Deliverable 3.4:

I-NERGY Models, AI Methods and Tools

31st August 2021

Lead: RWTH

Version: 1.00

Classification: Public

I-NERGY project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101016508

www.i-nergy.eu

I-NERGY project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101016508

I-NERGY Project Profile

Grant Agreement No.:	**101016508**
Acronym:	**I-NERGY**
Title:	**Artificial Intelligence for Next Generation Energy**
Type:	**Innovation Action (IA)**
URL:	**https://i-nergy.eu/**
Start Date:	**01/01/2021**
Duration:	**36 months**

Work Package:	**WP3 –** Data Management Enablers for Learning Model Validation and Sharing
Deliverable Leader:	**RWTH**
Authors (Organisation)	**Chijioke Eze (RWTH), Leonardo Lombardi (ENG), Carolina Fortuna (COMS), Charles Emehel (RWTH), Bettina Schaefer (RWTH), Leonardo Carreras (RWTH), Marija Borisov (ENG)**
Internal Reviewers:	**Vagelis Karakolis (ICCS), Sotiris Pelekis (ICCS), Spiros Mouzakitis (ICCS)**
Status:	**Final**
Date of Delivery:	**31ist August, 2021**
Version:	**1.00**
Classification:	**Public**

Document History

Version	Date	Author (Partner)	Remarks
0.1	16/03/2021	Chijioke Eze (RWTH)	ToC
0.2	14/04/2021	Leandro Lombardi (ENG)	ToC Review
0.3	26/04/2021	Chijioke Eze (RWTH)	ToC update
0.4	09/07/2021	Leonardo Lombardi (ENG), Marija Borisov (ENG)	Section 2.7, 2.8. 3.3
0.5	13/07/2021	Chijioke Eze (RWTH)	Section 2.1, 2.2, 2.3
0.6	19/07/2021	Carolina Fortuna (COMS)	Section 2.6, 2.9, 3.2, 3.4
0.7	20/07/2021	Charles Emehel (RWTH)	Section 2.5
0.8	23/07/2021	Chijioke Eze (RWTH)	Section 1.1, 1.2, 3.1 & 4
0.9	25/07/2021	Charles Emehel (RWTH)	Section 2.4
0.91	28/07/2021	Chijioke Eze, Charles Emehel, Bettina Shaefer, Leonardo Carreras (RWTH)	Refinement
0.92	25/08/2021	Chijioke Eze, Leonardo Carreras, Charles Emehel, (RWTH)	Refinement
1.0	31/08/2021	Chijioke Eze (RWTH)	Final version based on feedback from internal reviewers

Partners

	Participant Name	Short Name	Country Code	Logo
1	INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS	ICCS	GR	ICCS ΕΠΙΣΕΥ
2	ENGINEERING – INGEGNERIA INFORMATICA SPA	ENG	IT	ENGINEERING
3	RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN	RWTH	DE	E.ON Energy Research Center RWTH AACHEN UNIVERSITY
4	COMSENSUS, KOMUNIKACIJE IN SENZORIKA, DOO	COMS	SL	COMSENSUS
5	FUNDACION CARTIF	CARTIF	ES	[TECHNOLOGY CENTRE] CARTIF
6	DEUTSCHES FORSCHUNGSZENTRUM FUR KUNSTLICHE INTELLIGENZ GMBH	DFKI	DE	DFKI German Research Center for Artificial Intelligence
7	PARITY PLATFORM IDIOTIKI KEFALAIOUXIKI ETAIREIA	PARITY	DE	PARITY
8	FUNDINGBOX ACCELERATOR SP ZOO	FBA	PL	FundingBox
9	CENTRO DE INVESTIGACAO EM ENERGIA REN - STATE GRID SA	R&D NESTER	PT	R&D NESTER
10	ASM TERNI SPA	ASM	IT	ASM ASM Terni S.p.A.
11	SONCE ENERGIJA D. O. O.	SONCE	SI	SONCE Partner of SUNCONTRACT
12	IRON THERMOILEKTRIKI ANONYMI ETAIREIA	HERON	GR	HERON
13	ZELENA ENERGETSKA ZADRUGA ZA USLUGE	ZEZ	HR	ZEZ Zelena Energetska Zadruga
14	STUDIO TECNICO BFP SOCIETA A RESPONSABILITA LIMITATA	BFP	IT	BFP
15	VEOLIA SERVICIOS LECAM SOCIEDAD ANONIMA UNIPERSONAL	VEOLIA	ES	VEOLIA
16	RIGA MUNICIPAL AGENCY "RIGA ENERGY AGENCY"	REA	LV	REA Riga Energy Agency
17	FUNDACION ASTURIANA DE LA ENERGIA	FAEN	ES	FAEN Fundación Asturiana de la Energía

Executive Summary

Data-driven approach to artificial intelligence in energy systems comes with a need for careful pipelining of ingested or historical data and developed models. A considerable amount of time is often spent in data management before the actual model development task takes place. Machine learning and deep learning model development, deployment and operation usually is not a straightforward task in any project or business process. Another challenging factor in ML/DL development is the fact that historical data might be available for modelling but after deployment, live or real-time data will come with a slight shift in its distribution from the ones used in training, which might require the model to be retrained as the performance of the model declines. Similarly, for some tasks, it is often the case that data available for training is not adequate. Further, transfer learning can also be employed to create good models even when there are only few data samples available for the particular problem/task. There is also issue with data that is not fully available for model training but instead comes in batches. This can be solved by using incremental learning approach.

The high volume of data for model training calls for a need to employ distributed training techniques during model development to reduce training time and achieve better performance, thanks to breakthroughs in high performance computing devices or accelerators. It is also necessary to ensure interoperability of different models developed by using different frameworks or programming languages based on developer preferences or use cases. Similarly, the models need to be regularly evaluated to determine when to retrain the models to maintain the original performance. Effective model serving framework is also a necessity while ensuring confidentiality, integrity and access control to the models.

In this first release of *I-NERGY Models, AI Methods and Tools*, we present various AI methods, tools and techniques to realise and deploy various AI models for use by various I-NERGY AI services for different use cases.

ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AI DevOps	Artificial Intelligence Development Operations
AGI	Artificial General Intelligence
CI/CD	Continuous Integration/ Continuous Delivery
C-MAPSS	Commercial Modular Aero-Propulsion System Simulation
R^2	Coefficient of Determination
CT	Continuous Training
DPDA	Data Parallelism Deployment Algorithm
DL	Deep Learning
DistDGL	Distributed Deep Graph Library
DDLBench	Distributed Deep Learning Bench
FPGA	Field Programmable Gate Arrays
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HPC	High Performance Computing
IMG	Intermediate model generator
IRL	Inverse Reinforcement Learning
KPI	Key Performance Indicators
ILP	Linear Programming
LSTM	Long Short-Term Memory
ML	Machine learning
MAE	Mean Absolute Error
MPI	Message Passing Interface
MLOps	Machine Learning Operations
NNEF	Neural Network Exchange Format
ONNX	Open neural network exchange
RL	Reinforcement Learning
RAE	Relative Absolute Error
RSE	Relative Squared Root
ROC Curve	Receiver Operating Characteristic Curve
RMSE	Root Mean Squared Root
RUL	Remaining Useful Life
TPU	Tensor Processing Unit
VAE	Variational Autoencoder

Table of Contents

Executive Summary

ABBREVIATIONS AND ACRONYMS

Figures

Tables

1 Introduction

1.1 Purpose

This deliverable presents progress made regarding I-NERGY models, AI methods and selected tools to develop, train, evaluate and securely serve I-NERGY trained models for various I-NERGY services titled "I-NERGY Models, AI methods and Tools (1st technology release)". The deliverable will particularly focus on activities relating to Task 3.3 Incremental, Distributed & Self-learning Model Definition and Training, Task 3.4 Cross-stakeholder Transfer Learning, Task 3.5 Model Evaluation and Serving Framework, and Task 3.6 Security Framework.

First, we present technical backgrounds relating to models to be developed and utilised within the I-NERGY project. Next, we follow this with detailed overview of each component, its requirements as well as the proposed approach, technologies, tools to realise it within the context of I-NERGY project. We also discuss the justifications behind the choices of technology, tools etc. selected. Finally, we draw a conclusion on the way forward.

1.2 Structure of the Document

The document is structured as follows:

- Section 1 presents the purpose and the structure of the deliverable.
- In Section 2, we discuss the related technical and theoretical background relating to AI Model development, training, evaluations, and secure model sharing. To be specific, the following sub-topics: ML and ML pipeline, incremental ML, self-learning techniques in ML, distributed model training, ML inter-operability, transfer learning, model evaluation and serving techniques, model updating, access control techniques for data and trained models are discussed in this section. A particular emphasis is laid on each of these sub-sections regarding their meaning, existing approaches for their realizations (including pros and cons where appropriate).
- Following this is Section 3, which presents a detailed overview of each of the following components: Incremental, Distributed and Self-learning Model, Cross-stakeholder Transfer Learning, Model Evaluation and Serving Framework, Security Framework for Data and Trained Models. These are the key components needed to define, train, evaluate, and securely share I-NERGY trained models for various AI services needed in the pilot use cases. For each component, we present its purpose, requirements, proposed approach, required tools and justifications.
- Finally, Section 4 presents conclusion of the report and a glimpse into the expected changes in the next version of the document.

2 Background

In this section, we discuss general background, existing approach (including pros and cons where appropriate) for related terms and technologies to be used to realise the AI Trained Model Library of the I-NERGY project.

2.1 ML/DL Pipeline

Machine learning (ML) and deep learning (DL) in particular have greatly improved state-of-the-art in many application areas such as computer vison, natural language processing, computer games, robotics, etc. Thus, the reason why the field has been the preferred choice of many researchers in recent years.

It has been reported that most of the current effort in the area of ML/DL has often focused on model development and training while disregarding issues relating data handling, model serving and entire lifecycle management. [1] notes that DL modelling lifecycle results in the generation of a rich set of data artifacts, for instance, learned parameters and training logs comprising of lots of frequently conducted tasks such as those performed to understand the model behaviours and to evaluate new models. Handling of such artifacts and tasks is tedious and most times poses challenges to the users.

ML/DL pipeline helps to manage various stages and tasks involved in ML/DL development, deployment, and updating in a systematic manner. Unlike traditional software systems in which the greater percentage of development task is spent on coding, ML/DL modelling comprises various steps that even take much more time to accomplish than the actual coding part. Against the expectation of many people, it is only a very little fraction of code found in many ML systems that is actually used for learning or prediction. This is succinctly captured in Figure 1.

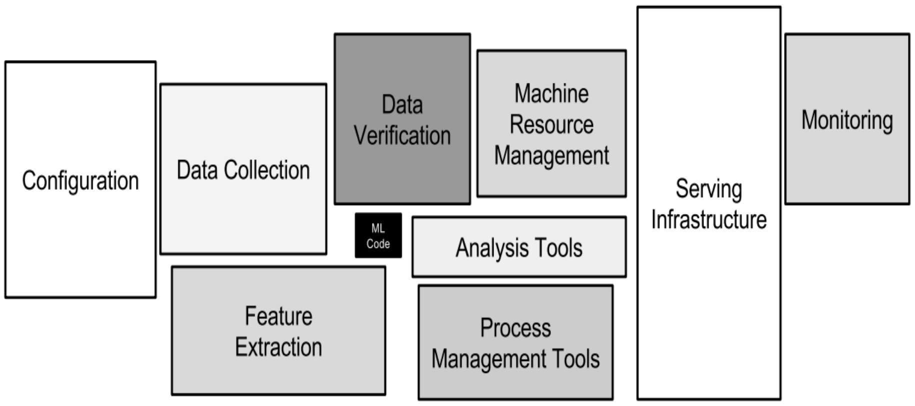


Figure 1 Diagram showing components of ML/DL systems [2]

The size of each block in the above diagram implies the proportion of time devoted to accomplish the tasks involved in it. This means that setting up model serving infrastructure, configurations, data collections etc. occupying bigger spaces in the diagram are the tasks that ML/DL experts spend most of their time on during the lifecycle of a model.

2.1.1 Model Development Workflow

Generally, ML model development involves several steps ranging from data collection to model deployment. A typical ML model development workflow is illustrated in Figure 2. It essentially consists of six (6) steps which are briefly discussed hereunder. It should be noted that this is just a typical way of developing and deploying ML models and thus, does not imply the only way as the steps can be further expanded or reduced (with some steps placed together).

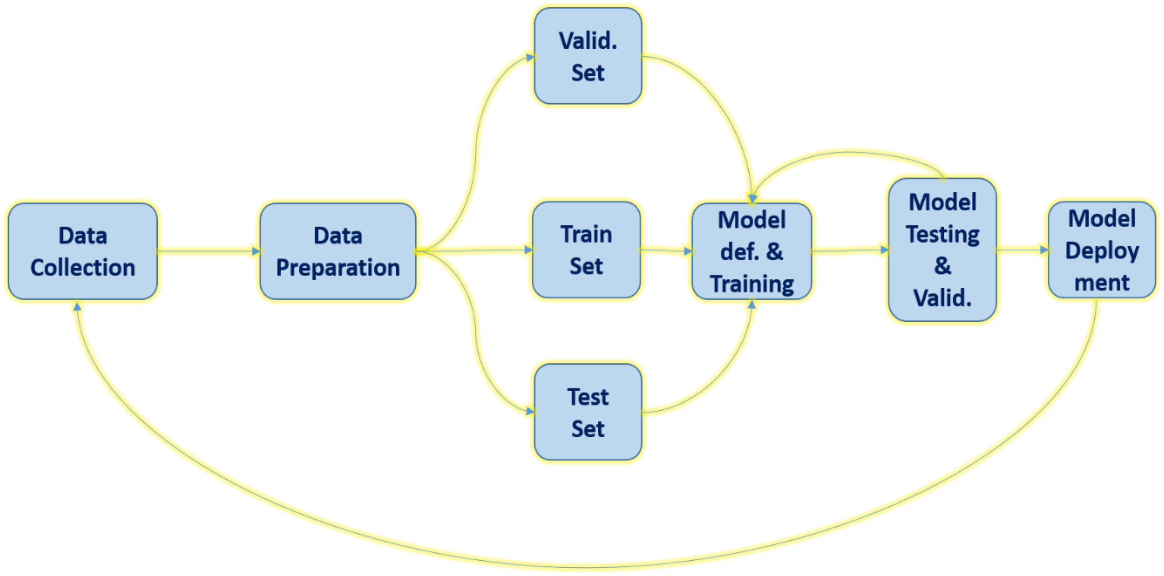


Figure 2 ML Development Workflow

Data collection: This is the first step in ML model development. In this step, the involved data scientists, or persons with the domain knowledge of the business define the problem to be solved and the data to be used for model development.

Data preparation/feature engineering: In this step, the collected data is transformed, cleaned, labelled and enriched to improve data quality for use in training models. Once data is prepared, feature engineering can then follow. Features refer to data values that the model will use both in training and in production. The data is then explored to define which attributes have the most predictive power, to finally arrive at a set of features. This is a very important step in classical ML, however, for DL models, feature engineering is basically not necessary.

Algorithm Selection: This step is done in line with the previous steps because the selection of which ML algorithm to use is one of the initial decisions in ML. At the heart of any model is a mathematical algorithm that defines how a model will find patterns hidden in the data.

Model training: Model training is the core part of the whole process. To train the model to be able to perform predictions on new data, data scientists/ML developers first fit the model to the processed dataset.

Model Testing and validation: Finally, trained models are tested on testing and validation data to ensure that they provide high predictive accuracy on data samples it was not originally trained on. Training and evaluation are iterative phases that are continuously carried out until the model achieves an acceptable performance.

Deployment: The final stage is deployment of the ML model in the production environment. This is the last step performed after which the end user can use the model to generate predictions on live data. Note that at this stage, data is also collected from the deployment model to identify when it starts degrading in performance. Once the deployed model has been confirmed to be performing below the expectation, it is then retrained. This often results in creation of different models out of which the best performing model replaces the existing deployed model.

2.2 Incremental Learning

One of the major issues with current deep learning models is their inability to learn in a continuous manner as humans. This is one of the greatest challenges towards attaining artificial general intelligence (AGI), i.e., to realize an AI system that can express (supra-) human-like intelligence that has been gaining strong interest in the research community in recent years. Incremental learning techniques that result in models with the ability to improve their performance as more data samples are seen, could be leveraged to bridge this gap.

2.2.1 Motivation and Challenges to Incremental Learning

There are many practical reasons why one may want to develop models that will be able to perform continuous learning. Key among these reasons are as follows:

- Data is coming in a streaming fashion
- We do not have infinite memory space for data
- Batch machine learning in production is hard
    - Models have to be retrained from scratch with new data
    - Increasing computational requirements
    - Features developed locally may not be available in real-time
- Incremental learning algorithms can learn from one observation at a time and observations do not necessarily need to be stored
- Features and labels may be dynamic

Despite the benefits of employing incremental learning strategy, it is also worthy of note that incremental models are subject to the following constraints:

- The model does not need to review all past knowledge whenever it tries to learn new facts.
- The model needs to learn continuously without being affected by catastrophic forgetting, i.e., forgetting previously learned facts.
- Most of the existing libraries do not support deep learning

2.2.2 Incremental Learning Strategies

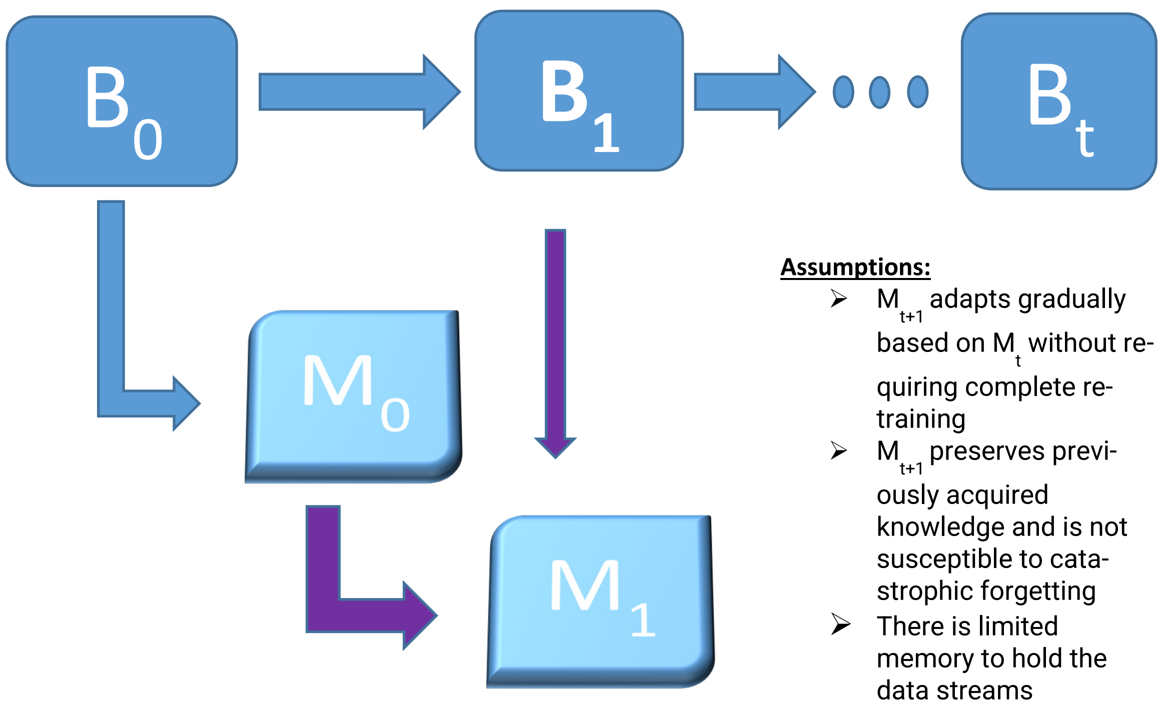
Herein, we present the major approaches for performing incremental learning.

- Online learning
- Class-incremental learning

We also mention that another major approach is *hybrid incremental learning* (involving a combination of online learning and class-incremental learning). Note that this approach is merely included for the sake of completeness and we will not elaborate much on it in the foregoing.

2.2.2.1 Online Learning

This incremental learning approach is generally referred to as learning new data of known classes. In this approach, the model is updated as new batches of data arrive. This is illustrated in Figure 3 below. The initial model M0 is trained with initial batch of data B0 at time t0. This process is repeated as time t goes from 0, 1, 2 … t.



Assumptions:

- M_{t+1} adapts gradually based on M_t without requiring complete retraining
- M_{t+1} preserves previously acquired knowledge and is not susceptible to catastrophic forgetting
- There is limited memory to hold the data streams

Figure 3 Illustration of Online Learning

2.2.2.2 Class-Incremental Learning

In this approach, the model will be exposed to data containing new classes with the aim of making sure the model remembers old classes. After each task, the model is trained by using all seen classes via a separate test set. As shown in Figure 4 below, each step produces new evaluation score (accuracy for example). As reported in [2], the final score is obtained by averaging all previous task accuracy scores in what is referred to as **average incremental accuracy**.

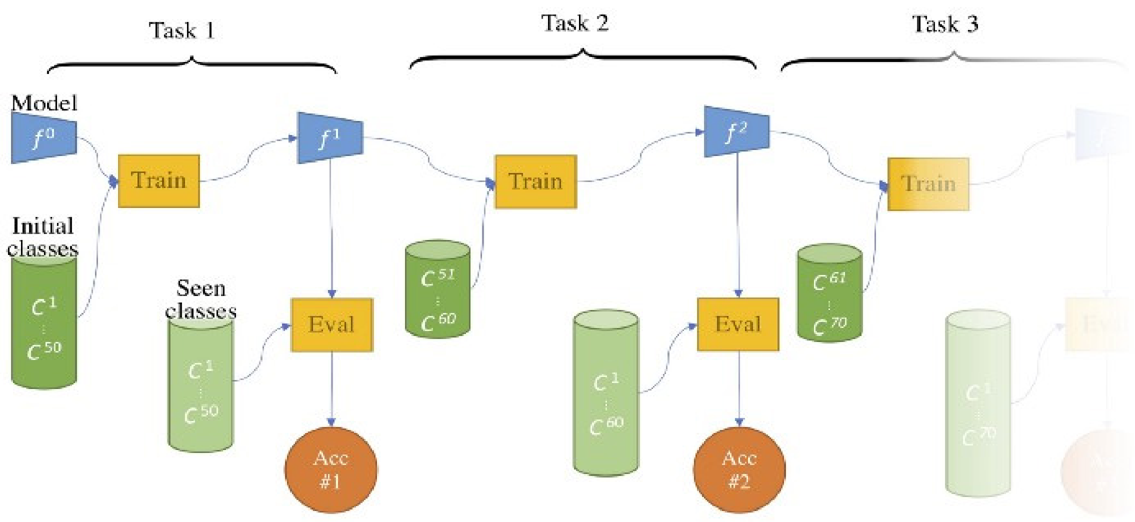


Figure 4 Class incremental Learning [4]

2.2.3 Comparison of Some Existing Incremental Learning Methods

Many incremental learning methods abound in the literature. However, we are not going to present a thorough review of all the methods, instead, we will only introduce some of them that we consider most relevant for use within I-NERGY. Table 1 shows a comparison of these methods by considering the algorithm type, methodology employed, technique employed and the related application areas.

Table 1 A Comparison of Incremental Learning Approaches

Author & Year	**Algorithm Type(s)**	**Methodology**	**Technique**	**Application Area(s)**
Li, P et al., (2018) [3]	Deep Learning	Learns features for use in incremental learning on industrial big data and updates the rule of tensor layers via two incremental learning algorithms	Employs deep convolutional model, parameter incremental and structure incremental algorithms	Industrial Big Data
Liu, J et al., (2018) [4]	A combination of Deep Learning and Machine Learning	Improves performance in fault diagnosis through dynamic and incremental compensation of the deep learning model	Denoising Autoencoder, Similarity computation and Support vector machine	Fault Diagnosis in industrial settings

Yao, C et al., (2019) [5]	Machine Learning	Learns distribution of new classes by employing Gaussian kernels for incremental class learning	One Class Support Vector Machine (OCSVM)	Class incremental learning tasks
Yang, Z et al., (2019) [6]	Neural Network	Detects concept drifts via Online Sequential Extreme Learning Machines (OS-ELMs)	Extreme Learning Machine	Energy Production Forecasting for Wind Plants
Li J et al. (2019) [7]	Machine learning	Solves time series forecasting problems by updating the weights of samples in a double incremental learning algorithm	SVM	Time series forecasting
Yu W, and Zhao C, (2019) [8]	Deep Learning	Improves accuracy of diagnosis task by extracting both nonlinear structure and fault tendency from data matrix of the consecutive samples	Broad Convolutional Neural Network (BCNN)	Industrial Fault Diagnosis

2.3 Self-learning in ML

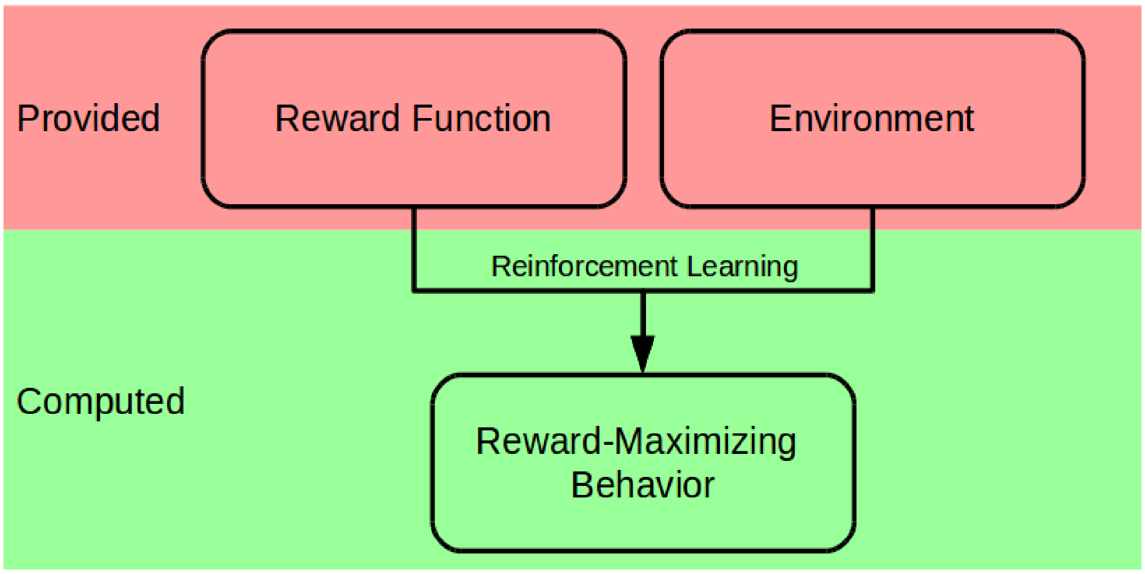
It is a well-known fact in the research community that contemporary machine intelligence has hitherto remained far from realising the main hallmarks of human understanding and consciousness. The main issues with current methods are attributable to the difficulty or implausibility of foreseeing and pre-programming each and every piece of information or knowledge [9]. Emergent intelligence approaches based on self-learning techniques and self-organization have been successfully employed in fusing understanding capabilities in machines. This understanding is different when compared with the constrained intelligence exhibited by machines employing classical approaches of intelligence via supervised knowledge acquisition techniques.

Particularly, self-learning systems are adaptive systems that can improve their performance through a learning process, mainly based on trial and error approach. A self-learning system first interacts with its users or the environment of interest and observes the changes produced due to its actions. Self-learning paradigms have been applied in many application fields such as autonomous vehicles, banking, finance, gaming, healthcare and robotics with huge benefits.

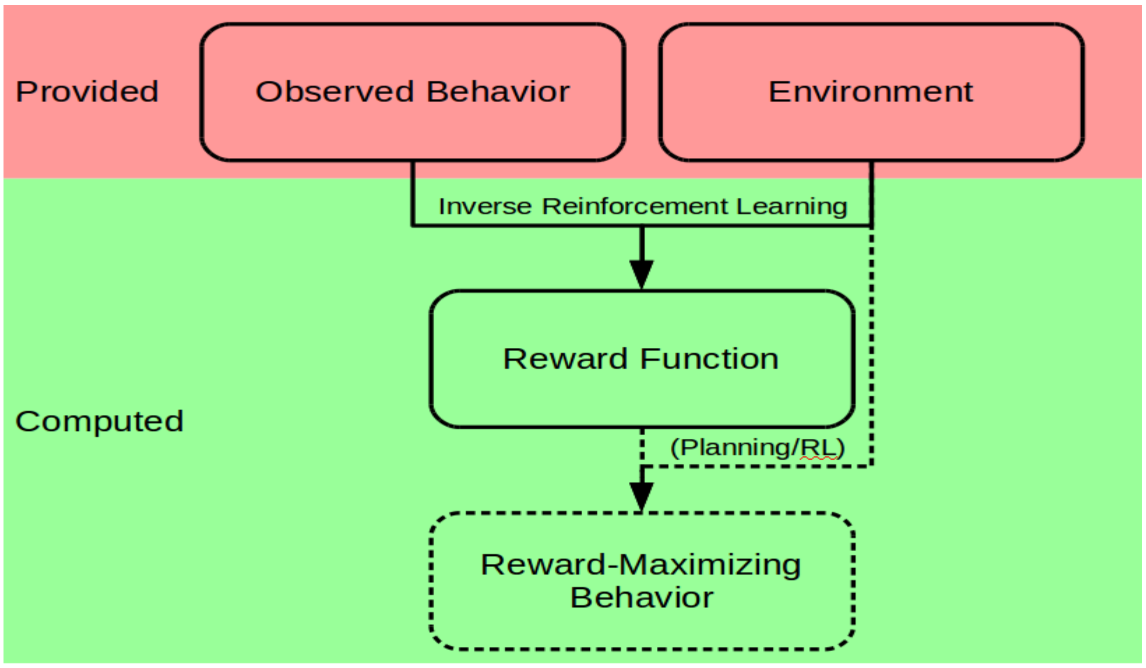
2.3.1 Self-learning Enabling Technologies

Development of self-learning systems is made possible through the use of many AI technologies such as reinforcement learning, inverse reinforcement learning, learning-by demonstration etc. In reinforcement learning (RL) setting, the reward received at a given time step depends on the state of the environment at the last time step (referred to as s), the state at the current time step (s'), and the action that the agent just carried out (*a*). The reward at each time step is determined by using a *reward function R* (which, given some *s,a,s'*, returns a single number *r=R*(*s,a,s'*), i.e. the current reward). In general, an RL agent meanders through its environment, performing different actions and receiving rewards and punishments based on the reward function. As it does so, it gradually learns to perform the right actions in the right states to maximize its *long-term discounted reward*.

One of the key assumptions in RL is that the reward function is specified by someone else, and then the agent performs the actions. An interesting question is whether the agent could instead watch someone else perform the actions, and then try to come up with its reward function. Because this inverts the standard RL process (reward -> behavior vs behavior -> reward), it is referred to as *inverse reinforcement learning (IRL)*. It is also referred to as *apprenticeship learning* (the agent seen as an apprentice observing its master perform some tasks). Because the reward function captures what the programmer really wants to realize, being able to infer a reward function from behaviour is like determining the goals of those whose behaviour you are observing. Thus, the agent could watch good behaviour, use this to deduce a reward function, and then employ traditional RL to maximize that reward function in the long term [10]. Figure 5 shows the steps involved in both RL and IRL.



(a) Reinforcement Learning Setup



(b) Inverse Reinforcement Learning Setup

Figure 5 Reinforcement Learning and Inverse Reinforcement Learning Setups [12].

Figure 5 (a) shows that in a reinforcement learning setting, both the reward function and the environment are provided while only the reward maximising behaviour is computed. However, in (b) the inverse reinforcement learning setting is depicted, illustrating that the observed behaviour by the agent and the environment are provided while the reward function and reward-maximising behaviour are computed.

2.4 Distributed Model Training

During the training stage in the ML pipeline, as stated in section 2.1, it is pertinent to ensure that training time shortens to the barest minimum while ensuring optimum performance especially during initial training, testing and evaluation of the model as well as during deployment. This will allow for proper diagnostic analytics in maintenance, predictive analytics in forecasting or prescriptive analytics in simulation or digital twins application, depending on the use case. Use cases involving deep learning model training take a very long time for training due to large volume of data and multiple layers of the neural network model.

Research from industry and academia has made breakthroughs in distributed computing devices and message passing interface algorithms. Also, the efficiency of distributed computing has been made using heuristic methods in parallel software developments [11]. In addition, network bandwidth utilization, management, and optimization of the deep learning algorithm leads to faster training time with high performance during training and deployment. These breakthroughs allow for new ways of handling **data**, **model** and **parameters** across the high-performance computing devices by the use of optimized deep learning algorithms. Eigen decomposition using FPGAs in [12] and task scheduling techniques in [13] are many novel and innovative ways of aiding distributed computing for ML and DL research and application. These new ways of training deep learning models involve distributing or parallelising functions during the training stage of the ML or DL pipeline and will be discussed on the subsections below.

2.4.1 The Need for Distributed Training

Fundamentally, in ML and DL applications, we often have to deal with large datasets and complex models. For the complex models, they have to deal with huge parameter computation with variables like gradients and activations passed between model layers and computing nodes to reduce loss function and update the weight and bias aspect of this parameter for each iteration. During the model training, the feedforward and the back propagation algorithm for calculating this gradient efficiently takes a lot of computing resources. The computation of the average cost or minimising the cost function during the deep neural network learning takes a lot of processor time. There is also a possibility of having to deal with graph data based on semantically annotated data in a different or the same database of a power network depending on the data pipeline architecture. Therefore, the iterative convergent algorithm, involving matrix and vector based parameters of the very deep neural network which is a changing function of large dataset, need to be parallelised or distributed [14]. However, the aforementioned task is always complex and often overwhelms the computing resources of a single node or device; hence the need for distributed computing. Computing resources like memory space, CPU time, storage capacity and network bandwidth play a big role in distributed training of deep learning applications. Machine learning and deep learning model training has gone into multi-node parallelization, thanks to advancements in HPC. There is move from the traditional shared memory method of loading dataset and model in a single processor, to achieving a trained model by the new distributed resources method of sharding of dataset and/or model in multicore distributed machines such as GPU, TPU or NVIDIA hardware accelerators.

The following motivate distributed model training:

- To train very large data set or complex model using different computational resources
- To reduce the time in model training which can take days or weeks
- TPUs and modern GPUs have very high memory space
- Big data with velocity, volume, veracity and variety characteristics brings high complexity

2.4.2 Distributed ML/DL Training Methods

There are several approaches that have been developed for distributed training based on the application or use case of the project. The methods of training also depend on the utilization of the computing resources and the algorithm of use. These approaches help to achieve a faster convergence of the cost function for the parameters in the training epochs. There are about four basic methods of distributed training:

- Model parallelism
- Data Parallelism
- Graph Parallelism

- Hybrid Parallelism

The chosen method will depend largely on the dataset, model size, training framework, type of neural network and the hardware target.

2.4.2.1 Model parallelism

Model parallelism is the technique of splitting the model into sub models consisting of different layers of the model on different machines or hardware accelerators. In model parallelism, all devices or nodes participate in the forward pass and backward pass of the model parameters between each layer of the model. Starting with the full training dataset in the first layer of the input machine, the device computes the gradient and activation and sends the activation of its final layer to the next device's input layer. This process repeats until the activations and gradients are updated across the complete model layers, and it is then reversed for the backward pass. Sequential training is involved in model parallelism due to the parameter dependencies in between devices during the forward and backward pass. The forward pass and the backward pass occurs in different devices or machines for each iteration or epoch.

A single training epoch transverses multiple nodes to achieve preliminary accuracy in each machine before getting final model accuracy in the final machine. In model parallelism, the distribution algorithm has to provide a Message Passing Interface (MPI) to all the machines involved in the model parallelism to update the training parameters between the layer(s)[15]. The decomposition of a complex model makes a model to be faster in training which will give rise to a more relevant prediction in a given use case. Figure 6 illustrates model parallelism.

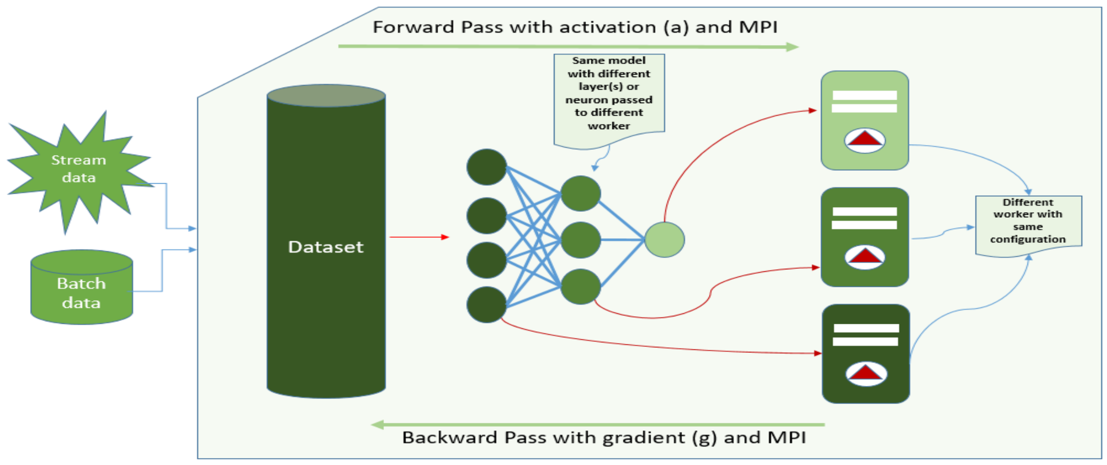

Forward Pass with activation (a) and MPI
Same model with different layer(s) or neuron passed to different worker
Stream data
Batch data
Dataset
Different worker with same configuration
Backward Pass with gradient (g) and MPI

Figure 6 Model parallelism

2.4.2.2 Data parallelism

Data parallelism is the technique of splitting the dataset into different batches. This is to cope with the large volume of data or big data ingested into the system. Each machine will have a local replica of the same model and will train on the sliced dataset. The batch data is assigned to each device or accelerator and computation is carried out for each iteration in the local device. During this iteration, the computing devices exchange their local gradients and activation values to a parameter server to calculate the global values using a sequential averaging algorithm and then update the model. This algorithm is known as Ring-Allreduce and fully explained in [16].

The same model that trains over different shards updates the model information between the machines through the MPI. Low latency during the communication of the gradient and activation parameters between the devices is crucial to achieving a very good training time as the computation is shared among the computing resources. In data parallelism, the forward pass and the backward pass iteration in each epoch occurs in one local worker or machine.

Data parallelism can also be optimized in edge computing devices using an efficient Integer Linear Programming (ILP) and data parallelism deployment algorithm (DPDA) as proposed by [17].

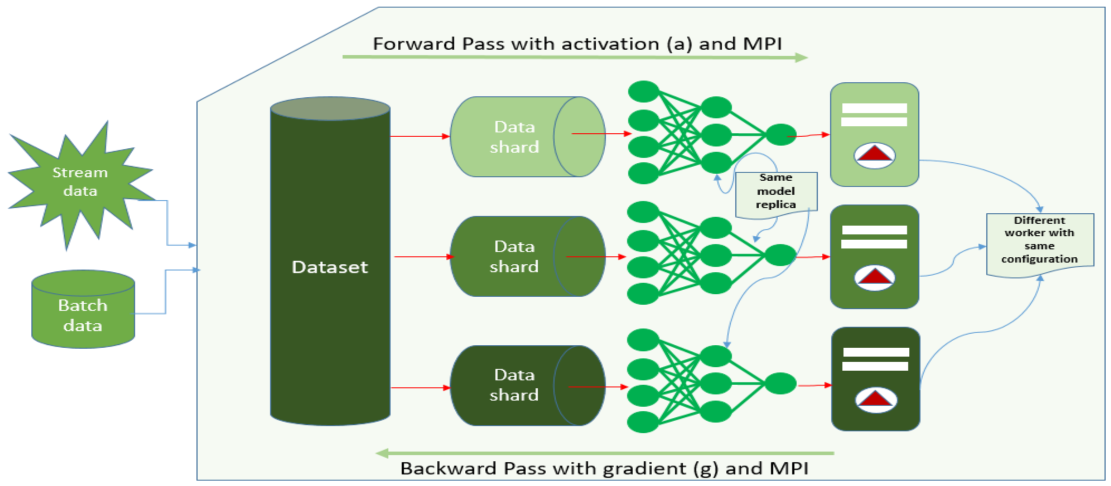

Forward Pass with activation (a) and MPI
Stream data
Batch data
Dataset
Data shard
Data shard
Data shard
Same model replica
Different worker with same configuration
Backward Pass with gradient (g) and MPI

Figure 7 Data parallelism

2.4.2.3 Hybrid parallelism

Hybrid parallelism is a type of parallelism involving the mixing of the mentioned data and model parallelism or even introducing a different customized type based on the use case of the project; e.g. Pipeline algorithm mentioned by [18]. This is applied based on the algorithm developed from the use case and the service provisioning for the project or business model. It also depends on the computing resources and type of data available or transformed during the staging part of the data pipeline. In [19], the authors proposed an inference engine algorithm for large sparse DNN using task graph parallelism with data and model decomposition. Also in [20], the authors proposed an efficient distributed graph embedded system which can utilize GPU clusters to train large graph models with billions of nodes and trillions of edges using a mix of data and model parallelism. Figure 8 shows the hybrid parallelism for a use case with large data set and very complex model.

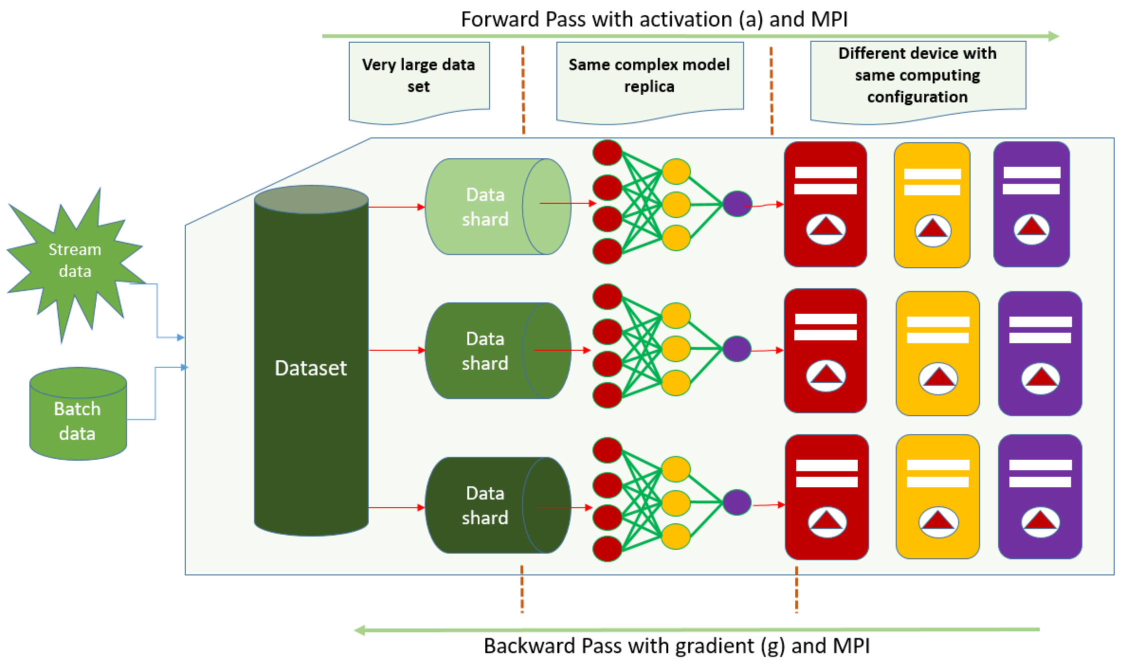

Forward Pass with activation (a) and MPI
Very large data set
Same complex model replica
Different device with same computing configuration
Stream data
Batch data
Dataset
Data shard
Data shard
Data shard
Backward Pass with gradient (g) and MPI

Figure 8 Hybrid parallelism

2.4.2.4 Graph parallelism

Ontological or semantic data with linked data structure for knowledge graph and knowledge representation can also be parallelised. Graph datasets can serve as an input to a Graph Neural Network (GNN). There might be the need to shard graph data to different GPUs to carry out GNN training. Graph parallelism is the technique of splitting a graph dataset or graph structure into different machines with the same model to cope with the semantic data or linked data structure for ontological data model. This method is necessary to train over large graph data. The data is partitioned as a graph that is associated with every vertex and edge. Graph parallelism method involves the application of update function as an operation on the graph vertex and then transforms the data in the scope of the vertex [14]. This will definitely require HPC nodes with each node having multicore GPU, TPU and VPU. In recommendation and anomaly detection systems the use of distributed training in graph neural network can be optimized by using Distributed Deep Graph Library (DistDGL) which is a system for training GNNs in a mini-batch technique across a cluster of machines [21]. Figure 9 demonstrates a typical graph parallelism setup.

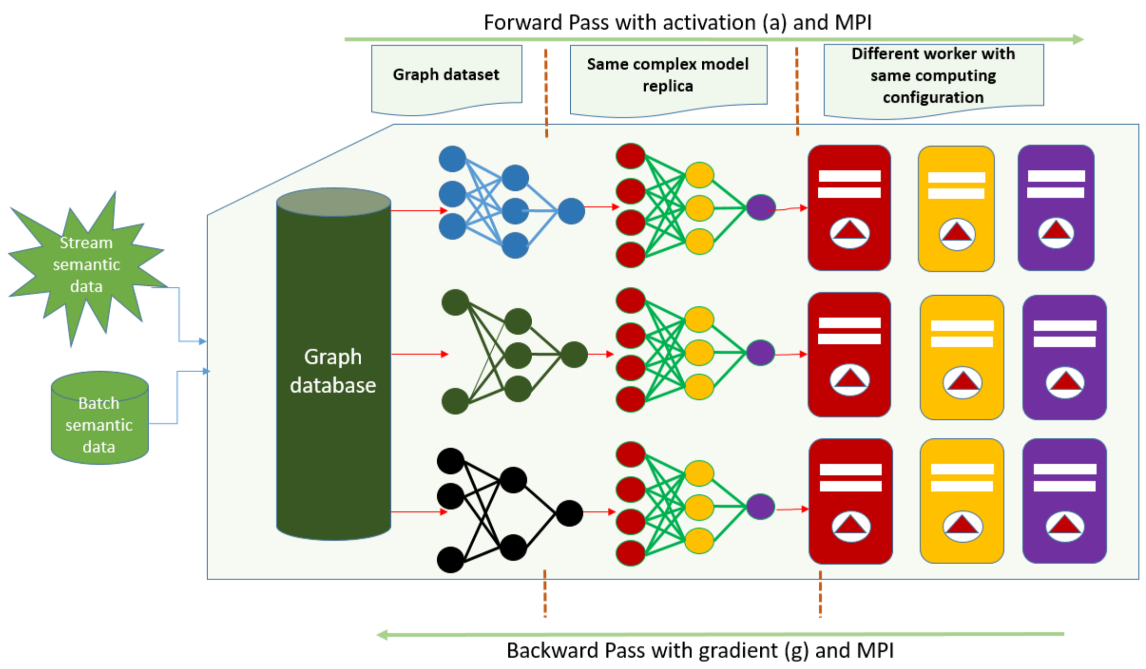

Forward Pass with activation (a) and MPI
Graph dataset
Same complex model replica
Different worker with same computing configuration
Stream semantic data
Batch semantic data
Graph database
Backward Pass with gradient (g) and MPI

Figure 9 Graph parallelism

2.4.3 Performance in Distributed Training

In various research communities, different benchmarking tools are used to measure the performance of distributed training or distributed deep learning. For a deep learning application development and deployment, the design, implementation and performance analysis comes with five **degrees of freedom** [22]:

- Dataset size
- Neural network model
- Distribution training model
- Deep learning framework
- Node or Device type

There could also be a sixth degree of freedom regarding network bandwidth since it will affect weight and activation updates between nodes and model layers thereby affecting the convergence time based on the MPI algorithm between the device for very large or complex models or large datasets.

These degrees of freedom will affect the key performance indicators (KPI) in deep learning training. In deep learning the most important key performance indicators as reported in [18] are:

- Training time
- Test Accuracy
- Memory usage

Training accuracy as KPI depends on the application and is highly affected by the quality of the dataset. The type of model as well as the hyperparameters can also have an effect on the KPI. Therefore, for distributed training, it is best to consider training time and memory usage since that depends largely on the device resources: processor speed, memory size, network bandwidth; which is the essence of distribution technology.

Distributed Deep Learning Bench (DDLBench) was used as a benchmarking suit tool to measure performance between three different implementation algorithms. These algorithms include [23]:

- Horovod
- GPipe
- PipeDream

Horovod is based on data parallelism while GPipe and PipeDream are hybrid parallelism methods involving model parallelism and pipeline parallelism [23]. A comparison of the performance of Horovod, GPipe and PipeDream using the following datasets: MNIST, CIFAR-10, ImageNet, and Highres and the following models: ResNet-18, ResNet-50, ResNet-152, VGG-16 and MobileNet v2 are shown in Figure 10. From the figure, it is obvious that Horovod has the best performance for the DDLBench benchmark suit. Hence, Horovod was selected for the I-NERGY project. Figure 10 shows the chart comparing the training time for the three mentioned algorithms implemented with Pytorch [24] with different GPU configurations. This choice was also corroborated in [16] where the authors outlined the advantages of Horovod and the implementation guidelines which will be mentioned in section 3.0.

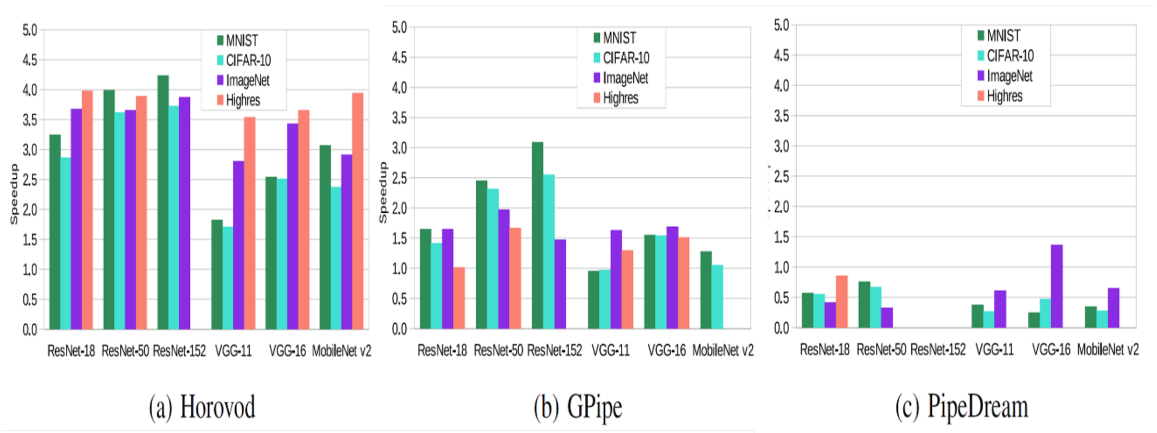


Figure 10 A comparison of the performance of Horovod, GPipe and PipeDream [23]

2.5 Machine Learning Inter-Operability

Interoperability between programming languages or development frameworks in machine learning and deep learning, during algorithm training and model selection and deployment in a production environment, is a key challenge in AI application. There is a need for interoperability between ML and DL models to reduce the deployment time of AI solutions. This is because these models are usually developed and optimized in different frameworks or toolkits based on the use cases. In addition, the use of various programming languages leads to very segmented or fragmented models when they need to work together in deployment scenarios. Models developed with different platforms and hardware accelerator targets, most especially in a full AI project implementation, depends on the service request at the serving or analytics layer [25]. There are many tools and techniques available for ML interoperability or conversion. The best tool will be an open standard for Machine Learning, Deep Learning and Knowledge Graph interoperability. Most of the Problems DevOps face in deployment include.

- Very high inference latency in model (ML and DL) production
- Train a model in one language and deploy in an application with different programming language. For instance, training a model in Python and deploying it in Golang.
- The ability to run the same model in different platform and hardware accelerator
- To run models on edge and device IoT hardware
- The need to create models in different framework and run in a platform of choice
- The need to shorten the training time of very large models

From research or findings for ML interoperability tools that will enable ML developers to make use of their preferred framework without bothering on the format for deployment and to standardize the conversion of ML models from different training frameworks, two fundamental methods have been identified as the best standard supported by a large industry and academia ecosystems. These are:

- Open Neural Network Exchange (ONNX)
- Neural Network Exchange Format (NNEF)

2.5.1 Open Neural Network Exchange

Open neural network exchange (ONNX) is an open-source tool representation format converter and runtime inference engine for machine learning, deep learning and transfer learning model pipelining which was initiated by Facebook and Microsoft and later was joined by Amazon. The ONNX toolkit can help AI DevOps engineers to easily transfer deep learning models between different frameworks to suit their solution requirements. The tool is supported by leading companies around the world making ONNX to have a huge ecosystem of optimizers, visualizers, frameworks, compilers runtimes and accelerators compatibility. ONNX has the converter part and the runtime part. The ONNX common format converter and ONNX Runtime inference engine makes it a very high performant tool across platforms and hardware accelerators [26] as shown in Figure 11.

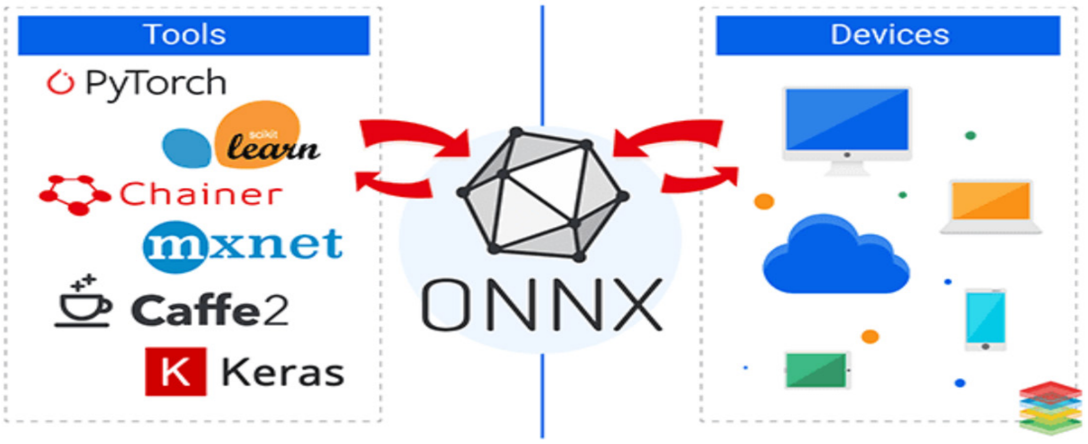


Figure 11 ONNX Format Converter to different deployment platform [26]

ONNX runtime is compatible with PyTorch, TensorFlow, Scikit-learn, etc. The trained format will not have to be changed if you use ONNX the native training tool. An ONNX solution guarantees a 2.5X performance improvement more than individual conversion techniques for each framework. ONNX, amongst the others, allows developers to utilize their favourite ML or DL development framework thereby reducing interoperability issues, streamlining the path from research to production, and increasing the speed of innovation. The ONNX tool can also help developers to carry out continuous development and continuous integration during each sprint of their innovation lifecycle. ONNX is an open source tool that enables AI developers to adapt their framework to custom target devices like FPGA for easy reconfiguration using a cross platform approach as proposed in [27].ONNX defines an extensible computational graph model which includes built-in operators and standard data types. ML models that have been trained on various frameworks to suit different data from various use cases can be converted to the ONNX format using tools such as TensorFlow-ONNX, KERAS, Scikit-Learn, CoreML, etc. Figure 12 Shows the different frameworks and the different hardware accelerator targets.

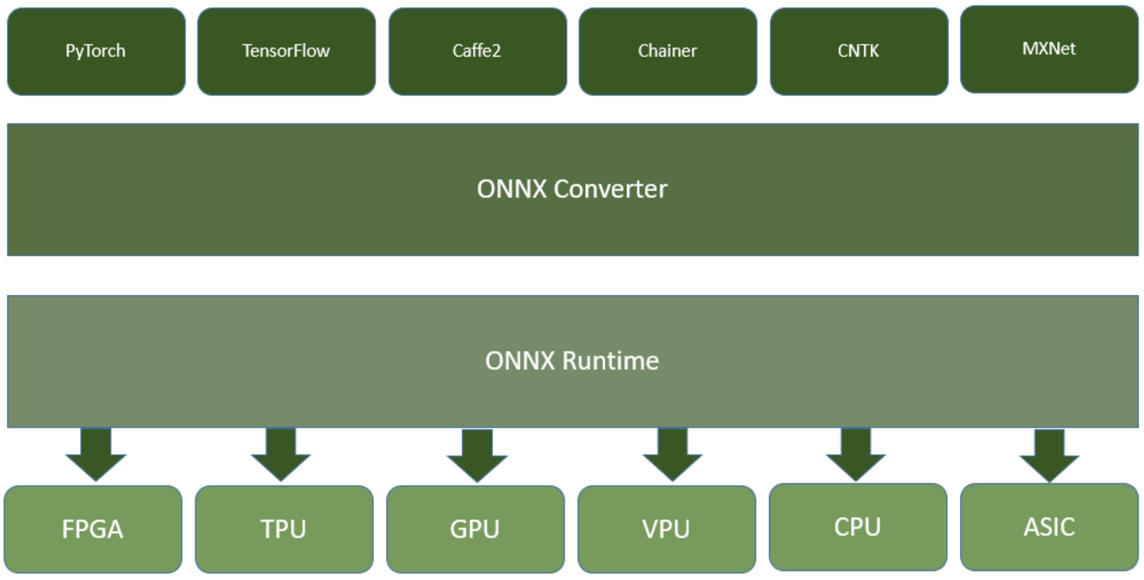


Figure 12 ONNX Runtime to different deployment Hardware accelerators

2.5.2 Neural Network Exchange Format

NNEF is a exchanged format which started out as OpenVX tool by Khronos group before being fully being standardized by a large ecosystem of academia and industry through a working group within Khronos[28]. In [29] they implemented a neural network model converter using NNEF to enable it to run the Darknet neural network engine or framework. In a bid to also standardize NNEF and make it applicable to more framework or neural network engine, [30] has implemented the converter algorithm for TensorFlow using a top-down parsing in deep learning computation graph. Figure 12 shows the frameworks supported by NNEF for training and inferencing interoperability. Figure 13 shows how NNEF enables NN network training and inferencing.

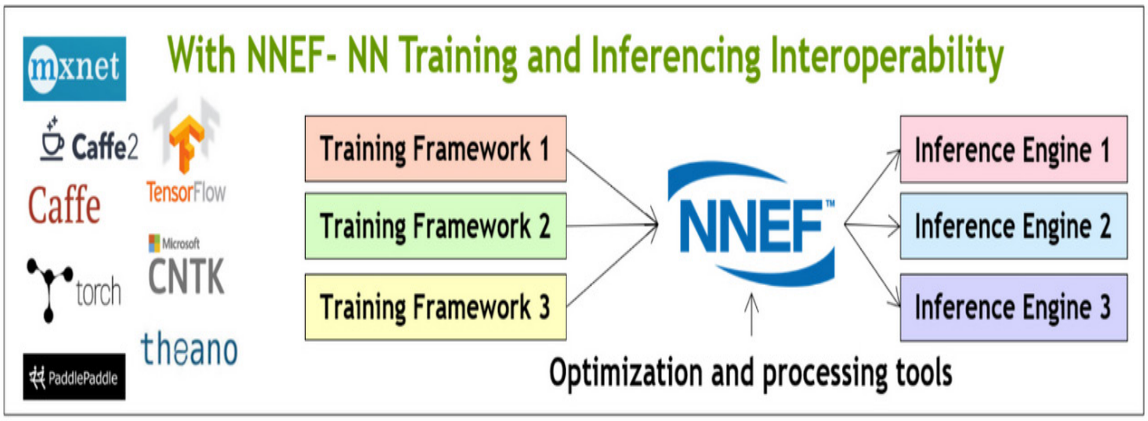
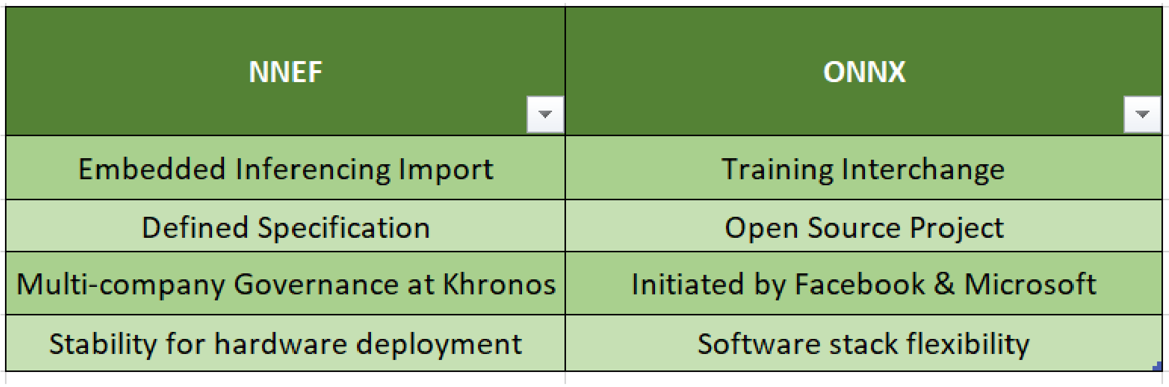


Figure 13 NNEF training inferencing and optimization tool [28]

2.5.3 Comparison Between ONXX and NNEF

ONNX and NNEF has their strengths and weaknesses as well as similarities and differences. ONNX and NNEF are somewhat complementary in the sense that ONNX is fast in keeping record of originating framework updates while NNEF ensures stability connection from training ML models to inferencing computation in gateway's hardware[28]. Figure 14 below highlights the key differences between ONNX and NNEF. It is also worthy of note that there is a conversion interface between ONNX and NNEF as showed in Figure 15.



NNEF	ONNX
Embedded Inferencing Import	Training Interchange
Defined Specification	Open Source Project
Multi-company Governance at Khronos	Initiated by Facebook & Microsoft
Stability for hardware deployment	Software stack flexibility

Figure 14 Key differences between ONNX and NNEF

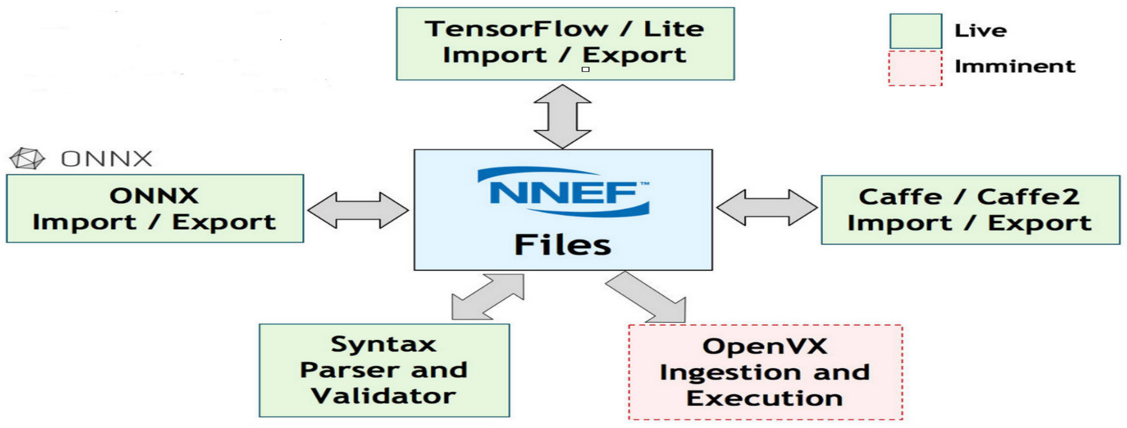

TensorFlow / Lite Import / Export
🟩 Live
🟥 Imminent
ONNX
ONNX Import / Export
NNEF Files
Caffe / Caffe2 Import / Export
Syntax Parser and Validator
OpenVX Ingestion and Execution

Figure 15 Compatibility between ONNX and NNEF [28]

2.5.4 ML interoperability Framework Choice for I-NERGY

For the I-NERGY project, ONNX is a better choice as it is an open-source standard and enjoys a larger community of industrial and academic research and development group. ONNX can also work with BentoML or Kubernetes as a DevOps orchestration tool. Since we have different services and ML development, which involves different partners that might be using different model frameworks for development, ONNX will help in harmonising the ML model to a model format that will be served in the upper layer of the architecture after the Transfer Learning is carried out for every incremental learning.

Figure 16 shows the conversion process from the trained ML model from different framework or toolkit to the generation of the computation graph of the model that can be easily ported to production. The different model framework parser converts the trained ML or DL model to the individual framework computation graph. The Intermediate model generator (IMG) then converts this different computation graph to an intermediate computation graph. The Target module generator then converts the intermediate computation graph into the final target model or the ONNX generic model depending on the final deployment environment for inferencing like Deep Learning Accelerator (DLA) [19].

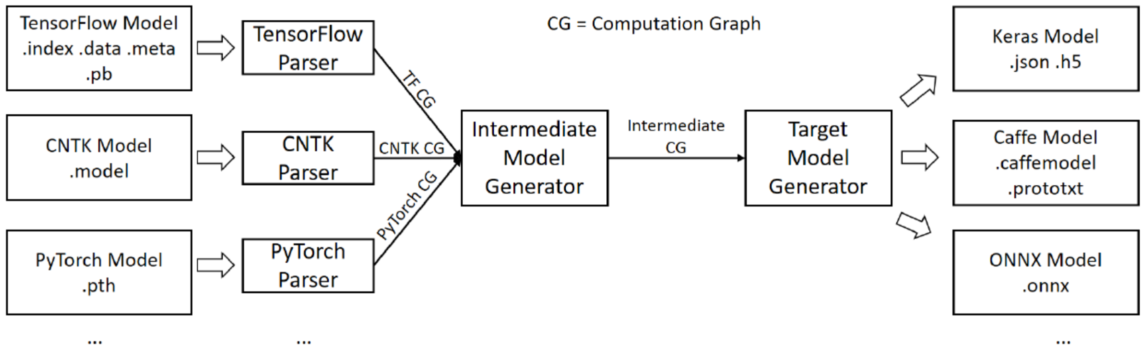


Figure 16 ONNX Conversion process [32]

2.6 Transfer Learning

Machine learning techniques, part of the broader field of artificial intelligence, lie at the intersection of statistics, computer science and neurobiology. Using sophisticated algorithmic approaches, the machine learning technique is able to learn automatically, i.e. approximate, the underlying distribution of a set of observations, i.e. data. When the data is sufficient and of good quality, the model is typically able to generalize well and, subsequently perform well also on other, previously unseen data from a different but related domain. For instance, a machine learning model developed to recognize certain shapes of faults in energy time series, can be tuned to recognize similar shapes of faults in for instance networking time series.

Transfer learning is an important tool in machine learning to solve the basic problem of insufficient training data. It tries to transfer the knowledge from the source domain to the target domain by relaxing the assumption that the training data and the test data must be i.i.d [31]. More formally, given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(\bullet)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$ [32]. A domain is a pair D={$\mathcal{X}$, P(X)} where $\mathcal{X}$ is the feature space and P(X) is the marginal probability distribution. A task is defined as a pair $\mathcal{T}$ ={$\mathcal{Y}$, P(Y|X)} where $\mathcal{Y}$ represents the label space and P(Y|X) represents the objective predictive function. It can be seen from the definition, that to realise transfer learning, either the domain or the task need to be different, i.e. $D_S \neq D_T$, or $T_S \neq T_T$. As summarized in Table 2, for the domain to be different, either the feature space $\mathcal{X}$, or the marginal probability distribution P(X) have to be different while for the task to be different, either the label space $\mathcal{Y}$, or the objective predictive function P(Y|X) need to be different.

Table 2 Summary of transfer learning cases.

$D_S \neq D_T$		$T_S \neq T_T$	
$\mathcal{X}_S \neq \mathcal{X}_T$	$P(X)_S \neq P(X)_T$	$\mathcal{Y}_S \neq \mathcal{Y}_T$	$P(Y\|X)_S \neq P(Y\|X)_T$
the measurement and/or window sizes of the two timeseries are different	the source domain time series are on wireless networks anomalies while the target- domain	source domain has binary target classes, whereas the target domain has 10 classes	source and target timeseries are very unbalanced in terms of the user- defined classes

leading to different feature spaces	documents focus on energy network fault detection	

According to a recent survey [32], they categorize transfer learning depending on the availability of data and/or labels as depicted in [33]. As for general machine learning, also for transfer learning the presence or absence of labelled data is a major factor in distinguishing the type of learning. When no labels for data is available, we are dealing with unsupervised transfer learning. When labelled data is available, then, in the case of transfer learning, the separation comes based on the per-domain availability as can be seen in the decision diagram in Figure 17. When the labelled data is available for the target domain, then we are talking about Inductive Transfer Learning while when data is available only for the source domain, we are talking about transductive transfer learning.

More recently, also a space based categorization has been proposed [33] distinguishing between homogeneous and heterogeneous transfer learning.

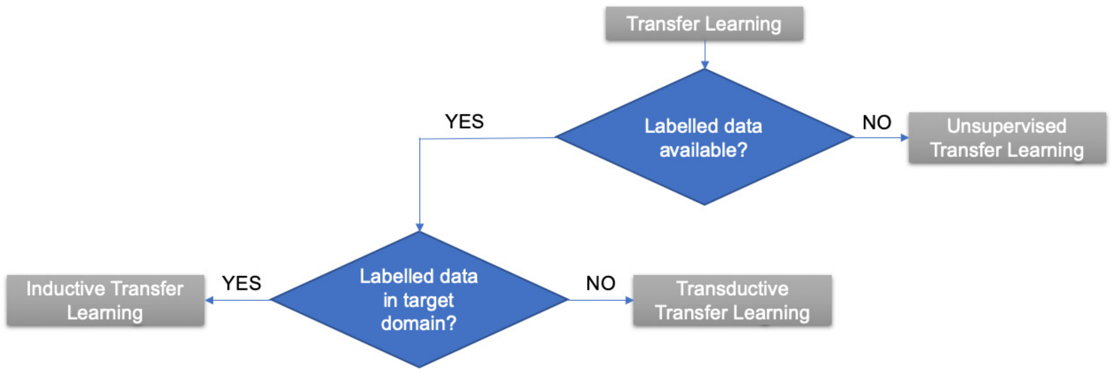

Transfer Learning
Labelled data available?
YES
NO
Unsupervised Transfer Learning
Labelled data in target domain?
YES
NO
Inductive Transfer Learning
Transductive Transfer Learning

Figure 17 Types of transfer learning

2.6.1 Inductive Transfer Learning

In inductive transfer learning, labelled data in the target domain is available and the target task is different from the source task, no matter when the source and target domains are the same or not. When plenty of labelled data is available in the source domain, this type of learning is similar to multi-task learning, however, the inductive transfer learning setting only aims at achieving high performance in the target task by transferring knowledge from the source task while multitask learning tries to learn the target and source task simultaneously. When no labelled data is available in the source domain, the learning is similar to self-taught learning.

In inductive transfer learning setting, a few labelled data in the target domain are required as training data to induce the target predictive function. In inductive transfer learning, four different types of knowledge can be transferred from the source to the target domain: knowledge about the instances, knowledge about the feature representation, knowledge about parameters or relational knowledge.

2.6.2 Transductive Transfer Learning

In transductive transfer learning, no labelled data in the target domain is available, the source and target tasks are the same, while the source and target domains are different. According to different situations between the source and target domains, transductive transfer learning can have two flavours. The first, when the feature spaces between the source and target domains are different while in the second, the feature spaces are the same, but the marginal probabilities between the two domains are different.

In transductive transfer learning, two different types of knowledge can be transferred from the source to the target domain: knowledge about the instances and knowledge about the feature representation.

2.6.3 Unsupervised Transfer Learning

In unsupervised transfer learning, no labelled data is available, and the target task is different from but related to the source task. However, the unsupervised transfer learning focuses on solving unsupervised learning tasks in the target domain, such as clustering, dimensionality reduction, and density estimation.

In unsupervised transfer learning knowledge about the feature representation can be transferred across the domains.

2.6.4 Suitability of Transfer Learning

While the concept and definition of transfer learning are generic, in practice transfer learning is frequently used when 1) the available training data for the target domain is insufficient and 2) when training the model from scratch each time is unfeasible or too expensive. In this project, starting from the use cases needing machine learning techniques, we will assess the availability and feasibility of data. Depending on the requirement and type of data, we will employ the most suitable transfer learning technique from the three listed in this section of the report.

As DL methods are known to significantly outperform other machine learning methods in several cases, they will be considered for the use cases of this project. At the same time, DL methods are data hungry and expensive to train, therefore ideal candidates for leveraging the benefits of transfer learning. For instance, leveraging the breakthroughs in machine vision, recent attempts to transform time series to images for shape or motif recognition or prediction exist [34],[35]. As an example, consider a non-intrusive load monitoring dataset where several devices need to be automatically detected. The time-series corresponding to these devices can be transformed into images using several techniques (Recurrence plots, Grammian angular fields, etc) as exemplified in Figure 18. A promising approach to develop a general and well-performing classifier for this task is to use existing machine vision models, such as AlexNet or ResNet and transfer them to our domain – this performs inductive transfer learning with knowledge about the parameters of the model.

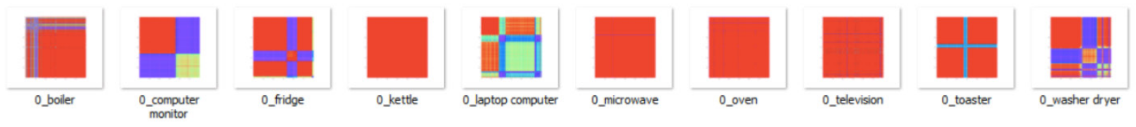


Figure 18 Example time series to image transformation of home appliance time series from a NILM dataset.

2.7 Model Evaluation and Serving Techniques

Model evaluation is a key part of the model development process, it helps to determine how good and accurate the model is with future data. Therefore, the whole dataset is usually split into different sets which are used in different parts of the whole process. This is important since the developers do not want to evaluate the model on the data that it learned from because it would most likely predict them correctly and cause overfitting. Methods are divided into two categories: hold-out and cross-validation.

The hold-out method simply focuses on splitting the data into the *train* and *test* set. Its purpose is to test the model on unseen data, providing an unbiased estimate of learning performance. A common split for this method is using 80% of the data for learning and 20% for testing, but it can also have a validation set that is used for fine-tuning the model's parameters in the training phase. This method is flexible, fast, and simple, but there is a risk where there are differences between the test set and the training set, resulting in the wrong evaluation.

In terms of cross-validation, the most common technique is *k-fold cross-validation*. Here the data is split into *k* subsets of equal size. The model is built *k* times where each time one of the *k* subsets is used for testing and other *k-1* subsets are used for training. The error estimation and accuracy are averaged over all *k* trials which solves the problem of the hold-out approach explained above. Figure 19 shows the difference between hold-out and K-fold cross-validation methods.

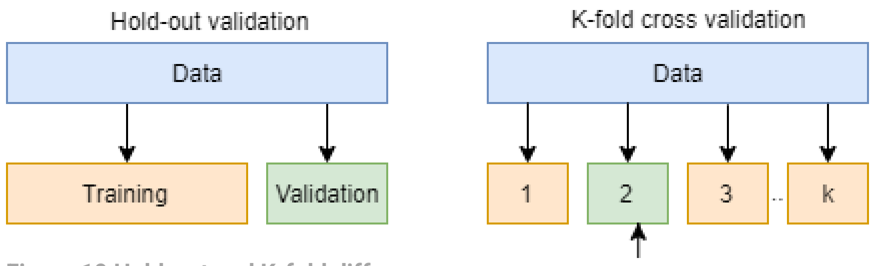


Figure 19 Hold-out and K-fold difference

Regarding the model evaluation metrics, problems can be divided into regression problems and classification problems. Fundamentally, classification is about predicting a label and regression is about predicting a quantity. That being said, there are many different techniques for both classification and regression models. Most used techniques for classification models are *Confusion Matrix, Gain and Lift Charts, K-S charts, and ROC charts*. The most used for regression ones are *Root Mean Squared Root (RMSE), Relative Squared Root (RSE), Mean Absolute Error (MAE), Relative Absolute Error (RAE), Coefficient of Determination (R^2).*

Model-evaluation – Classification

A confusion matrix is a table that describes the performance of a classification model on a test data with known true values. An example can be seen on Table 3. From this matrix we can see that there are 2 possible classes where some statement can be true or false. These are the basic terms, which are whole numbers:

- true positives (TPs) – cases where model predicted that the statement is true, and it is actually true
- true negatives (TNs) – cases where model predicted that the statement is false, and it is actually false
- false positives (FPs) – cases where model predicted that the statement is true, but it is actually false
- false negatives (FNs) – cases where model predicted that the statement is false, but it is actually true

A number of different rates can be computed from the confusion matrix, some of them are:

- Accuracy: (TPs + TNs) / n
- Error rate: (FPs + FNs) / n
- Sensitivity: TPs / (FNs + TPs)
- Precision: TPs / (FPs + TPs)

Table 3 Binary classifier confusion matrix example

N	Actually: TRUE	Actually: FALSE
Predicted: TRUE	True Positives (TPs)	False Positives (FPs)
Predicted: FALSE	False Negatives (FNs)	True Negatives (TNs)

In order to continue with upcoming parts of the document, a new concept will be introduced, MLOps, which is the approach that is intended to be adopted in the I-NERGY project. Similarly to DevOps, MLOps is used to merge the two processes of ML system development (Dev) and ML system deployment (Ops) which will be the goal of the Serving Framework. The whole process and the automated pipeline are explained in the next section (2.8) as shown in Figure 21.

As explained later, built models are automatically tested, evaluated and put in production, which are then monitored for further upgrading or removal. These are the key features that the model serving framework has to offer. A framework that allows the DevOps team to work side-by-side with data scientists and provides high-level APIs for them to create prediction services, DevOps infrastructure needs, and performance optimizations in the process. Some of the interactions and features of the framework can be seen in Figure 20.

To fit in with MLOps standards, the model serving framework has to offer a number of different features for both production, deployment and development. Production features would include support for multiple ML frameworks such as TensorFlow, Keras, PyTorch, Scikit, H20, etc. Considering constant model creation and recreation it would have to serve multiple models on multiple endpoints, automatically generate REST APIs, feedbacks, and health checks. Those models would usually be on containerized model servers, so integration with Azure, Kubernetes, Docker, or others is crucial.

In terms of model serving and deployment workflow for teams it would have to offer a central repository for all services via Web UI and API, utilities that simplify CI/CD pipelines, integration with training and experimentation management products, deployment to cloud platforms like AWS, launching offline batch inference jobs and others.

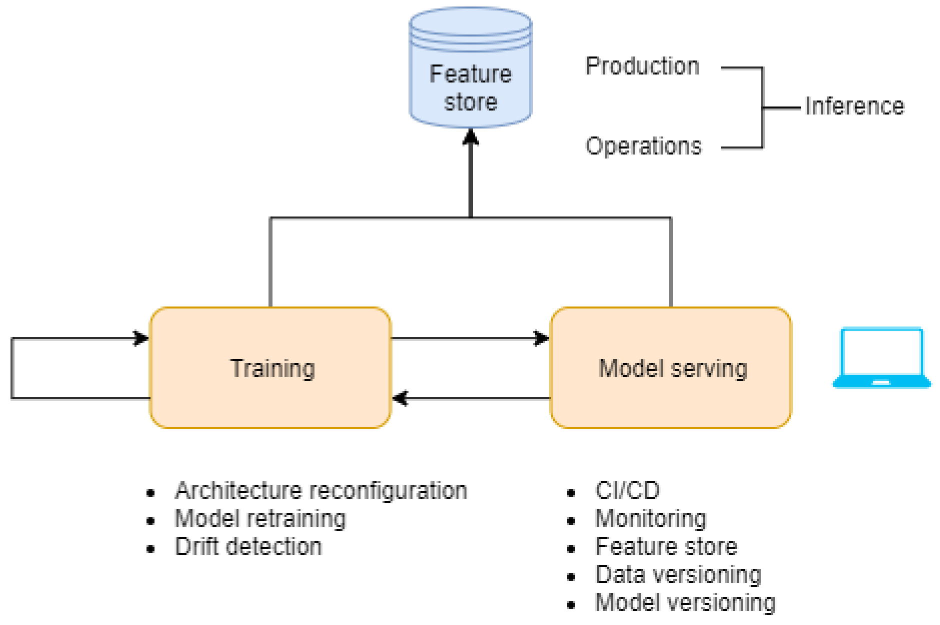


Figure 20 ML model serving

2.8 Model Updating

In MLOps terms, Figure 2.1 discussed in Section 2 represents the manual process that lacks many important characteristics such as CI, CD, performance monitoring, release iterations, etc. In terms of solving a lot of those problems and having model updating as the primary focus, ML pipeline will

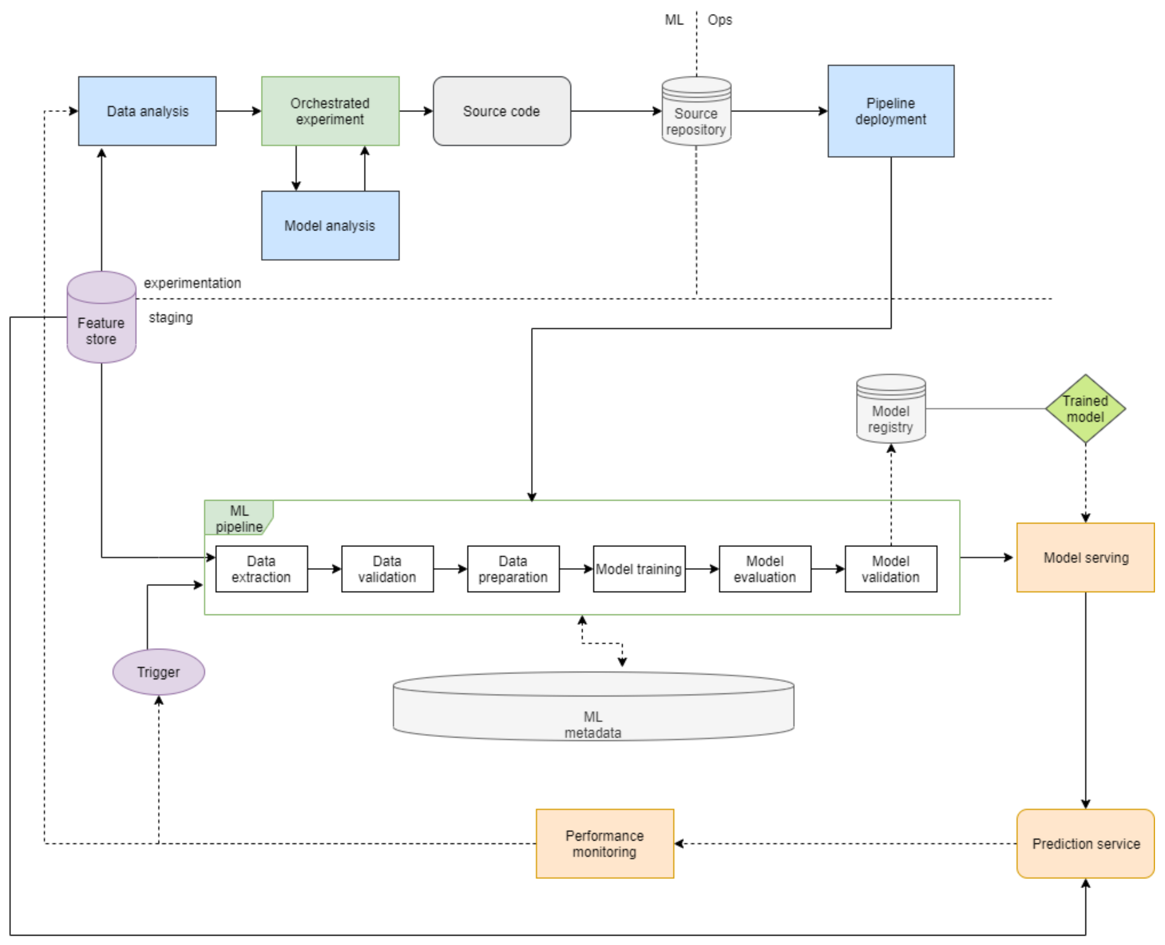


Figure 21 ML pipeline for CT

be used. The ML pipeline consists of data and model validation, pipeline functionalities, metadata management, and feature stores which will be explained later on.

Figure 21 represents the whole pipeline for continuous training (CT). It can be viewed through two big parts: the staging (production) part and the experimentation (testing and development) part. The key part of staging is the pipeline for data and model operations. Data is first being extracted, validated, and prepared for the model, which is then trained, evaluated, validated, and put in production. All the ML models in production are being monitored and tracked, any loss in their performance results in model retraining and replacement. Both experimentation and staging parts use the same feature store, which is a centralized place for data and features. This ensures that data scientists have the same data available to them as the ML pipeline, which they can use for experimentation, analysis and further improvements of the pipeline and the system itself.

Using the ML pipeline reduces the risk of human error, helps data scientists focus on more important tasks and makes ML more accessible to everyone involved. It also speeds up the whole process of testing, training and relearning ML models that are then put in production, which greatly increases the quality of the system.

Some of the functionalities that the ML pipeline offers are data and model evaluation notifications. When bad model performance or data anomalies are detected by the developers, model retraining, and replacement are performed.

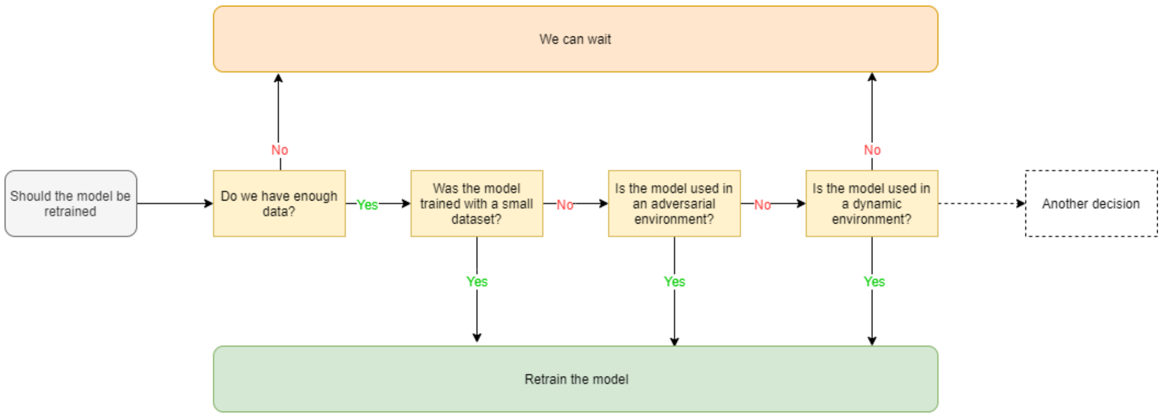

We can wait
No
No
Should the model be retrained
Do we have enough data?
Yes
Was the model trained with a small dataset?
No
Is the model used in an adversarial environment?
No
Is the model used in a dynamic environment?
Another decision
Yes
Yes
Yes
Retrain the model

Figure 22 simple decision tree based on model and data

In order to execute the pipeline, both data and models need to be validated. Data validation covers two different validation aspects of data, schema and value skews. Schema skews are input data anomalies, where the expected schemas for models or data processing are not matching the input ones. Value skews are statistical anomalies in data which means that retraining of the models needs to be triggered.

Regarding model validation, its new performance needs to be consistent on different segments of the data, better and more accurate compared to the previous one and also compatible and consistent with the prediction service API.

In order to track, understand and upgrade the system in a more efficient and fast way, each time the pipeline is executed the ML metadata will be stored. Some of the metadata properties that will be recorded are: parameter arguments passed to the pipeline, time and date when the pipeline was executed and how much time it needed to complete each step, component versions, rollbacks of previously trained models for metrics or if something goes wrong, all the model evaluation metrics produced in training and testing phases.

One of the most critical components for any ML process is feature engineering or creating features which takes a huge amount of work. Feature stores represent a centralized place used for sharing all the available features, they fit between the data and the ML models. Data scientists can reuse these features instead of creating them again and again, saving a lot of time and effort.

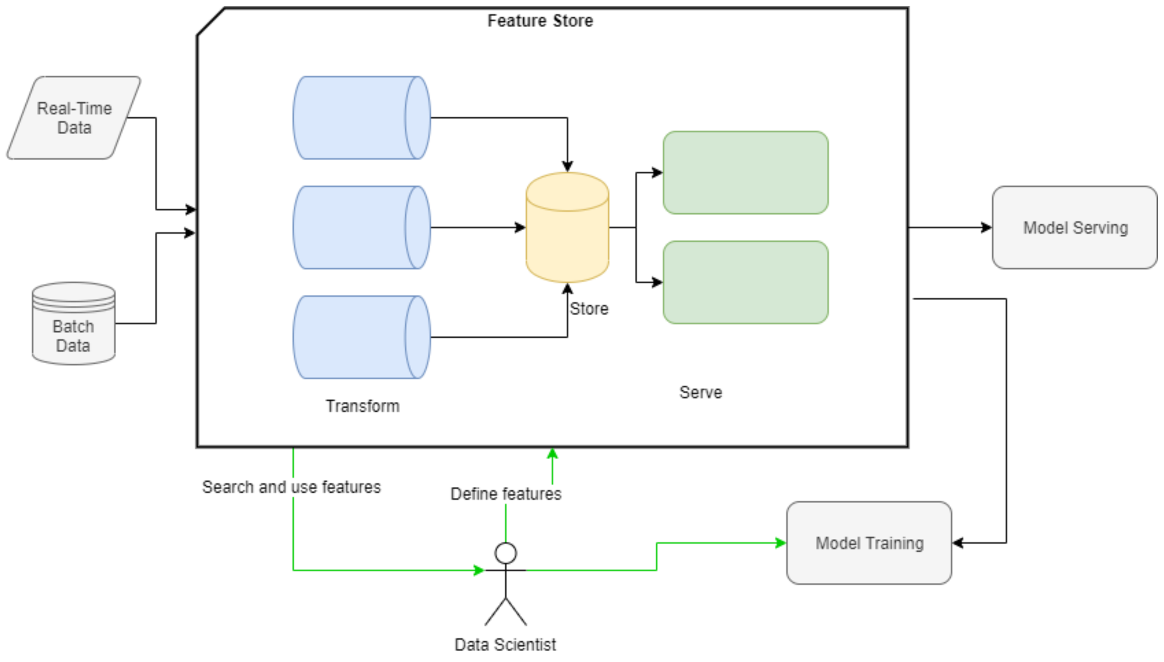


Figure 23 Feature store schema

Figure 23 represents a use case of a feature store. Raw data is being transformed and stored in a feature store, which is then ready to be served. It can be served to the pipeline in the production section, that uses it to create and serve models, or it can be served to a data scientist who can use the features to test new models, refactor or update the old features or define new ones.

2.9 Access Control Techniques for Data and Trained Models

Security and access control strategies and techniques for data and access control in ML/AI enabled system is becoming of critical importance now as such systems are coming to market supporting the operation of businesses and organizations of various sizes. To design suitable technique and put in place adequate policies and practices, the ML/AI lifecycle has to be considered and the points of vulnerability understood. In [36], the authors identify the workflow of AI systems at a) *training* and b) *operation* phases for both data and models, as well as at the c) *inference* phase. Starting from [36], [37], we identify the following main challenges to prevent attackers/malicious entities from compromising ML/AI-based systems:

i. **Corrupting the training data**: Off-line ML models are trained periodically and then the new models are deployed to production. A malicious entity could attempt to corrupt the training data in a way that would result in a model that would behave according to their malicious intentions. In other words, a small change implied by a malicious entity can affect the results in the favour of that entity.

   When the model is trained periodically on static data (i.e. offline model), the security system should ensure that 1) malicious entities cannot get access to the data repository and that 2) all the data available in the repository is legit and unaltered. When the model is trained continuously during operation (i.e. online learning), a mechanism for ensuring that the stream of data is legit and unchanged by possible injections. *According to* [36]*, this challenge is related to the Data Breach, Bias in Data and Data Poisoning attack types for which there are standards under development that attempt to address it: ISO/IEC CD 20547-4, ISO/IEC NP TR 24027 and ISO/IEC PD TR 24028*.

ii. **Corrupting the operation data**: During the operation of both on-line and off-line trained models, the incoming data points have to be reliable and uncorrupted. *According to* [36]*, this challenge is related to the Data Breach and Evasion attack types for which there are standards under development that attempt to address it: ISO/IEC CD 20547-4 and ISO/IEC PD TR 24028*.

iii. **Swapping the models**: For both on-line and off-line ML systems the models are available at the place of operation, i.e. a VM in the cloud, edge or on the IoT/embedded device. Additionally, periodic back-ups of the models may be available in a store or repository as well for redundancy or accountability when something goes wrong. A malicious entity should be prevented from accessing and potentially swapping both production and back-ups to avoid the replacement of a legit model with a malicious one.

iv. **Stealing the models**: In some cases, the malicious entity may not have any incentive to swap a legit model for their own (i.e. write access to the system), they may only wish to copy the model (i.e. read-only access) as per se that would bring them an advantage and harm the legit owner [38]. *According to* [36]*, this challenge is related to Model Extraction attack type for which there are standards under development that attempt to address it: ISO/IEC PD TR 24028*.

v. **Stealing the outputs of the models**: Rather than attempting to steal the model itself, a malicious entity may attempt to steal information/knowledge from the model that materializes as output from the inference.

To solve the above challenges established security practices such as pertaining to physical, software and hardware need to be employed as well as new AI specific ones [38] that open the need for new practices, regulation and standards with respect to. Ironically, AI is often also part of the solution for securing AI (i.e. AI for intrusion detection, anomaly detection, etc.) as recently highlighted [38], [39] . At the same time, while AI can be part of the solution, it can also be part of the attack when it is weaponized [40].

2.9.1 Techniques for mitigating security and access to data and trained models

While AI security is a fast developing area, according to [38], there are already techniques for securing systems against the various attacks identified in [36] and discussed in this section. Table 4 shows a mapping of the identified security challenges to the phases of de ML DevOps pipeline, attack type and possible mitigation technique.

Table 4 Mapping of identified challenges to the ML DevOps phase, possible attack type and mitigation technique.

Challenges	Phase	Attack type	Mitigation technique

Corrupting the training data	*Training (offline)* *Operation (online)*	*Data Breach, Bias in Data and Data Poisoning*	*Adversarial Samples, Data Filtering, Regression Analysis, Differential Privacy, Adversarial Detection, Input Reconstruction,*
Corrupting the operation data	*Operation (offline/online)*	*Data Breach and Evasion*	*Adversarial Detection, Input Reconstruction, Input Preprocessing*
Swapping the models	*Backup or operation*		*Model Verification, Ensemble Analysis, Model Pruning, Model Watermarking, Model Explainability*
Stealing the models	*Backup or operation*	*Model Extraction*	
Stealing the outputs of the models	*Operation*		

Adversarial Samples refer to perturbing the input features to cause a machine learning model to make a false prediction. As a mitigation technique, it can be used during the training phase to make the model robust to such attacks.

Data Filtering is a technique that can identify and filter out data that may be corrupted by a malicious entity.

Regression Analysis is a technique based on statistical methods that detects noise and abnormal values in data sets. This technique checks for abnormal values, or the distribution characteristics of data and eliminates suspicious ones.

Differential Privacy is a technique that adds noise to data or models so that in the case of stolen data or model they are not compromised.

Input Reconstruction is a technique that changes the input samples in the model inference stage to defend against evasion attacks. The changed input does not affect the normal classification function of models.

Adversarial Detection is a technique that identifies adversarial examples by adding an external detection model or a detection component of the original model in the model inference stage. Before an input sample arrives at the original model, the detection model determines whether the sample is adversarial.

Model Verification is similar to software verification analysis technologies. The technique uses a solver to verify the behaviour of a model. For example, a solver can verify that no adversarial examples exist within a specific perturbation range.

Ensemble Analysis is a technique to use multiple models, i.e. ensembles, to perform inference. Should a minority of the models in the ensemble be compromised, the uncompromised ones will likely compensate and keep the system functioning as intended.

Model Pruning is a technique that is used to prevent models from overfitting the training dataset. This technique is also used to prevent models to include some “hidden” behaviour induced by small portions of compromised training data.

Model Explainability is a technique that enables understanding and explaining the inference of a model. This can be used for developing fair AI systems, but also to explain behaviours that are suspicious and may be caused by a security breach.

Model Watermarking is a technique that includes a unique watermark in the model that can be used to recognize its authenticity and integrity.

The most suitable MLOps framework enabling the development and deployment of ML models should include as many of these security mitigation techniques as possible.

3 Components Overview and Methodology

Herein, we present a general overview of each component as well as the proposed approach to realise them within the context of I-NERGY.

3.1 Incremental, Distributed and Self-learning Model

The purpose or goal of the incremental, distributed and self-learning model definition and training component is to exploit, assess and implement different ML/DL algorithms and optimisation techniques (including leveraging the AI4EU package tools for ML, etc.) to enable efficient training of complex models and handling of computationally intensive problems arising in various use cases within the I-NERGY project. This will involve exploration of various procedures and measures for assessing the performance of the available alternatives based on the use cases examined and the algorithms being tested. In this regard, this component will examine a variety of training and evaluation procedures, as well as advanced criteria for selecting the most appropriate model for each use case. Validation procedures such as simple holdout tests, cross-validation and random sampling will be considered when defining and training ML/DL for I-NERGY tasks, while classification accuracy, logarithmic loss, confusion matrix, F1 score and mean absolute/squared error etc. are some performance measures that will accompany these procedures. It should also be noted that the type of the problem being solved (supervised or unsupervised learning / classification, regression or clustering), the size of the sample data and the objectives of the algorithm (accuracy vs. efficiency) will also be considered when performing an incremental analysis and determining the most appropriate model for a given use case. Further, different hyper parameters will be examined for each model of interest and the most effective ones will be determined for each use case task to maximize their potential and ensure that they are properly optimised for each training dataset. This is particularly important for effective handling of big data and deep learning applications within the I-NERGY project, where the amount of data that need to be processed and the number of parameters that have to be estimated are immense.

Table 5 presents key models to be developed as well as the corresponding related tasks within which the models will be developed. We arrived at these by analysing the pilot data in D1.2 and pilot service descriptions in D2.1. This is particularly important as different tasks within WP3 and WP4 are involved in model development. Thus, this will help to avoid duplication of efforts in model development. It was agreed during project meeting/discussions that T3.3 and T3.5 will collaboratively work on models that are not going to be developed within WP4. The models enumerated in the table below to be developed by T3.3 and T3.5 are not exhaustive as partners involved in the two tasks may be required to assist some service developers in developing some models.

Table 5 Key ML models to be developed

ML Model	Related Task

Predictive Maintenance Model for Circuit Breakers	T3.3&T3.5
Predictive model for asset failure (transformers, lines ...)	T3.3&T3.5
Predictive model for RUL of equipment to determine when to replace it at a minimal cost	T3.3&T3.5
Predictive Maintenance Model for PV Plants	T3.3&T3.5
Activity recognition and forecasting model	T3.4
Anomalous behaviour detection model	T3.4
ML model to cluster residential electrical loads (based on total electrical load profile)	T4.2
ML model to cluster residential thermal loads (based on total thermal load profile)	T4.2
ML model to cluster energy consumers into different behavioural clusters based on their flexible load patterns	T4.2
Generative model(s) to produce synthetic traces that can characterise real-world behaviour for EV charging sessions (relates to electrical DT)	T4.2
Model for determining the behaviour in relation to heat re-use of different assets such as heat pumps, spaces, etc. (relates to thermal DT)	T4.2
Aggregated models incorporating both electrical and thermal loads	T4.2
Solar radiation forecasting model	T4.2
Energy flexibility forecasting model	T4.4
Energy load/consumption/demand forecasting model	T4.4

3.1.1 Requirements

Several requirements need to be met in order to achieve the goals of this component. First, the processed dataset by the data services layer needs to be made available for model training. The nature of the data needs to be considered to develop a suitable model (classical ML model, DL model). Where the data meets the requirements for incremental learning as discussed in Section 2.2, a suitable incremental learning technique will be employed. To ensure effective model training using huge datasets from the pilots, it is necessary to adopt distributed deep learning methods when training the models. Both data and model parallelism, i.e., partitioning data and machine learning models to optimise and distribute the workload to multiple computational workers/nodes will be considered. To ensure high quality of services and automation, self- learning techniques will be explored for some I-NERGY solutions where appropriate. Because several partners are involved in the I-NERGY project, it is necessary to ensure flexibility in the choice of ML/DL frameworks to be used. To be specific, different partners may have preference to use one framework or the other, for instance; I-NERGY partners may prefer to use ML/DL frameworks such a TensorFlow, PyTorch, Caffe, Scikit-Learn, Torch, Theano etc. This calls for the need to use a suitable interoperable model format that can enable this flexibility when saving the trained models.

3.1.2 Proposed Approach, Tools and Justifications

Based on the requirements enumerated in subsection 3.1.1 above, several approaches and tools were considered. For instance, regarding model development, we are currently using TensorFlow and PyTorch, which are two of the most popular libraries/frameworks for machine learning. TensorFlow is an end-to-end open-source machine learning framework. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that makes it possible for researchers to push the state-of-the-art in ML and developers easily build and deploy ML powered applications [41]. Like TensorFlow, PyTorch [24] is an end-to-end open source machine learning framework which enables fast, flexible experimentation and efficient production through a user-friendly front-end, distributed training, and ecosystem of tools and libraries. One special thing about PyTorch is that it supports dynamic computational graphs, which means that the behaviour of neural networks defined in PyTorch can be changed programmatically at runtime, thus facilitating more efficient model optimization. This is a major advantage of PyTorch over other machine learning frameworks that treat neural networks as static objects. Thus, our plan is to make the best use of the strengths of these frameworks.

To achieve the requirement of distributed model training, we explored various distributed deep learning training frameworks such as distributed TensorFlow which requires the use of parameter server, Baidu's AllReduce [42], and Horovod [43]. After careful evaluation of the pros and cons of each distributed training solution as reported in section 2.4, we decided to opt for Horovod because it is more scalable and efficient than the other solutions. Horovod makes it easy to take a single-GPU training script and successfully scale it to train across several GPUs in parallel. It can be used with TensorFlow, Keras, PyTorch, Apache MXNet etc.

Regarding model interoperability, we leverage on ONNX. Refer to Section 2.5 for a more detailed discussion on ONNX and the reason why it is our preferred choice to achieve model inter-operability.

Figure 24 shows our proposed model development workflow that meets the requirements presented in the preceding section.

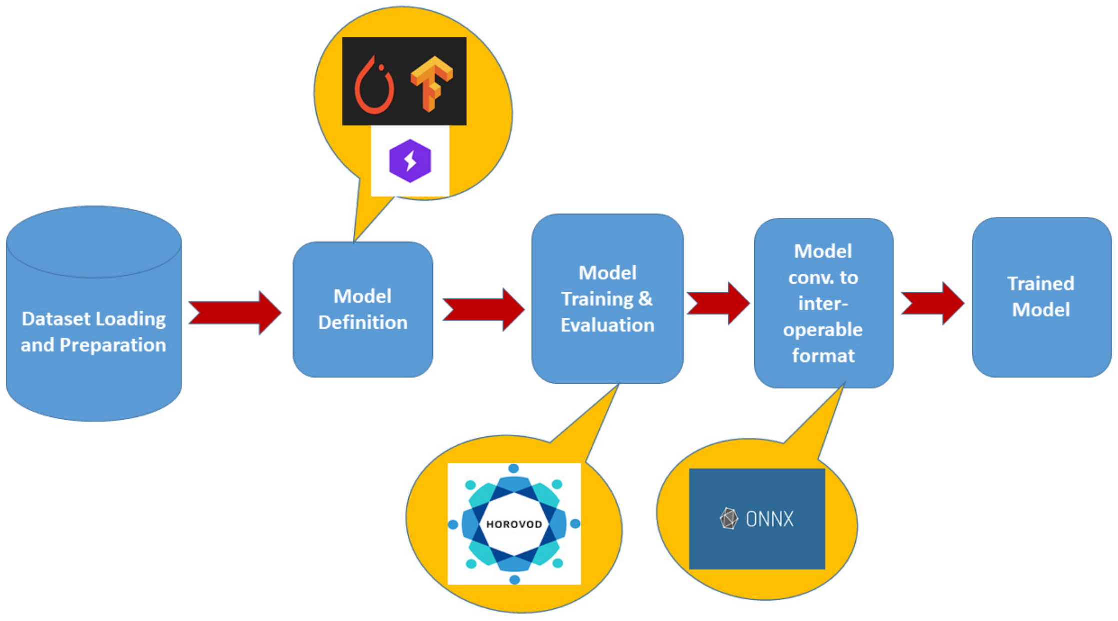

Dataset Loading and Preparation
Model Definition
Model Training & Evaluation
Model conv. to inter-operable format
Trained Model
HOROVOD
ONNX

Figure 24 Proposed Model Development Workflow

3.1.3 Distributed training via Horovod with Pytorch

Distributed training in Pytorch using Horovod by executing the following changes to the Pytorch training code:

- Run hvd.init().

- Pin each GPU to a single process.

  With a typical setup of one GPU per process, set this to *local rank*. The first process on the server will be allocated as first GPU, the second process will be allocated the second GPU, etc.

  ```
     if torch.cuda.is_available():
  torch.cuda.set_device(hvd.local_rank())
  ```

- Scale the learning rate by the number of workers.

  Effective batch size in synchronous distributed training is scaled by the number of workers. An increase in learning rate compensates for the increased batch size.

- Wrap the optimizer in hvd.DistributedOptimizer.

  The distributed optimizer delegates gradient computation to the original optimizer, averages gradients using *allreduce* or *allgather*, and then applies those averaged gradients.

- Broadcast the initial variable states from rank 0 to all other processes:

  hvd.broadcast_parameters(model.state_dict(), root_rank=0)

  hvd.broadcast_optimizer_state(optimizer, root_rank=0)

  This helps to ensure consistent initialization of all workers when training is started with random weights or restored from a checkpoint.

- Modify the code to save checkpoints only on worker 0 to prevent other workers from corrupting them.

This can be achieved by guarding model checkpointing code with hvd.rank() != 0.

An example implementation of this distributed training approach is shown in Table 6.

Table 6 Distributed Model Training via Horovod in Pytorch Example

```
import torch

import horovod.torch as hvd



# Initialize Horovod

hvd.init()



# Pin GPU to be used to process local rank (one GPU per process)

torch.cuda.set_device(hvd.local_rank())



# Load dataset...

train_dataset = ....



# Partition dataset among workers using DistributedSampler

train_sampler = torch.utils.data.distributed.DistributedSampler(

    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
```

```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=..., sampler=train_sampler)


# Model definition ....

model = ....

model.cuda()


optimizer = optim.SGD(model.parameters())


# Add Horovod Distributed Optimizer

optimizer = hvd.DistributedOptimizer(optimizer, named_parameters=model.named_parameters())


# Broadcast parameters from rank 0 to all other processes.

hvd.broadcast_parameters(model.state_dict(), root_rank=0)


for epoch in range(num_epochs):

  for batch_idx, (data, target) in enumerate(train_loader):

      optimizer.zero_grad()

      output = model(data)

      loss = F.nll_loss(output, target)

      loss.backward()

      optimizer.step()

      if batch_idx % args.log_interval == 0:

          print('Train Epoch: {} [{}/{}]\tLoss: {}'.format(

              epoch, batch_idx * len(data), len(train_sampler), loss.item()))
```

3.1.4 Example Implementation Using C-MAPSS Dataset for RUL Estimation

The Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) [44] dataset was developed by the National Aeronautics and Space Administration (NASA), and it includes four sub-datasets, that is, FD1, FD2, FD3, and FD4. In recent years, it has been used as the benchmarking prognostic models. Datasets include simulations of multiple turbofan engines over time, each row contains the following information:

1. Engine unit number
2. Time, in cycles
3. Three operational settings
4. Sensor readings

Each sub-dataset contains multivariate temporal data coming from 21 sensors.

We experimented with this dataset as the pilot data are not yet ready for model training. Table xxxx shows the results obtained by using various classical and deep learning based methods. Herein, we used variational autoencoder (VAE) for dimensionality reduction. VAE has an architecture of encoders and decoders. To be specific, the encoder is applied to extract features from the raw data; on that basis, the decoder uses the extracted features to reconstruct the raw data as close as possible. VAE's encoder is fed with the input, and its output has two vectors, that is, the mean of μ and variance of Σ. After training, the encoder part of the VAE was used to encode the test data samples to generate latent variables that are fed directly to the long short term memory (LSTM) and gated recurrent unit (GRU) models that forms the reliability models for remaining useful life (RUL). The results obtained by using various ML/DL models are shown in Table 7.

Table 7 RUL estimation using various classical ML and DL models on C-MAPSS dataset

Sr. No	Implemented Model	RMSE
1.	Modified Linear Regression	31,95
2.	Support Vector Regression	29,57
3.	Support Vector Regression with Feature Engineering	20,58
4.	Time Series Analysis	20,85
5.	Random Forest Regressor	20,31
6.	VAE + GRU	14,21
7.	VAE + LSTM	8,95

3.2 Cross-stakeholder Transfer Learning

The I-NERGY project proposed eight pilots where various stakeholders identified goals to be realised based on the data available within the pilot as described in D1.2. Starting from that information, we analyse the goals and data for each pilot and identify the type of data, the type of ML problem and the corresponding evaluation metrics that are needed to fulfil the goal and realise the desired services as summarised in Table 4. It can be seen from the table that, based on the details provided in D1.2:

A) all pilots produce time series data,
B) 3 pilots require predicting future values of the time series that is a regression problem in ML,
C) One pilot requires predicting maintenance needs which is typically a classification problem where discrete states in the future are predicted such as binary (i.e. maintenance/no maintenance) or multiclass (i.e. names for various types of maintenance).
D) Three pilots require both regression and classification
E) For one pilot, the ML problem is yet to be clearly defined.

The last column of the table identifies the evaluation metrics to be used to evaluate the quality of the model. The most frequently used evaluation metrics for regression problems are MSE and RMSE while the most frequently used ones for classification problems are precision, recall and F1 score. Accuracy is also often used for classification, however it can be a biased measure for unbalanced datasets.

Table 8 Pilots and goals mapping to ML domain

Pilot name	**Goal**	**Data source**	**Data type**	**ML problem type**	**Evaluation metrics**
Pilot 1	Predict grid consumption in short time intervals	Meters (TSO/DSO, DSO)	Time series	Regression	MSE, RMSE,
Pilot 2	Improve the operation of the facilities	Smart meters	Time series	-	-
Pilot 3	Simulate the behaviour of the power grid in real time, several uses cases including predictive maintenance	150 smart meters, 2 PMUs, 2 secondary electrical substations	Time series	Regression, Classification	MSE, RMSE, Accuracy, Precision, Recall, F1

Pilot 4	Predictive maintenance	Scada systems and meters	Time series	Classification	Accuracy, Precision, Recall, F1
Pilot 5	Monitoring of data related to consumption of EV charging and charging transactions	EV charging stations	Time series	Regression, Classification	MSE, RMSE, Accuracy, Precision, Recall, F1
Pilot 6	Estimate opportunities for P2P trading	Smart meters, PV in homes and public buildings	Time series	Regression	MSE, RMSE
Pilot 7	Predict grid consumption/production in short time intervals and enforce P2P energy trading	smart meters, scada systems	Time series	Regression	MSE, RMSE
Pilot 8	AI solution for energy efficiency investments de-risking	Public datasets, smart metering systems in selected buildings	Time series	Regression, Classification	MSE, RMSE, Accuracy, Precision, Recall, F1
Pilot 9	Not yet identified	---------------	-----------	---------------	-------------

By analysing the goal of the pilots in the first column of Table 4, it can be noticed that Pilots 6 and 7 focus on predictions for P2P trading. For Pilot 6, prediction of both consumption and PV generation can be done while for Pilot 7, consumption can be predicted based on smart meter data. Additional data from SCADA may augment the smart meter one or provide grounds for forecasting other aspects, including consumption, depending on the availability in the case of Pilot 7. As discussed in Section 2.6, transfer learning is suitable when not sufficient training data is available to develop a model that performs well enough. For instance, if it turns out that Pilot 7 would have insufficient data to train a good model to predict consumption, a model could be trained on the relevant data from Pilot 6 and then transferred to the data for Pilot 7, thus satisfying also the use cases of the Pilot 7 stakeholders. Furthermore, if there will still be room for improving the performance of the model, existing publicly available datasets[i] that are similar could be added to improve the performance and realise a suitable transfer. Similar reasoning can be applied for the predictive maintenance use cases from Pilots 3 and 4, while most of the other pilots seem to have needs that currently could rely on transferring learned models from external publicly available datasets.

3.2.1 Requirements

The requirements for realising the components can be summarised as follows. Each stakeholder should upload the datasets and the corresponding metadata describing the contents of the dataset to a common data store structured according to the pilots or goals as per Table 4. Based on the data and the existing ML pipelines, various models will be developed to reach the goals of Pilots and use cases. The models will be evaluated using the suitable metrics as identified in Table 4. Next, the suitability of improving the models using transfer learning will be assessed and the exact learning strategy will be selected as discussed in Section Section 2.6. Based on the goals and data summarised in Table 4, we anticipate the need for inductive or transductive transfer learning.

3.2.2 Proposed Approach, Tools and Justification

The realisation of the transfer learning component as part of the ML pipeline depicted in Figure 25 will be based on existing deep learning libraries namely Tensorflow, Keras or Pytorch. Keras is easier to use for non-experts, Pytorch is more suitable for mathematician and data science experts while Tensorflow requires good software engineering and machine learning skills. TensorFlow is more flexible and in many cases superior in sophisticated production systems. As all these libraries are changing fast and new functionality is added on a per monthly basis, we will use Keras and switch to Pytorch if needed.

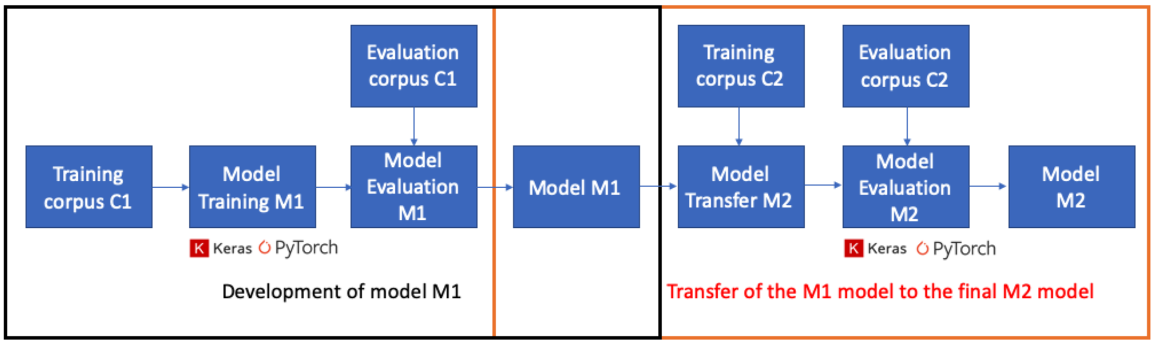


Figure 25 Realization of the transfer learning.

An existing model M1 will be taken or trained on a dataset represented by corpus C1. Once the model is developed, it will be transferred using the dataset from corpus C2 to the realise the final model M2 to be used in production. An example model is provided in Table 9.

Table 9 Example model development with Keras

```
"""
Import the necessary libraries for end-to-end model development.
"""
import h5py
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import matplotlib.pyplot as plt

import itertools

import pyts

from pyts.approximation import PiecewiseAggregateApproximation

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras import optimizers

from tensorflow.keras.layers import Convolution2D,Conv2D, Dense,Dropout,␣

,→Flatten, Activation, MaxPooling2D, Input, Conv1D, GlobalAveragePooling1D,␣

,→TimeDistributed, GRU, LSTM

print(tf.__version__)

  print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

........

"""

Model definition.

"""

input_a = Input(shape = X_train.shape[1: ])

x = TimeDistributed(Convolution2D(16, (7, 7), activation='relu', padding='same'))(input_a)

x = TimeDistributed(Convolution2D(16, (7, 7), activation='relu', padding='same'))(x)

x = TimeDistributed(MaxPooling2D(pool_size=(2, 2), padding='same'))(x)

..........

x = Dropout(0.10)(x)

out = TimeDistributed(Convolution2D(16, (3, 3), activation='relu', padding='same'))(x)

out = TimeDistributed(Convolution2D(16, (3, 3), activation='relu', padding='same'))(out)

out = TimeDistributed(Flatten())(out)

out = LSTM(16, return_sequences=False, unroll=False, dropout=0.2)(out)

prediction = Dense(num_classes, activation='softmax')(out)

model = keras.models.Model(inputs=input_a, outputs=prediction)

adam = optimizers.Adam(lr = lr)

model.compile(loss = 'categorical_crossentropy', optimizer = adam, metrics= ['accuracy'])
```

3.3 Model Evaluation and Serving Framework

In the context of the I-NERGY project, a number of Machine Learning models will be developed to meet the needs expressed by the pilots during the preliminary phase of the project where the use cases are defined and the various requirements are collected, and which will constitute the building blocks of the Analytics layer of the project, in particular, these trained ML and DL models will be used for digital twins and applications ready to be integrated by third-party applications. These ML models, after the development and training phase with the data created for this purpose, need a process of evaluation and refinement before the models can be served at the higher level. The evaluation of each model will be based on several criteria such as its portability/scalability, its training performance and the accuracy of its results. To measure the performance of ML models, specific performance enhancement libraries to accelerate deep learning networks on AI will be used. In addition, each model may also be evaluated by human inspectors or highly experienced workers in the respective organisation to detect anomalies and increase the predictive quality of the model. At the end of this evaluation and refinement phase, the models will be served to the Analytics layer via a dedicated serving framework that will be implemented taking into consideration specific requirements identified at this stage of the project.

3.3.1 Requirements

The main requirements that emerged during the first part of the project were the basis on which to make the first choices about the environment to be created to meet the needs of the model developers who, after the training period, wanted to test its quality and performance. Below are the main characteristics identified for the first release of the evaluation module.

Evaluation framework, main features/requirements:

- A common repository where the models to be tested can be saved and the different versions created during the refinement phase can be tracked.
- A common access point to the different datasets of the project, whether they come from resources of the different pilots or from sources outside the project.
- A common set of tools to test and evaluate the models and refine them if necessary, on the basis of the evaluations that emerge.
- A common set of tools to keep track of processing result logs and related metrics that can also be easily analysed through a graphical interface.
- An environment that can provide developers with the most common ML libraries and that can be easily updated with new libraries when needed.

As for the ML model serving module, requirements were collected concerning the environment to be created in order to satisfy the needs of the users of ML models which, after being evaluated, must be available for use by the Analytic layer services. Below are the main features / requirements identified for the first release of the serving module:

- Ease of use and the availability of a graphical web interface
- Web UI and related APIs for model registry and deployment management
- High-Performance online API serving and offline batch serving
- Support as many ML frameworks that are commonly used in model creation as possible.
- Cloud-native deployment

3.3.2 Model evaluation and Serving framework architecture

The model evaluation and serving framework is placed into the I-NERGY conceptual architecture within the I-NERGY AI Trained Models layer and are part of the AI Model Development phase. It will evaluate and serve the ML models already trained by the ML developers to the upper Energy Analytics Application layer to be used by the different services that will be developed during the project phases.

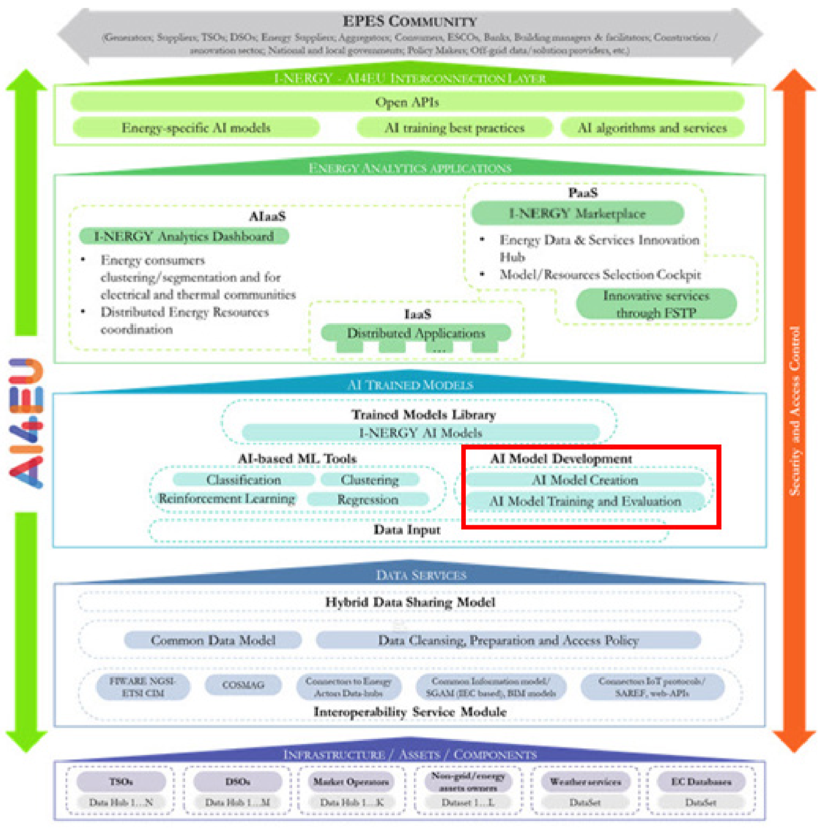

EPES COMMUNITY
(Generators; Suppliers; TSOs; DSOs; Energy Suppliers; Aggregators; Consumers, ESCOs, Banks, Building managers & facilitators; Construction / renovation sector; National and local governments; Policy Makers; Off-grid data/solution providers, etc.)
I-NERGY - AI4EU INTERCONNECTION LAYER
Open APIs
Energy-specific AI models
AI training best practices
AI algorithms and services
ENERGY ANALYTICS APPLICATIONS
AIaaS
I-NERGY Analytics Dashboard
Energy consumers clustering/segmentation and for electrical and thermal communities
Distributed Energy Resources coordination
PaaS
I-NERGY Marketplace
Energy Data & Services Innovation Hub
Model/Resources Selection Cockpit
Innovative services through FSTP
IaaS
Distributed Applications
...
AI TRAINED MODELS
Trained Models Library
I-NERGY AI Models
AI-based ML Tools
Classification
Clustering
Reinforcement Learning
Regression
AI Model Development
AI Model Creation
AI Model Training and Evaluation
Data Input
DATA SERVICES
Hybrid Data Sharing Model
Common Data Model
Data Cleansing, Preparation and Access Policy
FIWARE NGSI-ETSI CIM
COSMAG
Connectors to Energy Actors Data-hubs
Common Information model/ SGAM (IEC based), BIM models
Connectors IoT protocols/ SAREF, web-APIs
Interoperability Service Module
INFRASTRUCTURE / ASSETS / COMPONENTS
TSOs
Data Hub 1…N
DSOs
Data Hub 1…M
Market Operators
Data Hub 1…K
Non-grid/energy assets owners
Dataset 1…L
Weather services
DataSet
EC Databases
DataSet
AI4EU
Security and Access Control

Figure 26 I-NERGY conceptual architecture

The next image shows the architecture of the Evaluation Framework (the highlighted part of the AI TRAINED MODELS LIBRARY component in Figure 26) and the interaction between the different modules and layers. Model Evaluation and Serving Framework modules provide the already configured environment and tools for the ML models developer and users. This module is connected to the Data service layer via the database and the Kafka streaming module and will give the possibility to evaluate and serve the ML models provided by the Model Training Library that will share the already trained model via a dedicated shared repository.

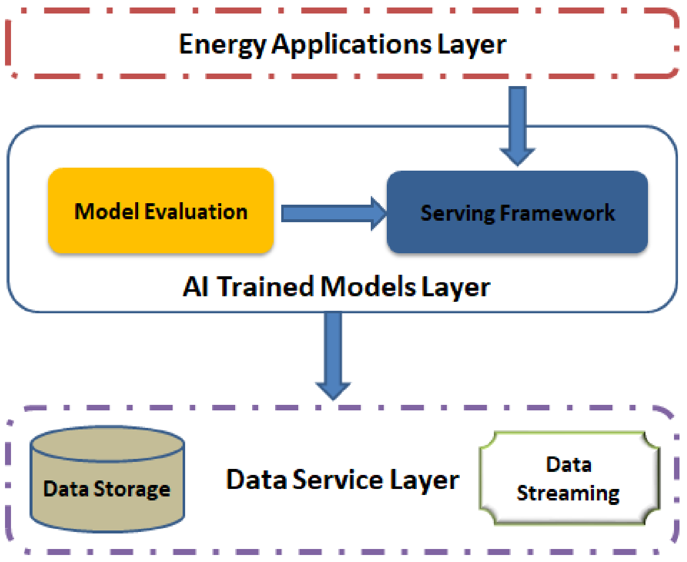


Figure 27 Model Evaluation and Serving Framework architecture

Based on the requirements and functionalities gathered in this first project phase and mentioned above, some possible software/frameworks and environment configurations were analysed. It was therefore decided to adopt specific solutions for both modules and to guarantee at the same time an interchange channel between the two to facilitate the transfer of the ML models evaluated and refined in the evaluation environment to the serving module. Below are the solutions identified and their main characteristics.

3.3.3 Model evaluation and Serving framework technological components

(i) **Model evaluation framework components**

Based on the first identified requirements the choice was made on common tools for testing and evaluation the already trained ML models installing and opportunely configuring the Anaconda Data Science Platform [45] that provides a set of tools able to provide to the ML developers and the data scientist and engineers the instrument for their evaluation and analysis. An analysis of the main used ML libraries has been done and at the time of writing, a set of these ML libraries has been installed and tested into the evaluation framework environment to permit the evaluation of the first test ML models. New libraries will be installed during the project life based on the incoming needs.

In detail, the following tools have been identified, installed and configured for the user needs:

- Anaconda platform with Jupyter Notebook
- Specific libraries that enable straightforward usage of trained ML models and related evaluation, such as Pandas, TensorFlow, Numpy, Keras, TensorBoard, Scikit-learn, Pytorch, Matplotlib, Seaborn etc.

- Python libraries to enable the connection to other modules and APIs such as Kafka Stream, Databases, Cloogy API, etc.

(ii) <u>Serving framework components</u>

Taking into account the first requirement identified for the Serving framework module the choice was made on the BentoML framework [46] that is the engine to be adopted for the I-NERGY serving framework module.

Below are the main characteristics of the Serving framework solution:

- Web UI to send requests such as prediction, and related APIs for model registry and deployment management. Yatai, the model management component of BentoML, is available via Web UI, CLI, Python API.
- Production-ready online API serving and adaptive offline batch serving.
- Possibility to serve multiple models.
- Different ML frameworks support such as: Tensorflow, PyTorch, Keras, XGBoost, FastText, CoreML, Spacy etc.
- Supports deployment with open-source platforms and technologies such as: Docker, Kubernetes, Kubeflow, etc. as well as one-click deployment with AWS Lambda, Azure Functions and others, and manual cloud deployment with AWS ECS, Google Cloud Run, Heroku.

3.3.4 Model evaluation and Serving framework deployment approach

The Evaluation framework is, in the first stage of the project, deployed in the traditional way, using Anaconda platform implementation for Centos 8 OS and by following the official guideline [47]. All necessary dependencies, libraries and packages are installed and configured on-premises.

Guide for Anaconda installation on CentOS includes the following steps:

- Requirements are privileged access or `sudo` capabilities, enough RAM, CPU power, and storage on the dedicated VMs, access to the terminal window, open HTTP(S) ports.
- Latest Stable Anaconda Version Installer can be downloaded with `curl` or `wget` tool and saved in the predetermined folder. Next, the verification of the downloaded file is needed.
- By entering the bash command in a terminal window the Anaconda Installer script is run. The configuration steps need to be executed as the installation progresses.
- To activate the Anaconda installation, set and load the path to the conda command.
- To verify the installation, use the conda command, such as conda info or conda --version.
- To run the Jupyter Notebook application, type the command: `jupyter notebook --notebook-dir=/opt/notebooks --ip='0.0.0.0' --port=8888 --no-browser --allow root &`. Note that /opt/notebooks is the folder where all notebooks, files, and data for the Evaluation framework will be placed.

- The token-based authentication is on by default, and it restricts access to the notebook server. This access token will be received upon the start of the Jupyter and should be kept secret.

The Serving framework's main part is BentoML. BentoML and all its components can be deployed using Docker and Docker Compose technologies. Yatai server, PostgreSQL database (or SQLite database) and Docker image for serving the services (containerised Model API Server) can be created based on the available instructions [48] and combining them with the known Docker and Docker Compose approaches. The technique of creating a bundle of ML models and serving them as a production API server is also available in the I-NERGY Serving Framework. All these tools are installed on the dedicated VM on-premises.

The steps for using Docker Compose are the following:

- Dockerfile can be created to enable easy setup of the application's environment and additional configuration of the base image. The `docker build` command is used to build an image from a given Dockerfile.
- Different frameworks and applications can be integrated using Doker Compose. A docker-compose.yml file allows services based on the applications to be configured, deployed, and run together.

Start up the applications by running the "`docker-compose up -d`" command.

Table 10 Example of docker-compose.yml file for BentoML services

```
version: "3.1"
services:
  bentoml:
    image: bentoml/model-server:latest
    command: bash -c "pip install wheel && pip install pandas && pip install sklearn
&& bentoml serve-guncorn /root/bentoml --enable swagger --yatai-url
217.172.12.204:8801"
    ports:
      - "8802:3000"
      - "8803:50051"
      - "8804:5000"
      - "8805:33513"
      - "8806:5000"
    volumes:
      - /root/bentoml/repository/TestClassifierMulti-
ple/20210715100649_A1C6F7:/root/bentoml
    networks:
      - yatai
  yatai-service:
    restart: unless-stopped
    image: bentoml/yatai-service:latest
    command: " --db-url=postgresql://postgres@yatai-db:5432/bentomldb --repo-base-url=/bentoml/repository"
    volumes:
      - /root/bentoml:/bentoml
    environment:
      - BENTOML_HOME=/bentoml
      - REPOSITORY_BASE_URL=/bentoml/repository
    depends_on:
      - yatai-db
    environment:
      - REPO_BASE_URL=/bentoml/repository
    ports:
      - "8800:3000"
      - "8801:50051"
    networks:
      - yatai
  yatai-db:
    image: library/postgres:9.6.20
    environment:
      - LC_ALL=C.UTF-8
      - POSTGRES_DB=bentomldb
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=
      - POSTGRES_HOST_AUTH_METHOD=trust
    volumes:
       - /data/bentoml/db:/var/lib/postgresql/data
    ports:
      - "8909:5432"
    networks:
      - yatai
networks:
```

```
yatai: {}
```

3.4 Security Framework for Data and trained Models

In Section 2.9, we discussed the challenges in providing secure data and models, identified existing attack types and mitigation techniques. As the development and lifecycle management of ML models is increasingly automated using MLOps frameworks, it is important for these frameworks to incorporate defense techniques some of the already identified by the community and summarised in Section 2.9, others still under work.

Currently, there are a number of available MLOps platforms [49] such as Amazon SageMaker, Azure Machine Learning, Google Cloud AI Platform, Metaflow, Paperspace, MLFlow, Algorithmia, TFX, Seldon, etc. Some of these are commercially hosted frameworks while others are available to the community as open-source projects that are under continuous development and extension. The ones that are commercially hosted are protected by standard commercial grade security and access control technologies while the others rely on standard security and access control frameworks. Another distinct open source framework that emphasises security is PySyft [50] that is more suitable for federated learning.

3.4.1 Requirements

For realising the security framework, the challenges identified in Section 2.9 need to be addressed by existing and emerging security components and libraries offering tools based on the defense techniques identified in the above-mentioned section.

3.4.2 Selected Approach, Tools and Justifications

For standard AAA-related security, BentoML is a model serving framework that ensures related security. Further, we will also consider adding Seldon [51] to BentoML for securing the ML API and thus addressing challenge 5 from Section2.9. For addressing challenges 1 and 3 we will also consider encryption using Python's hashlib[52], for addressing challenge 3 we will further consider cleverhans [53] library and standard software testing libraries and practices [54] for ML verification and testing. Privacy meter [55] will be further considered for addressing Challenges 4 and 5 discussed in Section 2.9.

4 Conclusion

In this document, we presented the first release of I-NERGY Models, AI methods and Tools deliverable. In this first edition of the document, we explored various related theoretical and technical backgrounds relating to I-NERGY Model development, evaluation and serving of models in a safe and secure manner to various AI services in the upper layer of the I-NERGY platform. Regarding the background, we categorically elucidated the following sub-topics: ML and ML pipeline, incremental ML, self-learning techniques in ML, distributed model training, ML inter-operability, transfer learning, model evaluation and serving techniques, model updating, access control techniques for data and trained models. Particular emphasis was laid on each of these sub-topics in terms of their meaning, existing approaches on how they are realised (including pros and cons where appropriate). We followed this up with a detailed overview of each of the following components: Incremental, Distributed and Self-learning Model, Cross-stakeholder Transfer Learning, Model Evaluation and Serving Framework, Security Framework for Data and Trained Models. These components are the bedrock upon which the I-NERGY trained models for various AI services needed in the pilot use cases are defined, trained, evaluated, and securely shared. We presented the purpose, requirements, proposed approach, required tools and justifications for each of the components.

Finally, considering the fact that the project is still at the initial stage, we envisage that major changes to the contents presented herein may be possible in the subsequent edition of the document as this is the first release of the *I-NERGY Models, AI methods and Tools* report.

References

[1] H. Miao, A. Li, L. S. Davis, and A. Deshpande, ‘Towards Unified Data and Lifecycle Management for Deep Learning’, in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, San Diego, CA, USA, Apr. 2017, pp. 571–582. doi: 10.1109/ICDE.2017.112.

[2] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, ‘iCaRL: Incremental Classifier and Representation Learning’, *ArXiv161107725 Cs Stat*, Apr. 2017, Accessed: Aug. 17, 2021. [Online]. Available: http://arxiv.org/abs/1611.07725

[3] P. Li, Z. Chen, L. T. Yang, J. Gao, Q. Zhang, and M. J. Deen, ‘An Incremental Deep Convolutional Computation Model for Feature Learning on Industrial Big Data’, *IEEE Trans. Ind. Inform.*, vol. 15, no. 3, pp. 1341–1349, Mar. 2019, doi: 10.1109/TII.2018.2871084.

[4] J. Liu, Y. An, R. Dou, and H. Ji, ‘Dynamic deep learning algorithm based on incremental compensation for fault diagnosis model’, *Int. J. Comput. Intell. Syst.*, vol. 11, no. 1, p. 846, 2018, doi: 10.2991/ijcis.11.1.64.

[5] C. Yao, J. Zou, Y. Luo, T. Li, and G. Bai, ‘A Class-Incremental Learning Method Based on One Class Support Vector Machine’, *J. Phys. Conf. Ser.*, vol. 1267, p. 012007, Jul. 2019, doi: 10.1088/1742-6596/1267/1/012007.

[6] Z. Yang, S. Al-Dahidi, P. Baraldi, E. Zio, and L. Montelatici, ‘A Novel Concept Drift Detection Method for Incremental Learning in Nonstationary Environments’, *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 1, pp. 309–320, Jan. 2020, doi: 10.1109/TNNLS.2019.2900956.

[7] J. Li, Q. Dai, and R. Ye, ‘A novel double incremental learning algorithm for time series prediction’, *Neural Comput. Appl.*, vol. 31, no. 10, pp. 6055–6077, Oct. 2019, doi: 10.1007/s00521-018-3434-0.

[8] W. Yu and C. Zhao, 'Broad Convolutional Neural Network Based Industrial Process Fault Diagnosis With Incremental Learning Capability', *IEEE Trans. Ind. Electron.*, vol. 67, no. 6, pp. 5081–5091, Jun. 2020, doi: 10.1109/TIE.2019.2931255.

[9] W. Dinalankara, D. De Silva, and D. Alahakoon, 'Self learning and its implications for machine understanding', in *2008 4th International Conference on Information and Automation for Sustainability: ICIAfS ; Colombo, Sri Lanka, 12 - 14 December 2008*, Piscataway, NJ: IEEE, 2008.

[10] D. Kasenberg, 'AI Ethics: Inverse Reinforcement Learning to the Rescue?', Aug. 04, 2017. https://dkasenberg.github.io/inverse-reinforcement-learning-rescue/ (accessed Aug. 17, 2021).

[11] G. Führ, A. Hallawa, R. Leupers, G. Ascheid, and J. F. Eusse, '3D Optimisation of Software Application Mappings on Heterogeneous MPSoCs', in *Architecture of Computing Systems – ARCS 2020*, vol. 12155, A. Brinkmann, W. Karl, S. Lankes, S. Tomforde, T. Pionteck, and C. Trinitis, Eds. Cham: Springer International Publishing, 2020, pp. 56–68. doi: 10.1007/978-3-030-52794-5_5.

[12] A. Burger, P. Urban, J. Boubin, and G. Schiele, 'An Architecture for Solving the Eigenvalue Problem on Embedded FPGAs', in *Architecture of Computing Systems – ARCS 2020*, Cham, 2020, pp. 32–43. doi: 10.1007/978-3-030-52794-5_3.

[13] T. Becker and T. Schüle, 'Evaluating Dynamic Task Scheduling with Priorities and Adaptive Aging in a Task-Based Runtime System', in *Architecture of Computing Systems – ARCS 2020*, vol. 12155, A. Brinkmann, W. Karl, S. Lankes, S. Tomforde, T. Pionteck, and C. Trinitis, Eds. Cham: Springer International Publishing, 2020, pp. 17–31. doi: 10.1007/978-3-030-52794-5_2.

[14] S. Wang, 'DISTRIBUTED MACHINE LEARNING'. Accessed: Aug. 17, 2021. [Online]. Available:

https://www.academia.edu/35877759/DISTRIBUTED_MACHINE_LEARNING_STANLEY_WANG_SOLUTION_ARCHITECT_TECH_LEAD_at_SWANG68

[15] D. F. Baruffa and S. Sobhee, 'Distributed DL/ML Solutions for HPC systems', *Intel AI*, p. 48, 2019.

[16] A. Sergeev and M. Del Balso, 'Horovod: fast and easy distributed deep learning in TensorFlow', *ArXiv180205799 Cs Stat*, Feb. 2018, Accessed: Aug. 17, 2021. [Online]. Available: http://arxiv.org/abs/1802.05799

[17] Y. Li, Z. Zeng, J. Li, B. Yan, Y. Zhao, and J. Zhang, 'Distributed Model Training Based on Data Parallelism in Edge Computing-Enabled Elastic Optical Networks', *IEEE Commun. Lett.*, vol. 25, no. 4, pp. 1241–1244, Apr. 2021, doi: 10.1109/LCOMM.2020.3041453.

[18] M. Jansen, V. Codreanu, and A.-L. Varbanescu, 'DDLBench: Towards a Scalable Benchmarking Infrastructure for Distributed Deep Learning', in *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*, Atlanta, GA, USA, Nov. 2020, pp. 31–39. doi: 10.1109/DLS51937.2020.00009.

[19] D.-L. Lin and T.-W. Huang, 'A Novel Inference Algorithm for Large Sparse Neural Network using Task Graph Parallelism', in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA, Sep. 2020, pp. 1–7. doi: 10.1109/HPEC43674.2020.9286218.

[20] D. Yang, J. Liu, and J. Lai, 'EDGES: An Efficient Distributed Graph Embedding System on GPU clusters', *IEEE Trans. Parallel Distrib. Syst.*, pp. 1892–1902, 2020, doi: 10.1109/TPDS.2020.3041219.

[21] D. Zheng *et al.*, 'DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs', in *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, GA, USA, Nov. 2020, pp. 36–44. doi: 10.1109/IA351965.2020.00011.

[22] S. Kunde, A. Pandit, and R. Singhal, 'Benchmarking performance of RaySGD and Horovod for big data applications', in *2020 IEEE International Conference on Big Data (Big Data)*, Atlanta, GA, USA, Dec. 2020, pp. 2757–2762. doi: 10.1109/BigData50022.2020.9378470.

[23] M. Jansen, V. Codreanu, and A.-L. Varbanescu, 'DDLBench: Towards a Scalable Benchmarking Infrastructure for Distributed Deep Learning', in *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*, Atlanta, GA, USA, Nov. 2020, pp. 31–39. doi: 10.1109/DLS51937.2020.00009.

[24] 'PyTorch'. https://www.pytorch.org (accessed Aug. 17, 2021).

[25] A. Shridhar, P. Tomson, and M. Innes, 'Interoperating Deep Learning models with ONNX.jl', *JuliaCon Proc.*, vol. 1, no. 1, p. 59, Aug. 2020, doi: 10.21105/jcon.00059.

[26] Q. Qu, 'ONNX : convert trained pytorch model to tensorflow model', Aug. 16, 2019. https://quq99.github.io/blog/2019-08/onnx-convert-trained-pytorch-model-to-tensorflow-model/ (accessed Aug. 17, 2021).

[27] K. Chen *et al.*, 'A DNN Optimization Framework with Unlabeled Data for Efficient and Accurate Reconfigurable Hardware Inference', in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, Daegu, Korea, May 2021, pp. 1–5. doi: 10.1109/ISCAS51556.2021.9401409.

[28] 'NNEF - Neural Network Exchange Format (NNEF)', *The Khronos Group*, Oct. 04, 2016. https://www.khronos.org// (accessed Aug. 17, 2021).

[29] K. H. Lee, J. Park, S. Yoo, S. J. Yoon, and C. Sik Cho, 'An Implementation of NNEF-Darknet Neural Network Model Converter', in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South), Oct. 2020, pp. 1186–1188. doi: 10.1109/ICTC49870.2020.9289343.

[30] B. Seo, M. Shin, Y. J. Mo, and J. Kim, 'Top-down parsing for Neural Network Exchange Format (NNEF) in TensorFlow-based deep learning computation', in *2018 International*

Conference on Information Networking (ICOIN), Chiang Mai, Thailand, Jan. 2018, pp. 522–524. doi: 10.1109/ICOIN.2018.8343173.

[31] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, 'A Survey on Deep Transfer Learning', *ArXiv180801974 Cs Stat*, Aug. 2018, Accessed: Aug. 17, 2021. [Online]. Available: http://arxiv.org/abs/1808.01974

[32] S. J. Pan and Q. Yang, 'A Survey on Transfer Learning', *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010, doi: 10.1109/TKDE.2009.191.

[33] F. Zhuang *et al.*, 'A Comprehensive Survey on Transfer Learning', *Proc. IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021, doi: 10.1109/JPROC.2020.3004555.

[34] H. Bousbiat, C. Klemenjak, and W. Elmenreich, 'Exploring Time Series Imaging for Load Disaggregation', in *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, Virtual Event Japan, Nov. 2020, pp. 254–257. doi: 10.1145/3408308.3427975.

[35] B. Bertalanic, M. Meza, and C. Fortuna, 'Time Series Imaging for Link Layer Anomaly Classification in Wireless Networks', *ArXiv210400972 Cs Eess*, Apr. 2021, Accessed: Aug. 17, 2021. [Online]. Available: http://arxiv.org/abs/2104.00972

[36] S. Dilmaghani, M. R. Brust, G. Danoy, N. Cassagnes, J. Pecero, and P. Bouvry, 'Privacy and Security of Big Data in AI Systems: A Research and Standards Perspective', in *2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, Dec. 2019, pp. 5737–5743. doi: 10.1109/BigData47090.2019.9006283.

[37] Y. Mirsky *et al.*, 'The Threat of Offensive AI to Organizations', *ArXiv210615764 Cs*, Jun. 2021, Accessed: Aug. 17, 2021. [Online]. Available: http://arxiv.org/abs/2106.15764

[38] F. Regazzoni, P. Palmieri, F. Smailbegovic, R. Cammarota, and I. Polian, 'Protecting artificial intelligence IPs: a survey of watermarking and fingerprinting for machine learning', *CAAI Trans. Intell. Technol.*, vol. 6, no. 2, pp. 180–191, Jun. 2021, doi: 10.1049/cit2.12029.

[39] Huawei, 'AI Security White Paper'. Accessed: Aug. 17, 2021. [Online]. Available: https://www-file.huawei.com/-/media/corporate/pdf/trust-center/ai-security-whitepaper.pdf

[40] E. Bertino, M. Kantarcioglu, C. G. Akcora, S. Samtani, S. Mittal, and M. Gupta, 'AI for Security and Security for AI', in *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, Virtual Event USA, Apr. 2021, pp. 333–334. doi: 10.1145/3422337.3450357.

[41] 'Why TensorFlow'. https://www.tensorflow.org/about (accessed Aug. 17, 2021).

[42] *baidu-allreduce*. Baidu Research, 2021. Accessed: Aug. 17, 2021. [Online]. Available: https://github.com/baidu-research/baidu-allreduce

[43] *Horovod*. Horovod, 2021. Accessed: Aug. 17, 2021. [Online]. Available: https://github.com/horovod/horovod

[44] 'C-MAPSS Aircraft Engine Simulator Data | NASA Open Data Portal'. https://data.nasa.gov/dataset/C-MAPSS-Aircraft-Engine-Simulator-Data/xaut-bemq (accessed Aug. 31, 2021).

[45] 'Anaconda | The World's Most Popular Data Science Platform', *Anaconda*. https://www.anaconda.com/ (accessed Aug. 17, 2021).

[46] 'BentoML'. https://www.bentoml.ai/ (accessed Aug. 17, 2021).

[47] 'Installation — Anaconda documentation'. https://docs.anaconda.com/anaconda/install/index.html (accessed Aug. 17, 2021).

[48] 'Getting Started — BentoML documentation'. https://docs.bentoml.org/en/latest/quickstart.html (accessed Aug. 17, 2021).

[49] 'Best MLOps Platforms to Manage Machine Learning Lifecycle', *neptune.ai*, May 14, 2021. https://neptune.ai/blog/best-mlops-platforms-to-manage-machine-learning-lifecycle (accessed Aug. 17, 2021).

[50] *Code for computing on data you do not own and cannot see*. OpenMined, 2021. Accessed: Aug. 17, 2021. [Online]. Available: https://github.com/OpenMined/PySyft

[51] StevenReitsma, 'Secure machine learning APIs with Seldon and BentoML', *Steven Reitsma*, Jan. 11, 2021. https://reitsma.io/blog/secure-ml-model-apis-with-seldon-and-bentoml (accessed Aug. 17, 2021).

[52] 'hashlib — Secure hashes and message digests — Python 3.9.6 documentation'. https://docs.python.org/3/library/hashlib.html (accessed Aug. 17, 2021).

[53] *CleverHans (latest release: v4.0.0)*. CleverHans Lab, 2021. Accessed: Aug. 17, 2021. [Online]. Available: https://github.com/cleverhans-lab/cleverhans

[54] 'A Software Testing View on Machine Learning Model Quality | by Christian Kästner | Medium'. https://ckaestne.medium.com/a-software-testing-view-on-machine-learning-model-quality-d508cb9e20a6 (accessed Aug. 17, 2021).

[55] 'GitHub - privacytrustlab/ml_privacy_meter: Machine Learning Privacy Meter: A tool to quantify the privacy risks of machine learning models with respect to inference attacks, notably membership inference attacks'. https://github.com/privacytrustlab/ml_privacy_meter (accessed Aug. 17, 2021).