterns for later used with anomaly detector in OM.

NeuronBoundaryCoverage: Computes the boundary coverage of neurons in a model. The difference between the two captured neuron activations with regards to two different sets of input data will be computed as the metric measures.

StrongNeuronActivationCoverage: Similar to NC, but this metric only considers strongly activated neurons in the coverage (e.g. the default threshold used is 0.5)

KMultisectionNeuronCoverage: Measures neuron coverage distributed across multi-section defined by an input parameter (default 10 sections). The coverage accounts for neurons that activated across selected sections taken an input data sample.

TopKNeuronCoverage: Measures coverage by passing an input data sample through the model and sorting the neuron activations across layers, then selected top-k activated neurons (top-k is defined by input parameter, default to 20). The coverage is defined as number of unique top-k activated neurons/total number of neurons.

DecisionCoverage: Passing in data samples and compute the number of unique decisions as the coverage.

3.4.4 Supervisory monitor

A supervisory monitor (also referred to as a supervisor) is a software component that oversees the predictions made by the DL components and estimates the trustworthiness scores, i.e. if the predictions are reliable enough for safety critical decisions to be based on. For shorter terms, a supervisory monitor is also referred to as a supervisor or a monitor.

Within the safety patterns architecture, supervisory monitors are key subcomponents of the "L1-Diagnosis and Monitoring" component in safety patterns architecture of D2.2.

The validated python supervisory monitor models will be ported into low level C-based libraries for use in the NVIDIA Jetson AGX Orin platform (DLLib Supervisory Monitor).

An approach to implementing supervisory monitor is through a multiple-supervisor approach (see Figure 25), which also uses decision function (ensemble method algorithms) to assess whether DL component predictions should be accepted or rejected, e.g. through analysis of different supervisor outputs as well as different DL components' predictions (in case several DL components are used in redundant architecture setup).

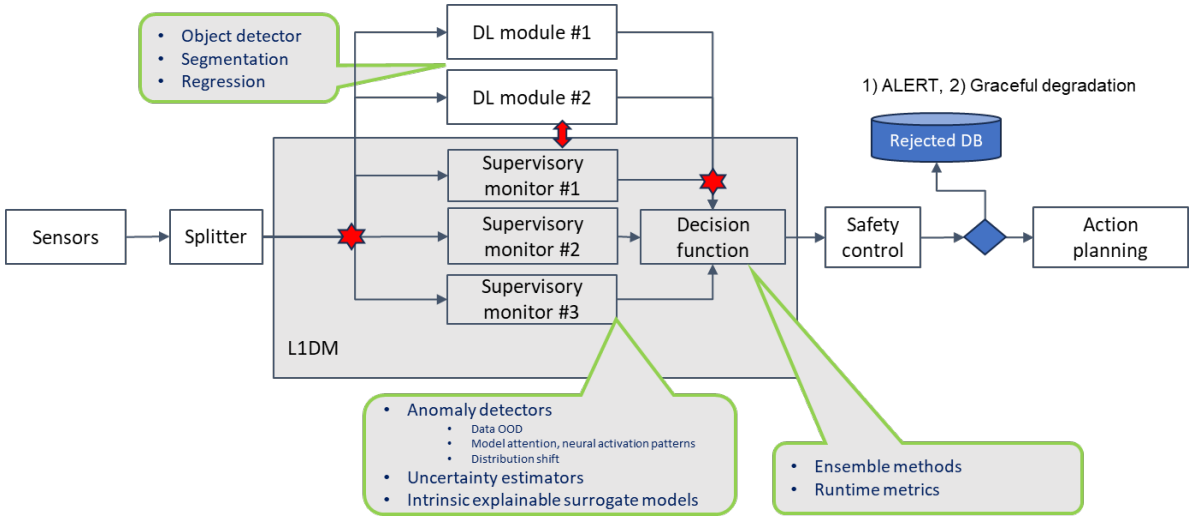


Figure 25: Multiple supervisory monitors architecture to support L1DM

By combining supervisory monitors with different failure detection competences, a range of potential monitored diagnostic faults in the DL components may be covered. The specific ensemble of these particular faults though may differ according to the Use Case. In our approach (implementing within EXPLib) we focus on three classes of supervisory monitor algorithms: i) Anomaly detector or OOD detector, ii) Uncertainty estimator, and iii) Surrogate models.

3.4.4.1 Anomaly detector

This supervisor monitors out-of-distribution (OOD) propabilities for:

- *Input* (comparing input data against the expected distribution of known datasets)
- *Model behaviour* (comparing layerwise neuron activation patterns against known pattern distribution, logged during the PhLM)
- *Output* (detect distributional shift, e.g. using timeseries of output predictions)

A candidate anomaly detector for use in deployment is a variational autoencoder (VAE), which is particularly applicable to anomaly detection at the input data and any interim feature (represented by DL network layer activations) and is implemented within our EXPLib. This is depicted in Figure 26. The bottleneck of the VAE provides a latent space distribution (as opposed to single point representations in "vanilla" autoencoders) following data input. Sampling from this space allows for reconstructions that are "of the type" of the original input rather than exactly the same as it (depending on the weighting of the reconstruction error and Kullbach-Leibler loss terms). The advantage of using a VAE as opposed to the vanilla flavoured version is that the trained model is more likely to generalize to new data that fits the distribution and therefore provides a better indicator of OOD if reconstruction error is high than does the vanilla version.

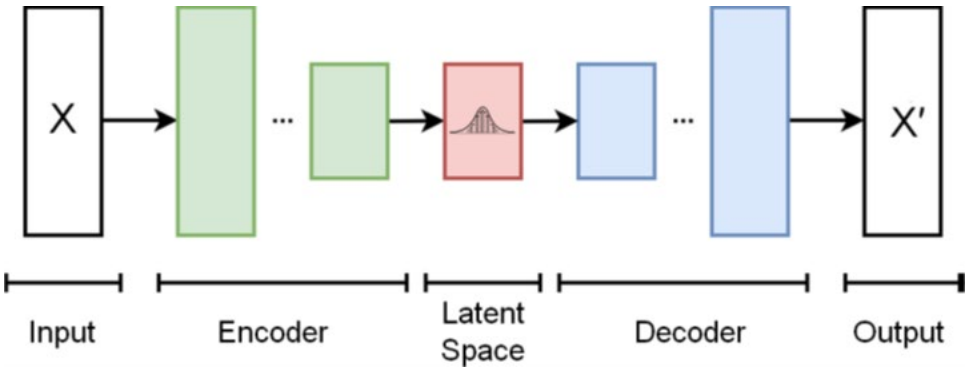


Figure 26: Variational autoencoder architecture for evaluation of OOD in deployment.

The autoencoder is trained on the datasets managed in PhDM. In the Inference stage (PhIM but also in Operation and Monitoring), the new data is evaluated as being OOD or not as a function of the reconstruction error (magnitude of output layer vector minus input layer vector) as compared to the reconstruction error produced using the original data. If a certain threshold is reached (deployment data reconstruction error higher than for original data, e.g. compared using Euclidean distance), the supervisor determines confidence in the prediction and whether the predictions should be trusted by the next module/user in the pipeline or not.

In Figure 26 are shown examples of types of anomaly detector methods using frames from a video sequence of a car on a motorway and where frames were inserted with properties that were considered OOD (at least in the sense that they were not consistent within the frame sequence). Certain anomaly detector supervisors may be more suited to certain data or monitoring tasks. VAEs might, for example, function better as detectors of images that differ in relation to a particular dimension of the data (in the example in Figure 27, the change in lighting is picked up by the detector). For certain other dimensions of data change, e.g. corruption of a subset of pixels as is shown in the third image from top, the optical flow detector appeared better at detecting the anomaly.

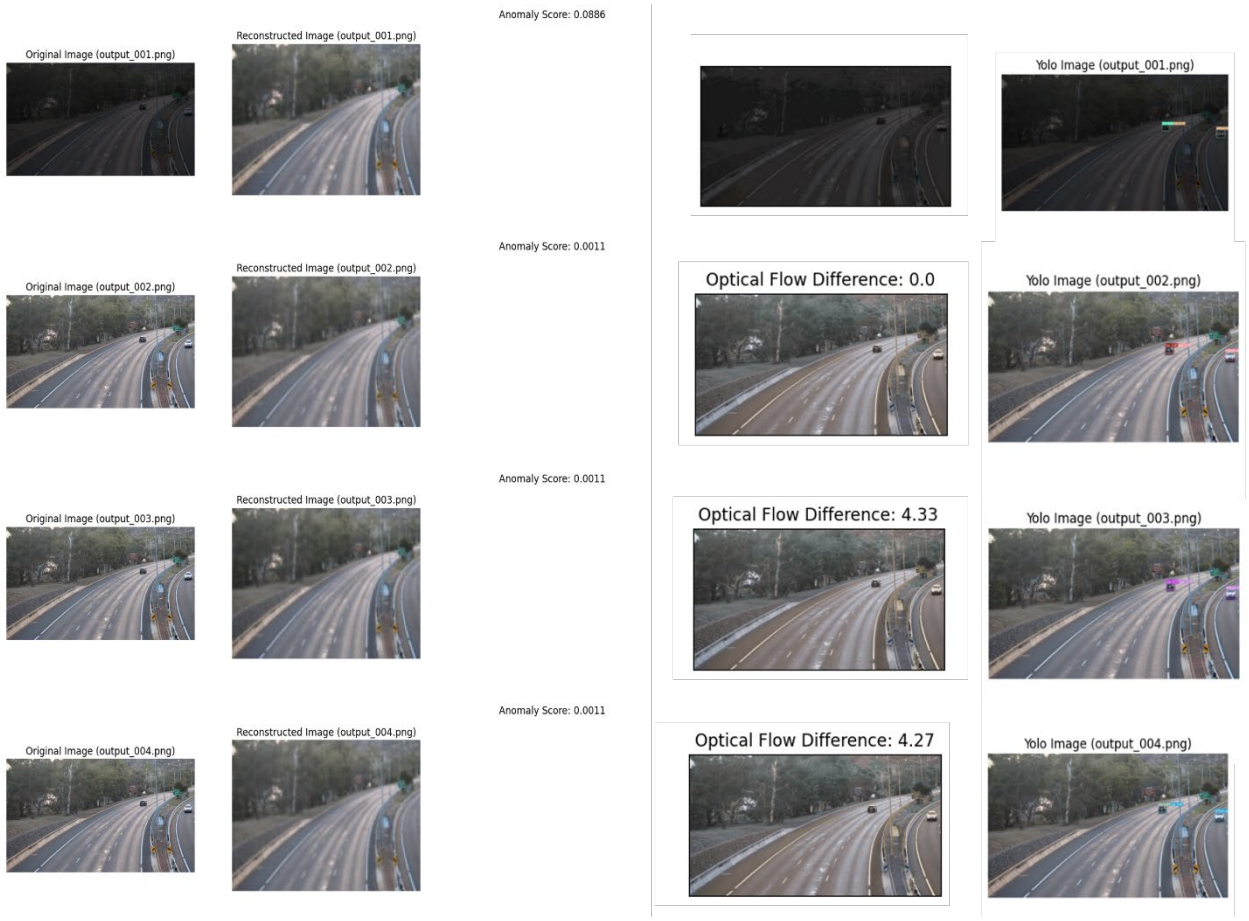


Figure 27: Use of supervisor functions for detecting anomalous inputs. Left side, the top images show the original image (doctored to be darker) and the reconstructed image (which is based on the statistics of the training data, in this case just the frames of the video sequence). As compared to the undoctored images below the anomaly detection is relatively high 0.0886 > 0.0011. Right side, an optical flow detector evaluates differences in the same sequence of images for inconsistencies anomalies.

Figure 28: Sequential video frame inputs to a pretrained Yolo7 model and to a trained Variational Autoencoder. Left. Original image inputs and VAE reconstructed image. Anomaly score by the VAE for each frame is provided in the centre of the image where it can be seen that the first image provides a larger score (as a result presumably of the edited change in luminosity of the frame). Right. Original images on the left side and Yoloy7 localization and classification on the right side with optical flow difference scores.

Note, the corrupted values (colouration of the car) in the 3rd frame from top in Figure 28 also leads to the Yolo7 model not being able to correctly locate and classify the object.

3.4.4.2 Surrogate models

Using a Decision Tree or Rule-based (Rule extractor) as a higher SIL supervisor to monitor the DL components provides another means to supervise output. An example of use of a surrogate model that is contained within EXPLib and callable via the aifsm_eval.ipynb script. In this example, body part images are used as inputs for a binary decision tree regression model (see Figure 18) to approximate the person detection (boundingbox, scores). The approximated person detection will then be compared with the detections from the DL model to assess the deviations of model behaviours (with regards to the logics embedded in the surrogate models, which have been verified during the AI-FSM lifecycle).

3.4.4.3 Uncertainty estimator and management

Uncertainty estimates shall be provided by the modified DL component itself.

Uncertainty management describes the rules used to combine results from different supervisors for the safe supervisor decisions (Accept/Reject), also referred to as decision function (see more in Section 3.3.4).

Currently contained within the EXPLib (in *xai_library->model_explainers->uncertainty_models*) are python codes for carrying out a number of available approaches that can be applied for DL model uncertainty – bayesianmodels.py and conformal_prediction.py. Reference to uncertainy

estimation and management is made in D3.1. Section 4.2. These methods are currently not callable within our aifsm_eval.ipynb script.

3.5 Runnable Walkthrough Example of EXPLib Usage

In this section is presented the current state of the EXPLib implementation in reference to i) tested components that are contained within the library, that function in a "stand alone" capacity, but are not necessarily currently accessible/integrated within the runnable progam, e.g. colab runnable script (aifsm_eval.ipnyb); ii) tested components that run within the colab script (aifsm_eval.ipnyb). Table 1 summarizes the AI-FSM stages that are represented top-to-bottom in a number of the cells of the aifsm_eval.ipnyb script (apart from the interface stages/cells that are presented within phase appropriate cells).

Table 1: Checklist of progress in implementation of the various stages of the AI-FSM and related XAI algorithms with respect to i) runnable within the implemented aifsm_eval.ipnyb script for particular AI-FSM phases/cells, ii) not yet implemented, iii) soon to iii) soon to be implemented, iv) runnable as a walkthrough example within the implemented aifsm_eval.ipnyb script.

Colab Stage	Sub-phases	Colab?	EXPLib supporting functions/tools
Interface (input)	Load Dataset	✓	Choose: **Pascal-Voc (W)**, Coco2017, etc.
	Load Model	✓	Choose: **Yolo7 (W)**, VGG16, Prototree, etc.
	XAI/Supervisor parts	×	Choose: VAE, Surrogate DT
PhDM	**1) Input/Output**:		
	• Read: Req.	✓	.json data validation schemas[37]
	• Write: Logging.	✓	.txt file with logs and validation results w.r.t. Reqs. .json file with measured dataset metrics
	2) Data Collection:		
	• file format	✓	**Logged (W)**
	• consistency of file format	✓	**Logged (W)**
	• file dataset	✓	**Logged (W)**
	• data volume	✓	Logged
	• data charac.	×	not evaluated currently
	3) Data Preparation:		
	• Labelling annotation	×	To update per dataset
	• Data augmentation	×	To update
	• Data cleaning	⏳	Use VAE to remove anomalies
	• Data pre-processing & normalization	⏳	Use Pixel Intensity Analysis and normalize intensities
	4) Data Verification: 4.1. Profiling		
	• Representativeness	⏳	Data explainer plots
	• Completeness	⏳	Checks on dataset dimensions, e.g. colour histogram/distribution over RGB range

Colab Stage	Sub-phases	Colab?	EXPLib supporting functions/tools
	• Balance	✓	**Distribution of classes per dataset (W)**,
		✓	**Distribution of features in dataset per class (pixel intensity analysis; W)**
		✓	Dimension reduction (PCA, tSNE, UMAP)
		✓	**Bounding box statistics (ODs, W)**
	• Accuracy	✓	Labelling error: Missing labels check
		×	Labelling error: Incorrect labels
		×	Other
	4.2. Data Prototyping		
	• Prototype sampling		ProtoDash
	• Prototype patch		**Prototype Clustering (W)**
PhLM	1) **Model Selection/Design**		
	1.1. Architecture selection	⏳	Domain-specific knowledge required; models can be tried out (Input)
	1.2. Initial hyperparameters	⏳	For each model – learning rate, optimizer, batch size, etc. should be logged
	1.3. Pre-trained vs Random initialization		
	• Pre-trained	✓	**Yolo7 (W)**, VGG16, VAE, Prototree
	• Trainable from scratch	✓	**Yolo7 (W), VAE (W)**, Prototree
	• Finetunable	✓	VGG16, VAE, Prototree
	1.4. Hyper-parameter tuning	×	Metaheuristic search
	2) Model Training		
	• Training models	✓	**VAE (W)**, Prototree
	• Finetuning models	✓	VAE, VGG16, Prototree
	• Stopping criteria	✓	**No. of epochs (W), no. of non-reducing loss validations – "patience" (W)**
	3) Model Evaluation		
	3.1. Overtraining checks	✓	VGG16, **VAE (W)**
	3.2.Performance (CM/mAP/other)	✓	**Yolo7 provides metrics (W)**
	3.3. XAI Investigation		
	• Transparency (decomposition)		**Decision tree approach – predicting wholes from parts (W);** Prototree (intrinsically explainable DT receives inputs from ResNet)
	• Anomaly detection checks	✓	VAE (on test dataset),

Colab Stage	Sub-phases	Colab?	EXPLib supporting functions/tools
	• Saliency maps (sparsity check)	✓	**Kernel-SHAP (Yolo7; W)** EigenCAM, GradCAM, LRP
	• Interpretability check	✓	Surrogate DT (update wrt layer checks and output predictions)
	• Complexity check	✓	Prototree (no. of prototypes check)
	• Concept disentanglement	✓	CRP (VGG16), L-CRP (Yolo7), CLIP
	4) Model Verification		
	• Compare models on explainability criteria[38]: i) Fidelity, ii) Stability, iii) Consistency, iv) Sparsity.	✓	i) Fidelity – ShapGAP (e.g. Prototree vs DT) ii) Stability – similar explanations for similar data points (ShapGAP) iii) Consistency (multiple evaluations of model and explanation consistency) iv) Sparsity (simplicity) – DT/Prototree (depth), crp (no. of relevant concepts in last layer)
	• Iterative (HiL) approach for assessing criteria	⏳	Metaheuristic search
PhIM	**1) Model Conversion – relevant to use in C libraries for Orin, e.g. quantization**	⏳	**ShapGAP to evaluate fidelity of parametrically simpler model (Yolo5 vs Yolo7; W)**; quantized model Yolo7-tiny wrt Yolov7 (not yet implemented); implement simple VGG (small) vs VGG16
	2) Model Optimization		
	• Pruning: layers, neurons etc.	×	Metaheuristic search
	• Neural coverage	×	Coverage metric extractors
	3) Model Deployment	-	For T3.5? **Load/Use trained model (W)**
	4) Inference Verification	-	For T3.5?
	5) XAI		
	• Anomaly detection (wrt verification dataset)	✓	**VAE (on verif. Dataset; W)**,
	• Surrogate models (checks fidelity/interpretability)	✓	**DT (W)**
	• Use of counterfactuals	⏳	Techniques yet to be implemented
	• Robustness analyses (add noise, flip pixels, weights etc.)	×	Techniques yet to be implemented

Colab Stage	Sub-phases	Colab?	EXPLib supporting functions/tools
OpXAI	• Supervisor OOD checks	⏳	VAE
	• Supervisor check of surrogate performance	⏳	Prototype based DT
	• Uncertainty		Not yet decided, example simple MLP model
	• Ensemble method	⏳	Voting, bagging
	• Metric extractor		Not yet decided

Key:

✓ = runnable from colab script (aifsm_eval.ipnyb)

× = not yet implemented (typically non-trivial, e.g. getting XAI algs to work with respect to OD)

✓ = (also purple font text) implemented but not callable from colab yet

⏳ = (also orange font text) a) not implemented at this project stage, or b) waiting for partner inputs or verification, or c) not able to get code to run completely yet (e.g. L-CRP).

⏳ = not yet implemented but not difficult (just in the queue)

W = part of Colab walkthrough toy/mini example.

At each phase of the walkthrough using dataset PASCAL VOC and target model Yolo7 log files of analyses, in relation to particular datasets or models, and plots are produced. The files contain a combination of stored values and placeholders (for further updating). See Appendix C for examples from PhDM, PhLM and PhIM. The supervisor-based outputs for PhIM also potentially serve as Operational XAI outputs for the DLLib (regarding metrics and supervisor specification). Concerning components required for the DLLib (see Figure 4), trained models for use in the DLLib can be found at in the operational_xai folder:

$root_dir/EXPLib/aifsm_phases/operational_xai/OOD/saved_vae_model/vae_model.pth

Yolo7 trained model (.pt) files (dl_model) can be found at:

$root_dir/EXPLib/dl_component/CNN/Object_Detectors/yolov7/weights/yolov7_training_files

As can be seen from Table 1, the phases of the AI-FSM ("Colab Stage" column) permit testing of the different sub-phases of each phase. Below we map onto descriptions provided in D2.1.

Data Management Phase (PhDM). This includes usages of different data explainer tools within the Data Management phase, with focus on assessments and data validation w.r.t. *PhDMT0001_Data_Requirements_Specifications_template* (D2.1), which are distributed over Data Collection, Data Preparation and Data Verification activities listed in Table 1.

For our walkthrough example, we use the PASCAL VOC dataset.

For data collection – data characteristics carried out and logged within colab for standard (non XAI based) such as file format, consistency of format, the specific dataset that the data belongs to, the

volume of data, are logged here in the walkthrough example (currently output to .txt file but will be output to another .json file with results compared against the aforementioned dataset_validation_rules.json.

For data verification, we evaluate *balance* of this dataset in relation to i) class distribution, i.e. evaluate the number of images within which each of the 20 classes of objects within the PASCAL VOC dataset are present, ii) bounding box statistics, i.e. distribution of dimensions of the bounding boxes over the dataset, iii) pixel intensity analysis, i.e. evaluate the pixel intensities of each image with respect to classes of data. This analysis can then be logged and compared against requirements via loading the dataset_validation_rules.json file stored within *aifsm_phases->data_management.* Figure 30, Figure 29, Figure 31 show plots for these analyses.

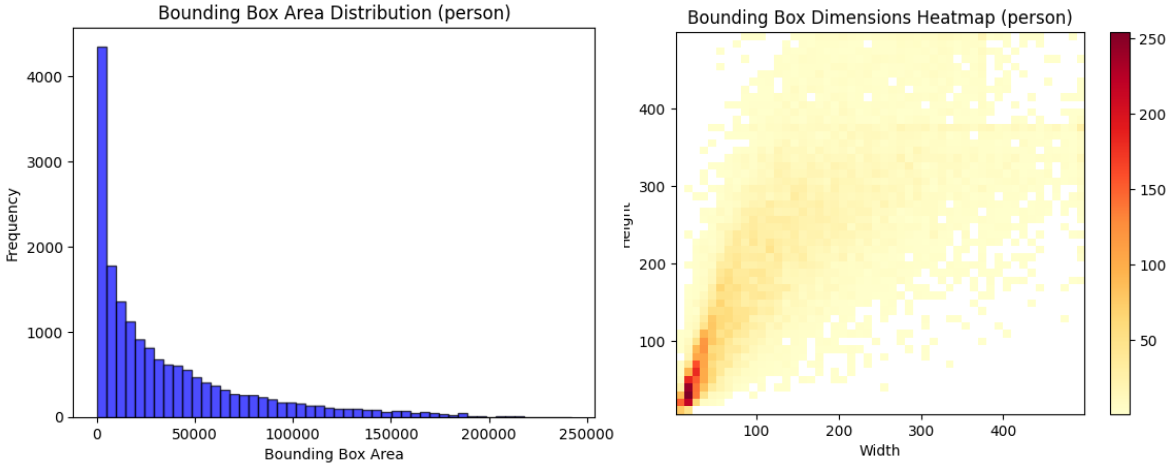


Figure 29: Bounding box distribution analysis of the PASCAL VOC dataset for the walkthrough example. Left. Distribution of bounding box dimensions for localizations of the person object. Right. Bounding box area distributions for person.

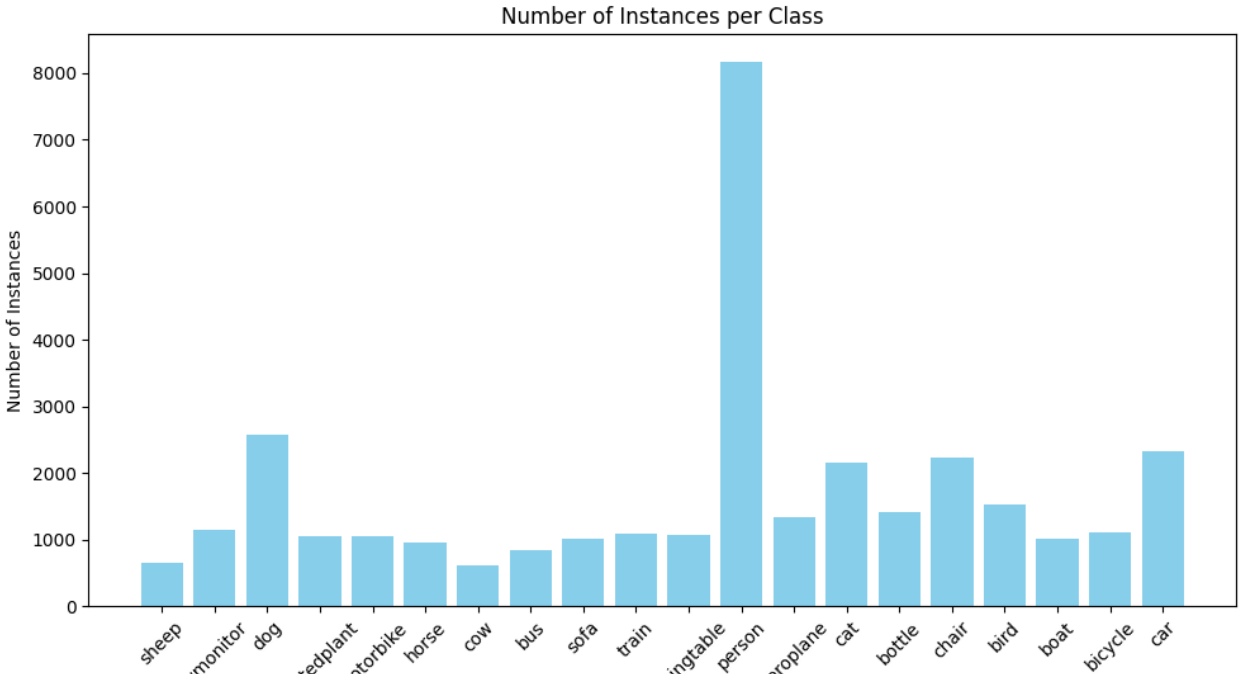


Figure 30: Class distribution analysis of the PASCAL VOC dataset for the walkthrough example. PASCAL VOC provides data instances that often contain multiple objects. Here it can be seen that "person" objects are much more represented than other objects and may potentially account for certain biases in model classification.

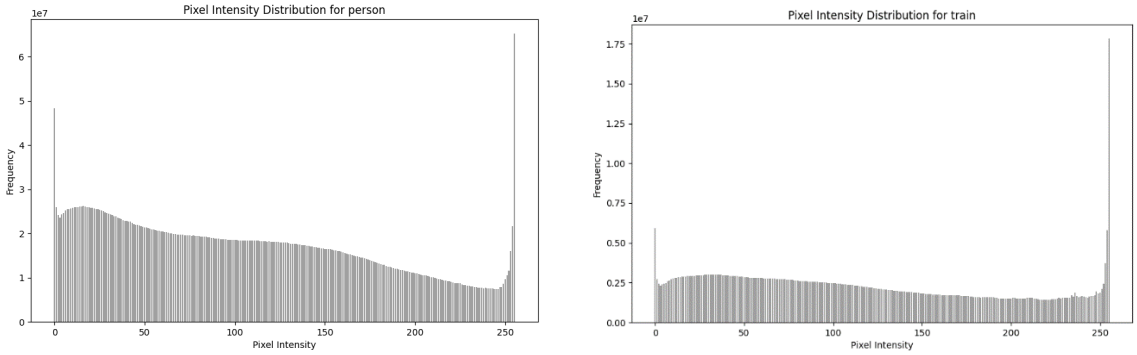


Figure 31: Pixel intensity analysis for example classes from PASCAL VOC dataset. Left: PIA for person. Right: PIA for train. The distribution for both classes show relatively large numbers of data instances for extreme values.

Learning Management Phase (PhLM). Within EXPLib is included folders and analyses for the different sub-phases of the PhLM as described in *PhLMG0002_Learning_Management_Guidelines.odt* and also D3.1.

For Model Training the walkthrough allows for training (using PASCAL VOC dataset) of the Yolo7 model and VAE (as the supervisor monitor component to be used for PhIM). Figure 32 illustrates Yolo7 trained performance metrics and VAE training versus validation loss performance over

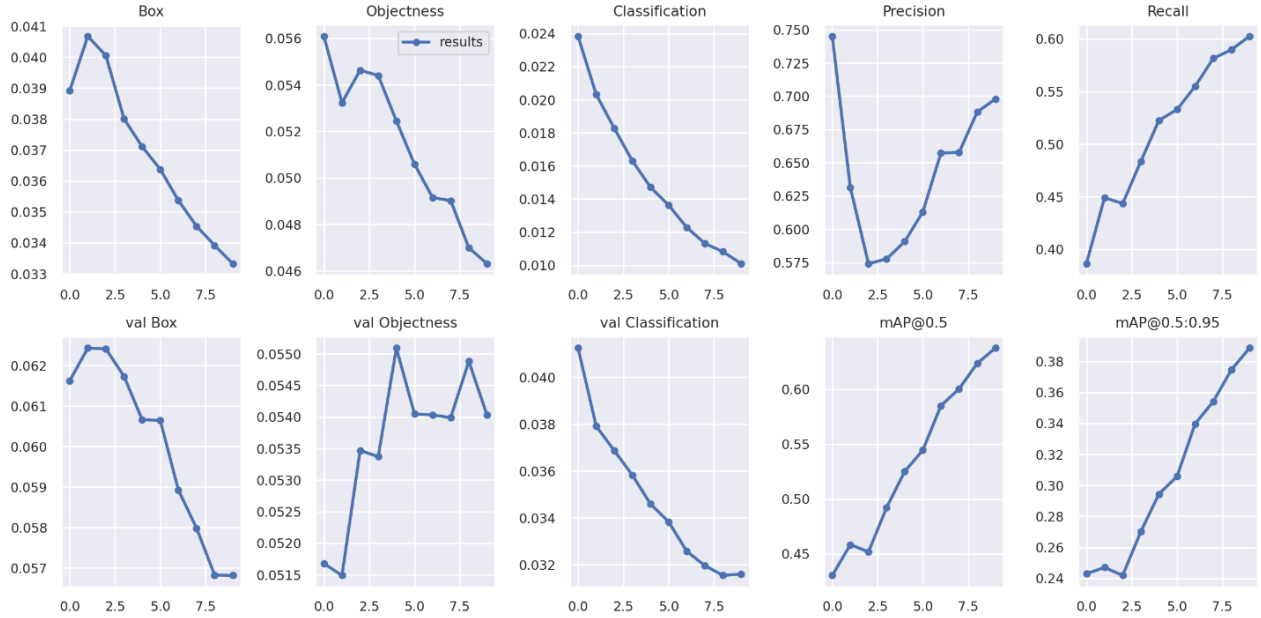


Figure 32: Training performance of Yolo7 over 10 epochs using PASCAL VOC. Note, only 10 epochs were run due to the training time involved.

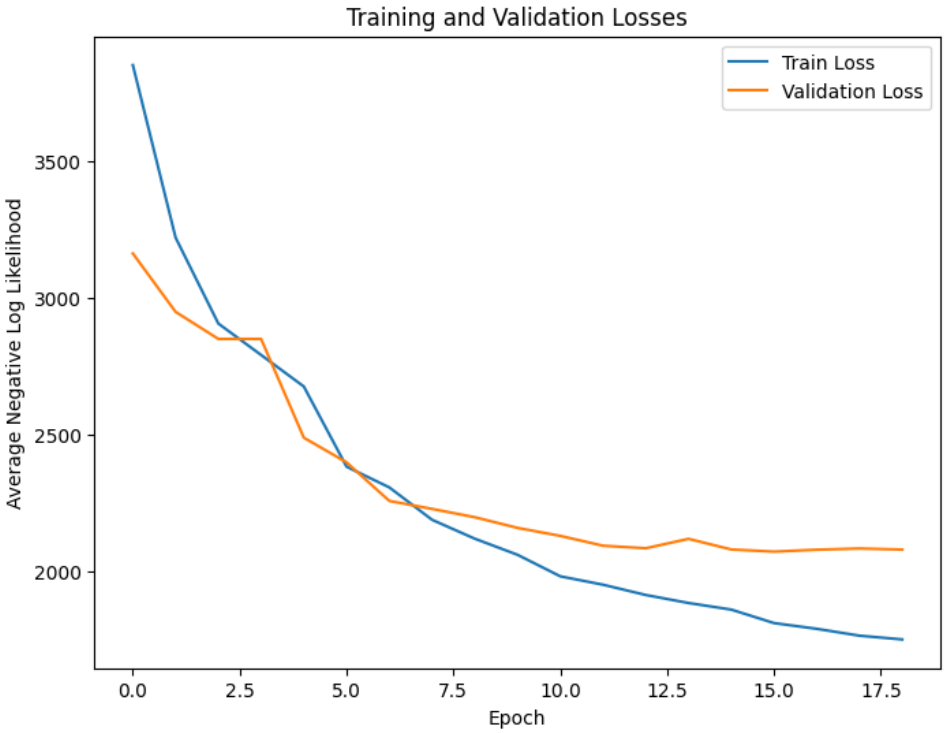


Figure 33: Training output of VAE (supervisor monitor algorithm) trained on PASCAL VOC. Training terminated after 18 epochs following 6 non decreasing validation loss results. This data is required to be logged to verify the model is not overtrained (does not overfit to the training data).

epochs. Figure 33 illustrated the trained VAE performance over epochs on the PASCAL VOC dataset with stopping criterion 6 non-reducing loss validation checks.

Inference Management Phase (PhIM). Within EXPLib is included folders and analyses for the different sub-phases of the PhLM as described in *PhLMG0003_Inference_Management_Guidelines.odt* and also D3.1. The VAE (OOD detector supervisor component) trained in the PhLM stage is used in this phase to evaluate anomalies in the test dataset. So far we have tested with respect to video frames taken from a video of car on a motorway and altered to contain "anomalous" features (see Figure 32 and Figure 33) and with another webcam customized dataset (see Figure 24 and D3.1).

For Model Conversion, this concerns carrying out analyses to meet requirements for model conversion for deployment, i.e. converting pytorch based Deep Learning components into models coded in C libraries to be deployed on the NVIDIA Jetson AGX Orin platform. Pre deployment it is necessary to evaluate the effects on performance and explanation fidelity when making changes to the model to reduce computational requirements for real time operation. Here we are carrying out analyses to compare Yolo7 with its quantized parameterically and architecturally smaller version Yolo7-tiny. We currently carry out checks to compare Yolov5 and Yolov7 on individual data points. We compare Yolo7 versus Yolo5 performance in Figure 34. The vectors of the flattened shap values superimposed over the images allow for cosine based and Euclidean distance evaluations between Yoloy5 and Yolo7. The images (left and centre) indicate that the bounded region for the person prediction provides positive (red pixels) shap values in both cases though the positive and negative (blue pixels) are somewhat inconsistent here. Nevertheless, using sufficiently well-trained models (in our example the Yolo7 model is only trained for 10 epochs) can provide a metric for comparison of consistent explainability over different models. This may be particularly important for quantization where a simpler model (e.g. Yolo7-tiny as compared to Yolo7) must be evaluated not only for consistency in performance to the more complex model but also for consistent explanations.

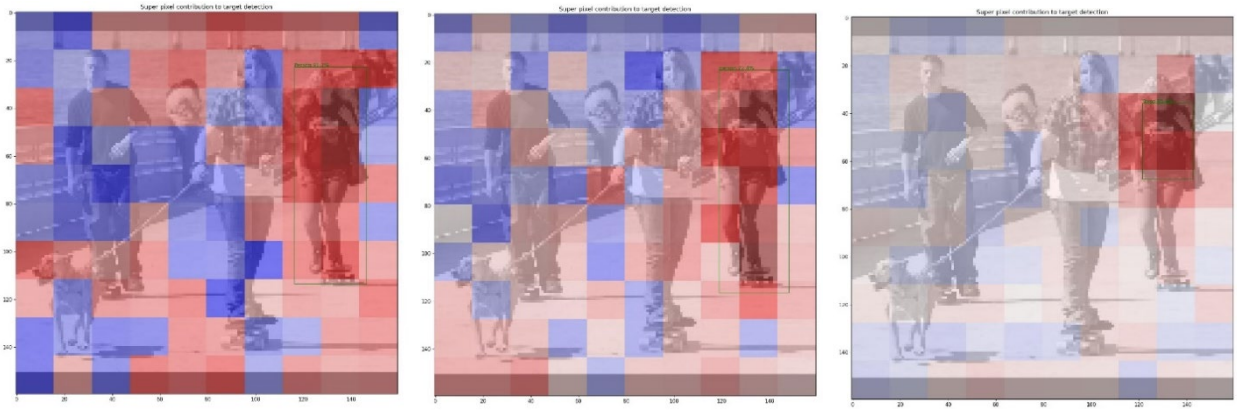


Figure 34: Shap values for Yolo7 (left) and Yolo5 (centre) in relation to a test image (not from PASCAL VOC) using the trained Yolo7 (best) model and with respect to a localized person (within a bounding box). Difference in performance is observably not great and is quantified using both cosine and Euclidean distance metrics. The right hand figure shows an example of the output of a Yolo7 model trained on the PASCAL-PART.v5i.yolov7pytorch dataset (more epochs than the Yolo5 and Yolo7 models) and can be shown to have high shap values (red pixels) for the torso component of the person. Note, different models (Yolo5, Yolo7 or Yolo7-tiny as examples) can only be reliably compared if trained to convergence (with indicators of not overfitting to the training data). The Yolo7 model here is trained on 10 epochs whereas the Yolo5 model is a downloaded pre-trained (many more epochs) model.

Further XAI approaches being evaluated: A number of analyses are being used to evaluate whether the trained and optimized model is ready for deployment in relation to the Inference Requirements Specifications (Figure 13). These analyses/checks relate both to model performance and explainability:

- *Anomaly detection* – using a trained model, VAE in EXPLib, it is possible to evaluate a new related dataset to assess whether the model is likely to generate anomalous predictions/explanations. The VAE can compare reconstruction error and latent distributions from the model using the training/test dataset to the model with the new dataset to assess significant deviations. This is implemented in EXPLib and usable through the colab walkthrough example.
- *Fidelity/Interpretability checks* – using an intrinsically explainable surrogate model (where 'surrogate' is considered as a substitute model for checking data relations or blackbox properties of the original model), as is implemented as a decision tree in EXPLib it is possible to gauge how different input-output relations in the original DL model (Yolo7 in the walkthrough example) can be approximated in a simpler model that allows for heightened interpretability. The input-output function might concern the mapping between the verification dataset (input) and prediction of a particular class of object (output) where the decision tree produces the rationale for the output (prediction) based on the input. In our walkthrough example, we investigate how well the decision tree can predict the class "person" using a dataset of person parts. These parts are labelled and could potentially be substituted in future investigations with labelled and disentangled "concept" (taken from the Localized-Concept Relevance Propagation, or L-CRP, algorithm) as a means to evaluate the fidelity between the surrogate model and the object detector (considering the mapping between Yolo7 - or other object detector - layers and (class) predictions).
- *Counterfactuals* – not yet accessible to colab/runnable code, in this part of the EXPLib we will verify the model according to its ability to detect counterfactual images, e.g. where a frame/image or images is inserted into a video stream whose characteristics differ, e.g. in terms of lighting. This has already been tested as an optical flow algorithm (see D2.2) and will be included in the EXPLib. This provides a means of verifying the model is able to identify anomalous data that might indicate an adversarial attack.
- *Robustness analyses* – not yet implemented, in this part of the EXPLib, we will provide code that implements verification analyses on the (DL) model's ability to be robust to noisy or corrupted values of the input data as well as gracefully handle small changes in values of weights (e.g. float precision changes) that may occur during model conversion. It is required that qualitative shifts in model performance, e.g. change in class prediction, as a result of small changes to these aforementioned properties should not cause significant prediction changes in the DL model.

The walkthrough of Yolo7 trained on the PASCAL VOC dataset and evaluated over the PhDM, PhLM and PhIM phases of the AI-FSM cycle whilst supported by EXPLib algorithms for XAI provides at this stage a simple demonstration that can be undertaken to measure performance and explainability of the various components. The Yolo7 trained model (model architecture definition and trained parameters) and supporting components (VAE trained model, explainer algorithms) are then loadable into DLLib providing its supporting library structure ready for conversion into the C libraries. The VAE serves as the OOD detector supervisor component.

3.6 Minimum Viable Product (MVP)

The aim of this section is to provide an MVP describing how EXPLib and DLLib can support the Toy model (D5.1, section 5) being compliant with the SAFEXPLAIN safety assurance framework (AI-FSM, D2.1) and the reference architecture patterns (D2.2) using a subset of proposed technical approaches in D3.1. The work with MVP has just started and will continue to the end of the project.

3.6.1 MVP reference architecture

Figure 35 illustrates the high-level architecture of the MVP, where the key focus (in scope of this deliverable) is on the Supervisory monitor component and the Decision function.

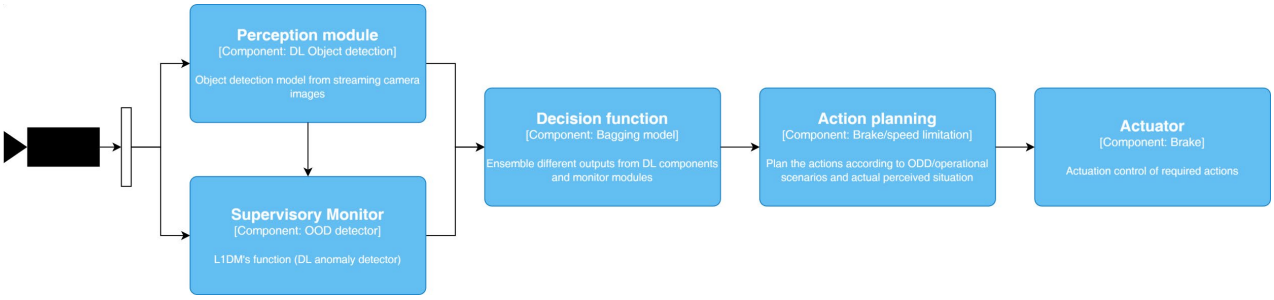


Figure 35: MVP highlevel reference architecture

Note that, at this point in time, the work on MVP has just started thus the following subsections are describing the planned actions.

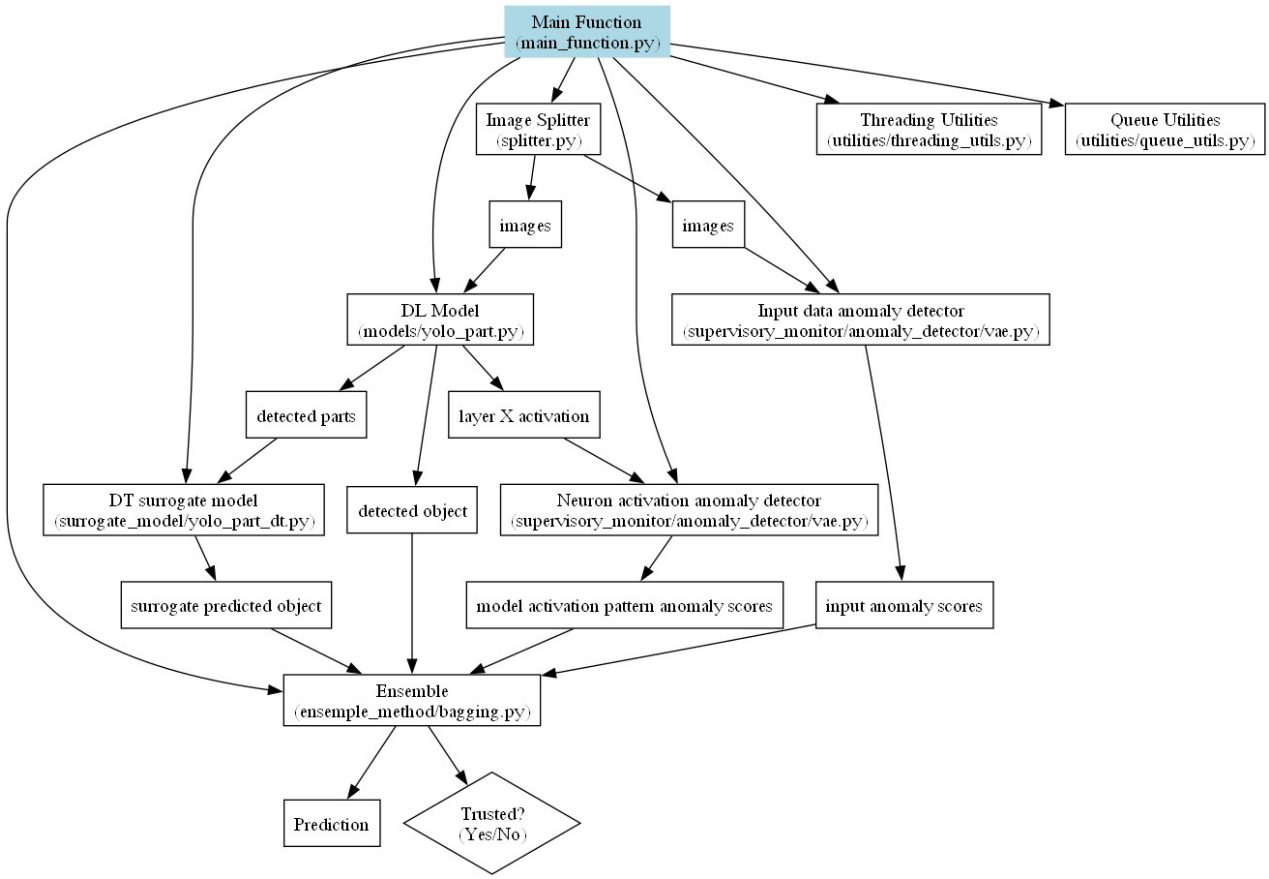


Figure 36: Realization structure of functions in Operational_XAI and DLLib

Figure 36 shows the realization structure of different python functions (reflecting the reference architecture in Figure 18). This structure is also used within DLLib.

The input image stream will be split by a splitter function into parallel input queues. DL component and Input data anomaly detector will read in the images from input queues and provide detections in the form of tuples (bounding box, object class, score) and input anomaly scores. The architecture is scalable and can adopt more DL components and supervisors.

Two extra supervisory monitors are also implemented: (i) Decision Tree based surrogate model that has been trained to provide object detection from detected parts, and (ii) anomaly detector, that uses VAE model trained on neuron activation patterns at a selected layer of the DL components).

The outputs from the DL component and the three supervisors will then be fed into decision function that uses one of the available ensemble methods to provide the consolidated prediction (bounding box, class) and the trustworthiness scores (either a binary Yes/No or a scalar for the next module in the pipeline, Action Planning, to decide).

The online metrics that can be extracted in real-time and post into the platform monitoring components can be selected from: model confidence scores, anomaly scores, and computed uncertainty levels (if uncertainty-aware DL model is deployed).

3.6.2 MVP component description

3.6.2.1 Sensors

Currently, the Camera sensor is used, where input data will be streaming timeseries images from a folder.

3.6.2.2 DL based perception module

The perception module is the Toy model described in D5.1, section 5. The module is an object detector CNN network based on SSD Lite MobileNetV3 large model. The reader is referred to D5.1 for further technical details.

3.6.2.3 Supervisory monitor

Initial implementation will start with using Variational Autoencoder (VAE) as an anomaly detector (OOD - Out of Distribution detector). The VAE model will be trained/verified with the same datasets used for Toy model development.

3.6.2.4 Decision function

Bagging Random Forest (RF) based model that works with input (bounding boxes, class, scores) and anomaly scores to derive the consolidated result in the form of (bounding box, class, trustworthiness core) will be used. The RF will be trained on the dataset's annotations (as its groundtruth) and DL model predictions + anomaly scores on the same datasets as its input data.

3.6.2.5 Action planning module

This module will take in the input from DL component and from supervisor (accept/reject) to decide:

- Planned actions of the ego vehicle (car, train, satellite)
- If safe mode or degradation mode should be enabled

This module is currently out of scope of the library, we will add a simple mockup module for testing.

3.6.2.6 Actuator

This module will take the planned actions from the action planning module and execute the actions. This module is currently out of scope of the library, we will add a simple mockup module for testing.

4 DL libraries

Building upon the exploration of AI-FSM lifecycle tools, data management strategies, and the pivotal role of XAI methods delineated in the preceding sections, and especially based on the work done in the EXPLib, this part transitions towards the ongoing and iterative implementation phase of DL software into the DLLib. The necessity for high-performance computing to accommodate the sophisticated models and datasets discussed previously underscores the adoption of low-level C-based libraries, such as cuBLAS and cuDNN (packaged within NVIDIA Jetpack), tailored for specific hardware architectures like NVIDIA GPUs and Arm-based CPUs. Given the recent arrival of crucial inputs from T3.4, which were slightly delayed, the DLLib work is now dynamically progressing for a set of diagnostic and supervision techniques as shown in Table 2. The current structure of DLLib is illustrated as a folder structure in Figure 37.

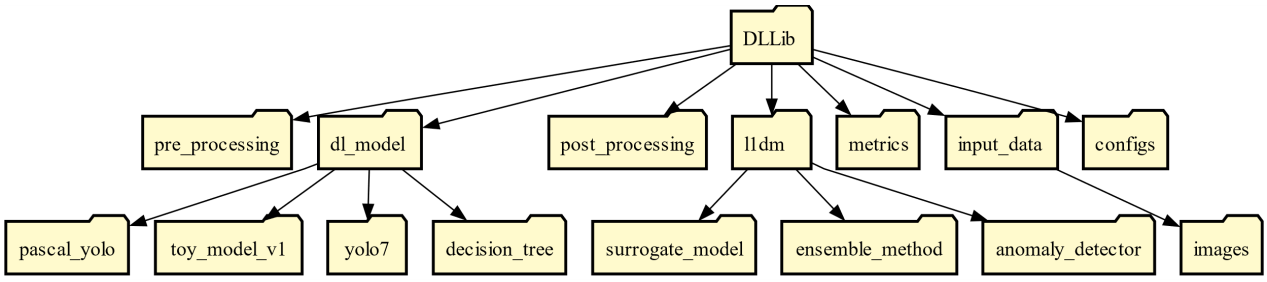


Figure 37: DLLib structure of the different sub-components as folders.

Table 2: DLLib components and current status considering the Railway Use case as an example.

Component	Status	Notes
Core Libraries	In Progress	Integration with cuBLAS and cuDNN ongoing
DL Models	In Progress	SSD model ready; Yolo models for Railway Use Case pending
Input Data Management	Awaiting Review	Offline verification datasets prepared
Runtime Metrics	In Development	Implementation of metrics system underway
Pre-processing	In Progress	Optimization for NVIDIA platform ongoing

Component	Status	Notes
Post-processing	Planned	Development to start post initial deployment tests
L1 Diagnosis & Monitoring	Conceptual	Detailed design in the next phase

The intricate relationship between the theoretical frameworks and practical application manifests in this transition, where the design and selection of DL models, data management protocols, and XAI methodologies, converge on the technological frontier of DL implementation. This phase leverages the groundwork laid by the examination of DL components and libraries, steering the project towards the implementation strategies that align with the unique computational and operational requisites of our target hardware platforms. Central to this section is the task of adopting, and where necessary, re-implementing existing C and C++ based libraries guided by the specifications derived from prior work (i.e. the core functionalities of the AI frameworks such as TensorFlow or Pytorch whenever is needed). This not only involves a deep dive into the functionalities offered by these low-level libraries but also an exploration of open-source equivalents that promise competitive performance without the overhead of starting from scratch. Such strategic decisions are underpinned by the project's commitment to efficiency, performance optimization, and the broader objective of harnessing the AI4EU platform's potential to foster reuse and minimize redundant development efforts.

As we delve into the specifics of implementing DL software across various hardware architectures, the focus sharpens on performance optimizations tailored to the case studies on the SAFEXPLAIN selected prototyping platform (NVIDIA AGX Orin), integrated with the low-level middleware PMULib (described in D4.1[39]). This encompasses a spectrum of activities from adapting black-box libraries to our needs, overhauling data preprocessing routines for efficiency, to fine-tuning runtime metrics for optimal performance. Each step is meticulously planned to ensure that the leap from theory to practice not only preserves the integrity of our design principles but also amplifies the capabilities of our DL systems to meet and exceed the project's ambitious goals. We integrate the core aspects of this implementation strategy across the detailed subsections.

4.1 DL Models

The DL model at the core of DLLib's deployment focus is a trained and verified Single Shot MultiBox Detector (SSD) object detector. This model is to be used for the Space Use Case (whereas variants of Yolo will be used for the Railway Use Case). This choice is motivated by the SSD model's exemplary balance of speed and accuracy in real-time object detection tasks, making it an ideal candidate for deployment in performance-sensitive applications. To further enhance the model's utility, particularly in the context of our project's emphasis on explainability, the SSD model undergoes modifications to incorporate explainability features directly into its operational framework.

Incorporating Explainability:

- **CAM Images and Neural Activation Patterns:** The model will be adapted to generate Class Activation Maps (CAM) images and neural activation patterns, providing visual

explanations of the model's decision-making process. This adaptation enables users to visualize the areas within an image that significantly influence the model's predictions, thereby enhancing the interpretability of the model's output.

- **Bayesian Modifications:** To address model uncertainty, the SSD model will be further enhanced through Bayesian modifications. This approach introduces a probabilistic layer to the model, allowing it to estimate uncertainties associated with its predictions. Such modifications are crucial for applications where decision-making under uncertainty is a key concern, as they provide a measure of confidence in the model's outputs.

Model Optimization for Deployment:

- The deployment phase involves optimizing the modified SSD model for the NVIDIA Jetson AGX Orin platform, leveraging the platform's computational resources to maximize inference speed while maintaining high accuracy. This optimization process takes into account the specific architectural features of the Orin platform, ensuring that the model utilizes the hardware's capabilities effectively.
- **Low-level Library Utilization:** The optimization process will include the strategic use of low-level C-based libraries, such as cuBLAS and cuDNN, to accelerate computation-intensive operations. These libraries facilitate efficient execution of the model's mathematical operations, enhancing the model's performance on the target hardware.

This comprehensive approach to the development and deployment of the DL model in DLLib not only ensures that the model's performance is optimized for the NVIDIA Jetson AGX Orin platform but also addresses the growing demand for transparent and interpretable AI systems. By integrating explainability features directly into the model, DLLib sets a new standard for deploying AI models that are both high-performing and understandable, aligning with the broader objectives of fostering trust and accountability in AI applications.

4.2 Input Data

DLLib adopts an innovative approach to input data management to simulate the intricacies of real-world operational environments accurately. This strategy involves the utilization of offline verification datasets, meticulously curated to mirror the data characteristics captured by real camera sensor image streams. These datasets are stored within a dedicated folder structure designed to ease the development and testing process, providing developers with a reliable framework for verifying the performance and accuracy of DL models under simulated conditions.

ROS2 Integration:

- **Data Stream Simulation:** The verification datasets are read and then posted to a ROS2 topic, which closely resembles the ROS2 topics used to interface with actual camera sensors. This method ensures a seamless transition from offline testing to real-world application, enabling developers to fine-tune models and systems within a controlled environment that accurately reflects operational conditions.
- **Operational Consistency:** By mimicking the real camera sensor data stream using ROS2 topics, DLLib ensures that the models and systems developed are fully compatible with the deployment environment. This approach significantly reduces integration challenges, facilitating smoother deployment processes and ensuring that the models perform as expected in real-world scenarios.

The focus on simulating real-world data streams and integrating with ROS2 topics underscores DLLib's commitment to creating a robust and efficient development ecosystem. This environment not only supports the optimization of DL models for specific deployment scenarios but also enhances the reliability and effectiveness of the deployed solutions, ensuring they meet the stringent demands of real-world applications.

4.3 Runtime metrics

DLLib introduces a mechanism for monitoring and optimizing runtime metrics, a cornerstone for deploying DL models effectively. This system encompasses tools for extracting comprehensive performance metrics, including frame processing time, confidence levels, DL model performance, and supervisory monitor outputs, which now include explainability measures. This expansion allows for a dynamic trade-off mechanism in runtime, offering a 2D manifold representation in a three-dimensional space encompassing time, explainability, and performance. This innovative approach facilitates more informed predictions, balancing accuracy, trustworthiness, and timeliness essential for safety-critical systems.

Example implementations illustrating this concept include:

- **Multiple Parallel DL Modules**: Employing DL modules with varied performance metrics and execution times per frame enables flexible adjustments based on real-time requirements.
- **Multiple Supervisor Modules**: These modules provide safety-related assessments at different timestamps, offering a nuanced understanding of system performance within critical time windows.
- **Trade-off Component**: This component continuously monitors and logs metrics from each system component, adjusting operational parameters to meet predefined requirements effectively.

Such advanced metrics and trade-off measures are designed to optimize operational efficiency, providing a detailed framework for balancing system performance against time constraints. This approach underscores the DLLib's commitment to deploying highly responsive and reliable DL systems.

4.4 Pre-processing

In DLLib, pre-processing extends beyond mere data manipulation to include crucial Non-AI algorithms for data format checking, conversion, and validation. These preliminary steps are vital for ensuring that the input data is in the correct format and meets quality standards before being fed into DL components and supervisory systems. Leveraging the computational capabilities of cuBLAS, cuDNN, and similar libraries, DLLib executes these non-AI algorithms with high efficiency, ensuring rapid and accurate preparation of data for processing.

The pre-processing phase thus encompasses:

- **Data Format Checking:** Verification of data formats against expected standards to ensure compatibility with DL models.

- **Data Conversion:** Transformation of data into formats required by DL models, utilizing optimized routines to minimize processing time.
- **Data Validation:** Assessment of data quality and integrity, identifying and handling outliers or missing values to maintain the robustness of DL models.

This comprehensive approach to pre-processing underlines DLLib's commitment to operational efficiency, enhancing the performance and reliability of the deployment process. By addressing these non-AI aspects with the same rigor as DL model optimization, DLLib ensures a streamlined data pipeline, contributing to faster model training, improved overall system performance, and the delivery of high-quality, actionable insights through deployed DL models.

4.5 Post-processing

Given the current dynamic state of our project, with ongoing developments and pending tasks, the immediate necessity for complex post-processing tasks, such as reformatting model outputs for downstream analysis, is deemed non-essential. However, recognizing the diverse nature of DL model outputs, particularly in scenarios involving multiple model types (e.g., YOLO vs. SSD), there exists a potential requirement for standardized post-processing routines. These routines would aim to parse and normalize outputs, ensuring uniformity across different model architectures for seamless integration with subsequent modules.

To address future needs, DLLib anticipates the development of lightweight, adaptable post-processing tools capable of:

- **Parsing Diverse Output Structures:** Developing parsers that can interpret and transform varied output formats from different DL models into a standardized format.
- **Normalizing Outputs:** Implementing normalizers to ensure that model outputs, regardless of their origin, conform to a unified structure, facilitating easier consumption by downstream applications or systems.

This strategic approach allows for the flexibility to incorporate more complex post-processing tasks as the project evolves, ensuring DLLib remains adaptable to emerging requirements while minimizing the immediate burden on system resources and complexity.

4.6 L1 Diagnosis and Monitoring

DLLib is planned to incorporate a Supervisor (L1 Diagnosis and Monitoring) system aligned with the reference architecture pattern outlined in document D2.2. This system is designed to enhance the reliability and efficiency of deployed DL components through the integration of advanced supervisory monitor components, including:

- **OOD Detector Algorithms:** Utilizing trained Variational Autoencoder (VAE) descriptors, this component identifies data points or patterns that deviate significantly from the model's training distribution, flagging potential outliers or novel scenarios that require additional scrutiny.
- **(Intrinsically Explainable) Surrogate Model:** A simplified representation of the deployed DL models, allowing for rapid analysis and understanding of model behaviors. This component facilitates the exploration of model responses to various inputs, aiding in the identification of potential issues or areas for improvement.

- **Ensemble Methods:** Combining predictions from multiple models to improve reliability and accuracy. This approach leverages the strengths of various models to achieve better performance and robustness than any single model could on its own.

By integrating these supervisory monitor components with low-level libraries, DLLib ensures a deep and nuanced understanding of the deployed system's performance. This integration allows for the effective monitoring and management of DL components, leveraging detailed performance and error metrics to make informed decisions about model adjustments, parameter tuning, and resource allocation. The comprehensive L1 Diagnosis and Monitoring system ensures that the deployed DL system remains efficient, accurate, and reliable, adapting dynamically to evolving operational conditions and maintaining high standards of performance over time.

References

[1] SAFEXPLAIN, "D2.1: SAFEXPLAIN Safety Lifecycle Considerations." Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2024.

[2] SAFEXPLAIN, "D3.1: SAFEXPLAIN Specifiability, explainability, traceability, and robustness proof-of-concept and argumentation." Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2024.

[3] SAFEXPLAIN, "D2.2: SAFEXPLAIN DL safety architectural patterns and platform." Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2024.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Jun. 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.

[5] E. Mariotti, A. Sivaprasad, and J. M. A. Moral, "Beyond Prediction Similarity: ShapGAP for Evaluating Faithful Surrogate Models in XAI," in *Explainable Artificial Intelligence*, L. Longo, Ed., in Communications in Computer and Information Science. Cham: Springer Nature Switzerland, 2023, pp. 160–173. doi: 10.1007/978-3-031-44064-9_10.

[6] SAFEXPLAIN, "D5.1: SAFEXPLAIN Case study stubbing and early assessment of case study porting." Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2024.

[7] J. R. Zilke, E. Loza Mencía, and F. Janssen, "DeepRED – Rule Extraction from Deep Neural Networks," in *Discovery Science*, T. Calders, M. Ceci, and D. Malerba, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 457–473. doi: 10.1007/978-3-319-46307-0_29.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[9] A. Howard *et al.*, "Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South): IEEE, Oct. 2019, pp. 1314–1324. doi: 10.1109/ICCV.2019.00140.

[10] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2.

[11] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015, Accessed: Dec. 06, 2021. [Online]. Available: http://arxiv.org/abs/1409.1556

[12] M. Nauta, R. Van Bree, and C. Seifert, "Neural Prototype Trees for Interpretable Fine-grained Image Recognition," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA: IEEE, Jun. 2021, pp. 14928–14938. doi: 10.1109/CVPR46437.2021.01469.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[14] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 5987–5995. doi: 10.1109/CVPR.2017.634.

[15] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "RepVGG: Making VGG-style ConvNets Great Again," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA: IEEE, Jun. 2021, pp. 13728–13737. doi: 10.1109/CVPR46437.2021.01352.

[16] J. Redmon, "Darknet: Open Source Neural Networks in C." 2013. [Online]. Available: http://pjreddie.com/darknet/

[17] M. Dreyer, R. Achtibat, T. Wiegand, W. Samek, and S. Lapuschkin, "Revealing Hidden Context Bias in Segmentation and Object Detection through Concept-specific Explanations," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jun. 2023, pp. 3829–3839. doi: 10.1109/CVPRW59228.2023.00397.

[18] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, in NIPS'17. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 4768–4777.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[20] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," *arXiv:1405.0312 [cs]*, Feb. 2015, Accessed: Jun. 25, 2020. [Online]. Available: http://arxiv.org/abs/1405.0312

[21] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, *The Caltech-UCSD Birds-200-2011 Dataset*. California Institute of Technology, 2011.

[22] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[23] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," *arXiv:1602.04938 [cs, stat]*, Aug. 2016, Accessed: Jul. 12, 2021. [Online]. Available: http://arxiv.org/abs/1602.04938

[24] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 618–626. doi: 10.1109/ICCV.2017.74.

[25] M. B. Muhammad and M. Yeasin, "Eigen-CAM: Class Activation Map using Principal Components," in *2020 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2020, pp. 1–7. doi: 10.1109/IJCNN48605.2020.9206626.

[26] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation," *PLOS ONE*, vol. 10, no. 7, p. e0130140, Jul. 2015, doi: 10.1371/journal.pone.0130140.

[27] R. Achtibat *et al.*, "From Attribution Maps to Human-Understandable Explanations through Concept Relevance Propagation," *Nat Mach Intell*, vol. 5, no. 9, pp. 1006–1019, Sep. 2023, doi: 10.1038/S42256-023-00711-8.

[28] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv:1312.6114 [cs, stat]*, May 2014, Accessed: Dec. 06, 2021. [Online]. Available: http://arxiv.org/abs/1312.6114

[29] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training." arXiv, Nov. 13, 2018. doi: 10.48550/arXiv.1805.06725.

[30] J. Ren *et al.*, "Likelihood Ratios for Out-of-Distribution Detection." arXiv, Dec. 05, 2019. doi: 10.48550/arXiv.1906.02845.

[31] ISO/IEC TR 5469:2024, "Artificial intelligence — Functional safety and AI systems." 2024. Accessed: Mar. 01, 2024. [Online]. Available: https://www.iso.org/standard/81283.html

[32] L. McInnes, J. Healy, N. Saul, and L. Großberger, "UMAP: Uniform Manifold Approximation and Projection," *Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018, doi: 10.21105/joss.00861.

[33] S. Jesus *et al.*, "How can I choose an explainer? An Application-grounded Evaluation of Post-hoc Explanations," in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, in FAccT '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 805–815. doi: 10.1145/3442188.3445941.

[34] J. H. Friedman, "Multivariate Adaptive Regression Splines," *The Annals of Statistics*, vol. 19, no. 1, pp. 1–67, 1991, doi: 10.1214/aos/1176347963.

[35] D. W. Apley and J. Zhu, "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 82, no. 4, pp. 1059–1086, Jun. 2020, doi: 10.1111/rssb.12377.

[36] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-Precision Model-Agnostic Explanations," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Art. no. 1, Apr. 2018, doi: 10.1609/aaai.v32i1.11491.

[37] L. Attouche, M.-A. Baazizi, D. Colazzo, G. Ghelli, C. Sartiani, and S. Scherzinger, "Validation of Modern JSON Schema: Formalization and Complexity." arXiv, Feb. 01, 2024. doi: 10.48550/arXiv.2307.10034.

[38] J. Dai, S. Upadhyay, U. Aivodji, S. H. Bach, and H. Lakkaraju, "Fairness via Explanation Quality: Evaluating Disparities in the Quality of Post hoc Explanations," in *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*, in AIES '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 203–214. doi: 10.1145/3514094.3534159.

[39] SAFEXPLAIN, "D4.1: SAFEXPLAIN Interim Platform Technologies Report." Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2024.

Acronyms and Abbreviations

- AI- Artificial Intelligence
- CAM – Class Activation Maps (XAI algorithm)
- CRP – Concept Relevance Propagation (XAI algorithm)
- DLLib – Deep Learning Deployment Library
- EXPLib – Explainable Deep Learning Library
- Dx.x – deliverable
- DL- Deep Learning
- KPI – Key Performance Indicator
- LIME - Local Interpretable Model-agnostic Explanations (XAI algorithm)
- L-CRP – Localized Concept Relevance Propagation (XAI algorithm)
- LRP – Layerwise Relevance Propagation (XAI algorithm)
- OOD – Out of Distribution
- PhDM – Data Management Phase
- PhLM – Learning Management Phase
- PhIM - Inference Management Phase
- SHAP - SHapley Additive explanations (XAI algorithm)
- SSD – Single Shot Detector (Object detector algorithm)
- Tx.x – task
- VAE – Variational Autoencoder
- WPx – Work Package
- YOLO – You Only Look Once (Object detector algorithm)

Appendix A

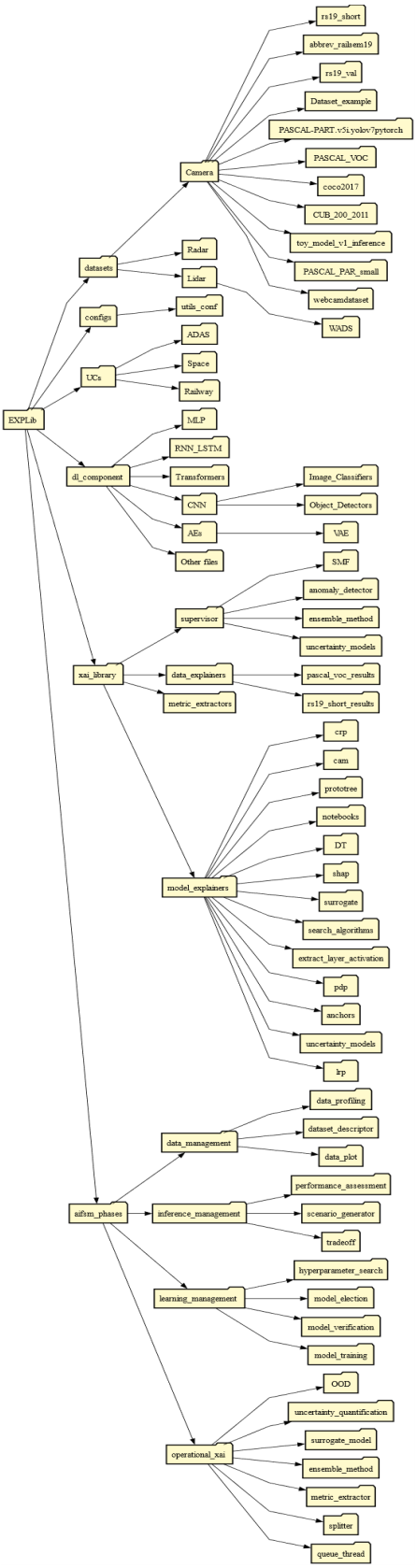


Figure 38: showing full EXPLib structure as it is currently populated (some files not shown).

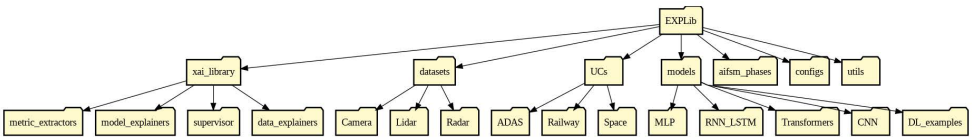
Appendix B

README.txt file describing the cell by cell walkthrough for the aifsm_eval.ipynb colab script. The file maps to the state of development of the EXPLib (and colab walkthrough demo) in Table 1. This file is being updated in relation to explainable methods/algorithms that are being integrated within the runnable colab version and an offline version of EXPLib.

A shareable Drive link to the walkthrough code is available on request.

EXPLib AI-FSM explainer library.

The following is a library created for providing support and permitting development of dependable Deep Learning components according to protocols used on the SAFEXPLAIN project (https://safexplain.eu/).



The ***EXPLib*** contains scripts for supporting the different steps of AI-FSM - i) Data Management Phase (PhDM), ii) Learning Management Phase (PhLM), iii) Inference Management Phase (PhIM).
Scripts relevant to these phases are contained in the ***aifsm_phases*** folder.

For each of the AI-FSM phases is callable a number of functions/scripts contained within the ***xai_library*** folder.

In /EXPLib. is contained a runnable jupyter script runnable in colab: *aifsm_eval.ipynb* . This script allows for supporting relevant tests for the Use Cases in relation to the different AI-FSM phases.

The walkthrough example of usage of the EXPLib can be achieved by carrying out the following steps:

Mandatory -
Run Cell 1 : mount Drive using your root Drive directory.
Run Cell 2: set up your root directory and change directory to EXPLib.
Run Cell 3: for necessary installations as well as global variables
Run Cells 4: function needed for selecting dataset (or model)

PhDM walkthrough -
PhDM Data Collection -
Run Cell 5 : for UMAP visualization imports
Run Cell 6 : for importing the PhDM_controller.py script that links to a number of functions for data management as well as data explainer functions.
Run Cell 7 : for permitting input to select dataset
Run Cell 8 : for running function in Cell 7 (input desired requested dataset).
Run Cell 9: run analysis for the PhDM phases -

- For the *rs19_short* dataset analysis can be done with respect to i) class distribution, ii) format type/consistency, iii) dimension reduction analyses - for saved images go to *./xai_library/data_explainers/rs19_short_results/*

- For the *PASCAL-VOC* dataset analysis can be done with respect to i);
  - data logs are saved in: $root_dir/EXPLib/aifsm_phases/data_management/data_eval_file.json
  - data plots are saved in: $root_dir/EXPLib/aifsm_phases/data_management/data_plot/pascal_voc_results

Run Cells 10-11: for logging data analysis of non-object detector datasets, i.e. only for image classification.

PhLM walkthrough -
Run Cells 7: need to select dataset if training model
Run Cell 12: for importing the PhLM_controller.py script that links to a number of functions for learning management as well as model explainer functions.
Run Cell 13: choose model, either an object detector or an image classifier depending on dataset selected
Run Cells 14-17: (NB: if you are not going to train/finetune, no need to run these cells)

- for training or pre-training (not yet implemented) models. Note, for Yolo7 this must be trained off Drive (cloud). You need to specify full paths in order to save weights into Drive. Yolo7 provides logged data for i) per epoch mAP performance; ii) Confusion Matrix (final training data epoch?), iii) PR and F1 curves, iv) test and train batch inference examples.
- For Cell 15, you can set hyperparameters, ideally would be in the calling function in Cell 18, naturally.
- For Cell 18, if you wish to train you should assign false to the "pretrained" flag

Note, running Cell 17 requires restart of colab and annoyingly you have to go back to step 1: rerun all previously run cells (except Cell 17).
Run Cell 18: Run Inference example with trained/pretrained model
Run Cell 19: Train VAE anomaly detector on dataset (or our "anomaly" test dataset) - relevant if comparing different model architectures and then loading for the PhDM stage.
Run Cell 20: PhLM XAI - run Decision Tree - interpretability of dataset/concepts
Run Cell 21: PhLM XAI - run saliency map - Kernel-Shap for Yolo7 : This will work with different trained models (e.g. best.pt as well as all_in_one.pt) but test image reshaping is critical to avoid axis out of bounds errors.

- data logs (for Yolo7 with PASCAL VOC) are saved in: $root_dir/EXPLib/aifsm_phases/learning_management/model_verification/model_output_files/model_learning_eval.json
- data plots are saved in: $root_dir/EXPLib/aifsm_phases/learning_management/model_verification/model_output_files/explainer_values

PhIM walkthrough -
Run Cell 22: for importing the PhDM_controller.py, PhLM_controller.py and PhIM_controller.py scripts - all needed here.

Run Cells 23:

- Shap-GAP (euclidean, cosine distance) of Yolo5 vs Yolo7 models possible here

Run Cells 24-26:

- Inference verification checks using trained (on training dataset) VAE for anomaly detection, and OF (Ikerlan) model

Run Cells 20: For re-running Decision Tree (surrogate)

Run Cell 27: Log results :

- data logs are saved in: $root_dir/EXPLib/aifsm_phases/inference_management/performance_assessment/supervisor_log.json
- data plots are saved in:
- $root_dir/EXPLib/aifsm_phases/inference_management/performance_assessment/plots

Appendix C

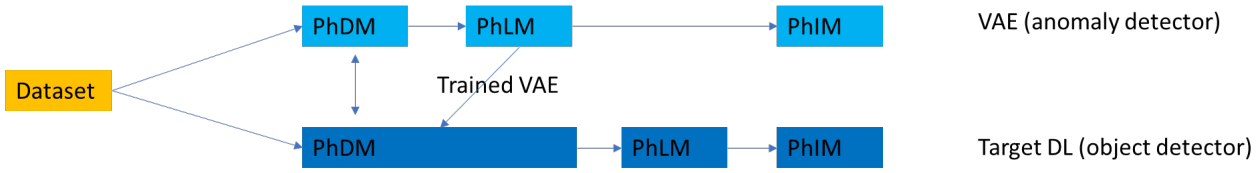


Figure 39 visualization of the AIFSM support pipeline for two models: i) Variational Autoencoder (VAE) as a trainable anomaly detector to be used in DLLib, ii) Target Deep Learning algorithm (in the walkthrough case Yolo7) as a trainable object detector that is to be the dependenable component.

In Figure 39 is illustrated the walkthrough pipeline for passing VAE (aonomaly detector) and Yolo7 (Target DL/object detector) through the AIFSM supported phases. Note, the trained VAE in the PhLM phase of the VAE's pipeline serves as the loaded VAE model in the PhDM phase of the Yolo7 model pipeline. VAE and Yolo7 models are to be saved and ready for use in DLLib as well as logged regarding training, performance and relevant explainable components.

Below are provided (snippet) example log files (.json format) for each of the PhDM, PhLM and PhIM phases. These files provide output from the walkthrough demo that uses the PASCAL VOC dataset and target model Yolo7. More information will be provided for these files following M18.

Table 3: PhDM snippet walkthrough log file (size precludes inclusion of entire file).

data_eval_file.txt


```
{
    "data": {
        "data_info": {
            "dataset": [
                "PASCAL_VOC"
            ],
            "image height mean": 398.94,
            "image height sd": 67.76,
            "image width_mean": 456.67,
            "image width sd": 67.85
        }
    },
    "balance": {
        "class_distribution": {
            "sheep": 650,
            "tvmonitor": 1150,
            "dog": 2572,
            "pottedplant": 1054,
            "motorbike": 1052,
            "horse": 964,
            "cow": 606,
            "bus": 842,
            "sofa": 1014,
            "train": 1088,
            "diningtable": 1076,
            "person": 8174,
            "aeroplane": 1340,
            "cat": 2160,
            "bottle": 1412,
            "chair": 2238,
            "bird": 1530,
            "boat": 1016,
            "bicycle": 1104,
            "car": 2322
        },
        "bbox_distribution": {
            "Centre Y mean": 232.36,
            "Centre Y sd": 96.86,
            "Centre X mean": 215.14,
            "Centre X sd": 57.91,
            "Bbox height mean": 149.78,
            "Bbox height sd": 103.84,
            "Bbox width mean": 227.51,
            "Bbox width sd": 107.56
        }
    },
```


Below is also provided an additional snippet with information regarding the trained OOD anomaly detector evaluated on test data:


```
    "OOD anomaly detector info": {
        "Vae values": {
            "model": {
                "model path": "/content/drive/MyDrive/SAFEXPLAIN-WP3-Railway/WP3/EXPLib/EXPLib/dl_component/AEs/VAE/saved_vae_model"
            },
            "image(s)": {
                "image": "/content/drive/MyDrive/SAFEXPLAIN-WP3-Railway/WP3/EXPLib/EXPLib/datasets/Camera/PASCAL_VOC/VOC2012_train_val/VOC2012_train_val/JPEGImages/2007_000032.jpg"
            },
            "metric": {
                "mse reconstruction error": 0.0144759500399231 91
            }
        }
    },
```

Table 4: PhLM walkthrough log file.

model_learning_eval.json | data_eval_file.txt

Schema: <No Schema Selected>


```
{
    "dl_component": {
        "model": "yolov7",
        "weights": "yolov7_training.pt.1"
    },
    "data": {
        "dataset": "PASCAL_VOC"
    },
    "evaluation_results": {
        "epoch": "9/9",
        "gpu_memory": "15.5G",
        "losses": {
            "box": "0.03333",
            "objectness": "0.04631",
            "class": "0.01009",
            "total": "0.08973"
        },
        "labels": "469",
        "image_size": "640",
        "metrics": {
            "precision": "0.6979",
            "recall": "0.6026",
            "mAP@.5": "0.6374",
            "mAP@.5:.95": "0.3887"
        }
    }
}
```

Table 5: PhIM walkthrough log file.


```
{
    "Dl Component (original model)": "",
    "Original dataset": "PASCAL_VOC",
    "Test dataset": "",
    "anomaly(OOD)_detector vals": {
        "OOD model": "vae",
        "epochs": 50,
        "batch size": 10,
        "patience": 6,
        "test data": "",
        "anomaly_scores": [
            0.08011891692876816,
            0.011549245566129684,
            0.048127420246601105,
            0.05978499725461006
        ]
    },
    "EBM (Surrogate) vals": {
        "original model": "yolo7",
        "input_vector": "PASCAL_PARTS",
        "max_depth": 5,
        "min_split": 20,
        "min_leaf": 15,
        "max_nodes": 20,
        "extraction_vec": [
            0,
            1,
            4
        ],
        "mse - model vs surrogate": 17308.47
    },
    "Uncertainty estimator": {
        "": ""
    },
    "XAI_checks (Shap)": {
        "original model": "yolo7",
        "comparison model": "yolo5",
        "input_vector": "",
        "shap_vals": "",
        "shap_to_pixel_vals": "",
        "bounded_shap_to_pixel_vals": "",
        "shap_gap euclidean": "0.25",
        "shap gap cosine": "0.58"
    }
}
```