Industrial Machine Learning for Enterprises

Deliverable D3.1

Baseline methods and techniques for advanced model engineering

Project title:	IML4E
Project number:	20219
Call identifier:	ITEA AI 2020
Challenge:	Safety & Security

Work package:	WP3
Deliverable number:	D3.1
Nature of deliverable:	Report
Dissemination level:	PU
Internal version number:	1.0
Contractual delivery date:	2022-01-31
Actual delivery date:	2022-02-04
Responsible partner:	University of Helsinki

Contributors

Editor(s)	Mikko Raatikainen (University of Helsinki)
Contributor(s)	Mikko Raatikainen (University of Helsinki), Harry Souris (Silo AI), Juhani Kivimäki (University of Helsinki), Lalli Myllyaho (University of Helsinki), Ville Kukkonen (Granlund), Davor Stjelja (Granlund), Gábor Gulyás (Vitarex Studio), Tamás Csarnó (Vitarex Studio), Dorian Knoblauch, Mariia Kucheiko, Abhishek Shresta (Fraunhofer FOKUS)
Quality assuror(s)	Gábor Gulyás (Vitarex Studio), Tamás Csarnó (Vitarex Studio), Harry Souris (Silo AI)

Version history

Version	Date	Description
1.0	Feb 3, 2022	Published version

Abstract

The purpose of this document is to provide the baseline for the IML4E project together with other deliverable D2.1. This document covers MLOps, reuse, AutoML, validation, verification, testing, debugging, maintenance, and monitoring. The document describes or defines the key concepts and terms in order to form a shared understanding within the project as well as elaborates the technological landscape with respect to the state of the art and practice of methods and tools that are of interest for the IML4E project. Finally, a summary as a technological baseline for IML4E projects is summarized pointing out areas that will be addressed during the project.

Keywords

MLOps, reuse, AutoML, validation, verification, testing, debugging, maintenance, monitoring

Executive Summary

The purpose of this document is to provide the baseline for the IML4E project together with other deliverable D2.1. This document describes or defines the key concepts and terms in order to form a shared understanding within the project as well as elaborates the technological landscape with respect to the state of the art and practice of methods and tools that are of interest for the IML4E project. Finally, a summary as a technological baseline for IML4E projects is outlined pointing out gaps in state of the art that will be addressed during the project.

Reuse in ML system differentiates intra-organization reuse within a system's life-cycle and between different deployment instances as well as broader extra-organizational reuse in terms of open-source reuse. The IML4E project will primarily focus on intra-organizational reuse. Additionally, the feasibility to apply AutoML, which aims to minimize human assistance in ML tooling, will be considered over the course of the project.

The specific focus in the IML4E project will be on various verification and validation activities meaning that the system is built correctly, and the right system is built, respectively. This covers both the ML model and ML system viewpoints as well as debugging, testing and continuous validation as activities. The contributions are about conceptual understanding and advancing solution approaches.

Respectively, another key focus will be in monitoring and management of a deployed ML model and system. This requires seamless integration as a part of overall MLOps architecture and processes.

Table of contents

1 Introduction

1.1 Role of this Document

The purpose of this document is to provide the baseline for the IML4E project. The document focuses on ML model engineering being parallel with the data engineering-focused deliverable "D2.1 Baseline methods and techniques for data collection, processing, and valorisation". This document describes or defines the key concepts and terms in order to form a shared understanding as well as outlines the technological landscape with respect to the state of the art and practice of methods and tools that are of interest for the IML4E project. Finally, a summary as a technological baseline for IML4E projects is outlined along with future directions.

1.2 Intended Audience

The intended audience of the present document is composed primarily of the IML4E consortium for the purpose of capturing the baseline of the project that the project will advance. However, this document is public and can provide an overview of the current practices to a reader. This document describes technologies for the technically oriented audience rather than the general public or layman.

1.3 Definitions and Interpretations

The terms used in this document have the same meaning as in the contractual documents referred in [FPP] with Annexes and [PCA] unless explicitly stated otherwise.

1.4 Applicable Documents

Reference	Referred document
[FPP]	IML4E – Full Project Proposal 20219
[PCA]	IML4E Project Consortium Agreement
[D2.1]	Baseline methods and techniques for data collection, processing, and valorisation

Table 1 – Contractual documents.

2 Basic Concepts

This section provides the definitions and descriptions of the basic concepts related to model engineering. The reference model (Google 2022a) used in the IML4E project is show in Figure 1.

Artificial Intelligence (AI) consists of all technical aspects that aim to get computers to imitate intelligent behaviour observed in humans (Russel 2002) including machine learning, natural language processing (NLP), language synthesis, computer vision, robotics, sensor analysis, optimization, and simulation.

A subclass of AI is Machine Learning (ML) that consists of techniques that enable computers to change their functionality based on given information (e.g., sensor data or training data), thus improving their behaviour for a given goal. ML techniques include decision trees, neural networks, support vector machines, and more. Machine learning can be categorized into supervised, unsupervised and reinforcement learning (Hinton and Sejnowski, 1999).

Supervised Learning utilizes training data for classification. The training data contains the desired outputs that are trained to the machine learning solution (Hinton and Sejnowski, 1999). Training data is usually formed manually by humans, collected automatically from empirical outcomes, such as weather data, or can be transferred by previously trained networks that could be labelled as teacher networks.

Unsupervised learning uses raw data for classification. The machine learning solution constructs its own outputs and predictions of classification based on the given input data (Hinton and Sejnowski, 1999).

Reinforcement Learning uses trial and error based on data made on the fly by an oracle, such as a repeatable simulation or a game, to find the optimal outputs for the machine learning solution (Hinton and Sejnowski, 1999).

Neural Networks (NNs) are a part of ML. NNs are computer programs inspired by biological neural network processes (Goodfellow et al., 2016) consisting of perceptrons, convolutional neural networks, recurrent neural networks, Boltzmann machines, deep neural networks, and many more. Basic NNs with one to a few layers of neurons usually require user assistance in forming classification classes.

Deep Neural Networks (DNNs) are a part of NN (Goodfellow et al., 2016). A DNN is a neural network that consists of multiple layers providing the DNN with the ability to form new classification classes regardless of human interference.

For clarity, we make a difference between

- A ML model as depicted above representing a component that has been trained based on data
- A ML system performs a purposeful function in a realistic context and includes a ML model and other supporting software components.

In practice, a ML system includes one or more ML models as a key part of the ML system.

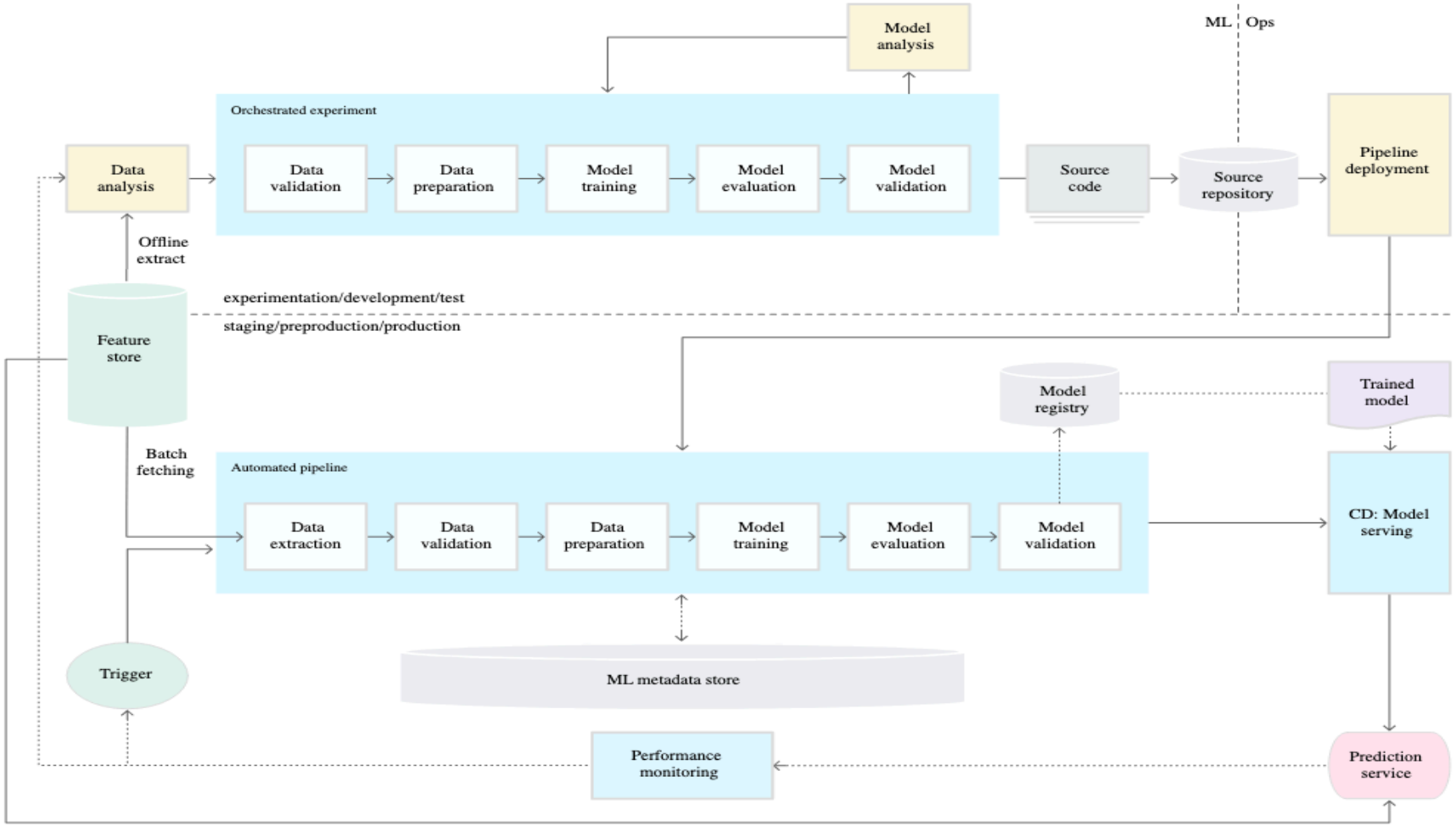

Figure 1 – The reference MLOps model in the IML4E project (Google 2022a) (Licenced under CC BY 4.0 https://creativecommons.org/licenses/by/4.0/)

3 Transfer Learning, Model Reuse and AutoML

3.1 Transfer Learning and Model Reuse

Transfer learning and model reuse can be divided into internal intra-organization and external extra-organization approach. This is likewise as software reuse in software product lines and open-source components, respectively.

In the case of intra-organizational reuse, we differentiate inter-product reuse, which is a part of and elaborated later in the case of model maintenance. Essentially, a model is deployed to a product or system to replace the existing model, e.g., because of model drift or decay. On the other hand, model reuse can take place between different deployments. For instance, edge deployment meaning that deployment is made to the customer's computing nodes can take advantage of the same model in different deployment scenarios. Edge deployment conveys many challenges related to non-functional characteristics, such as scalability and privacy.

Traditionally machine learning models have been hidden behind technology industry walls. Recently, however, we have seen a significant introduction of multiple new open-source machine learning models. These shared off-the-shelf pre-trained open-source machine learning models, frameworks, and datasets provide competitive state of the art capabilities in terms of performance, cost-effectiveness, and adaptability in different application domains when applied to new machine learning tasks. This may provide affordable new avenues for software engineers, researchers, students, businesses, and private enthusiasts to help reap the benefits of available data without requiring them to invest work on reinventing the wheel (Zhou 2016). Modular Neural Networks (MNNs) are another type under the NN category. A MNN is a network, that consists of multiple independent neural networks (modules) managed by some intermediary (program) that inputs values to each network, and takes their results (Hrycej, 1992) in some order or structural manner. While our ongoing systematic literature review (still unpublished work) indicates exponential interests in reuse of open ML models especially in academic settings, it is opposite in the case MNNs that show quite steady popularity over the years.

3.2 AutoML

The concept of AutoML can be defined in various ways. According to Yao et al. (2018), AutoML is the intersection of automation and machine learning, seeking to maximize performance of machine learning tools with regards to different configurations, in such a way that human assistance is minimized. This is done within confinements of a limited computational budget.

AutoML techniques can be applied to some or all of the components of the machine learning pipeline. These include data preparation, feature engineering, model generation and model evaluation (He et al. 2021). In data preparation AutoML tools can be used to collect, clean, augment and label data. Automated feature engineering seeks to either find the best representative subset of features, construct new features out of the existing ones or extract features using dimensionality reduction techniques. Model generation begins by selecting a model out of the space of available model structures. In deep neural networks optimal architecture can be sought using neural architecture search. After the model and its architecture are selected, the hyperparameter values can be set using automated hyperparameter tuning. Finally, automated methods can be applied to model evaluation.

There are many existing frameworks for AutoML. Most of these frameworks can be integrated with common machine learning libraries (sklearn, pytorch, tensorflow) with ease. Some of them can be used to automate only the hyperparameter tuning phase. Others seek to automate most if not the entire machine learning pipeline. Some open source AutoML systems are presented and evaluated in published benchmarks (Gijsbers et al., 2019; Zöller & Huber, 2021). A comprehensive comparison of existing neural architecture search methods can be found in He et al. (2021). Though AutoML solutions can provide easy to access and fast to implement solutions in generating machine learning pipelines, they are not yet able to beat human experts in terms of performance (Zöller & Huber, 2021).

4 Model Troubleshooting, Debugging and Testing

4.1 Basic Concept

SWEBOK (Bourque & Fairley, 2014) makes a difference between:

- "**Validation** is an attempt to ensure that the right product is built—that is, the product fulfills its specific intended purpose".
- "**Verification** is an attempt to ensure that the product is built correctly, in the sense that the output products of an activity meet the specifications imposed on them in previous activities."

In other words, a validation process is for assessing whether or not the final product or ML System in the case of IML4E works as it is supposed to, or whether or not "the right product was built." This is the typical division appearing in various engineering disciplines, including software engineering.

It is important to note that what we mean by validation is not the same as what is meant with *ML model validation* often in the context of ML. That is, ML model validation often means testing the generalization ability of a model (Wang et al. 2013). We do not rule out that ML model validation could be used by some as a validation method for the entire system, but it is not what we strictly mean by validation. In a sense, ML model validation means practically the same as ML system verification (Zhang et al. 2020, Breck et al. 2017, Myllyaho et al. 2021).

A dependable system delivers correct service consistently, does not suffer from long periods of down-time, and is easily corrected and altered. Threats to dependability originate from failures, errors, and faults (Avizienis et al. 2004). Failures are deviations from the desired service. Failures result from propagating errors, i.e., incorrect functioning of the system. Errors are caused by faults that are defects in system's components (software or hardware).

Dependability can be reached by diminishing these threats (Avizienis et al. 2004). Two means of diminishing threats are *fault prevention* and *fault tolerance*. Fault prevention aims at not introducing faults into systems involving typically testing, whereas fault tolerance aims at a system design such that occurring errors are stopped from propagating and causing system failures. In software systems, fault tolerance is achieved by error detection and error recovery (Knight 2012). As faults are inherently possible or even present in ML models with no 100% accuracy, preventing ML system failures needs both fault prevention and fault tolerance.

4.2 ML System Troubleshooting and Misbehaviour

ML systems have peculiar characteristics inherited from the included ML models. That is, ML models are characterized as non-deterministic statistical approximations by their nature and they are bound to function completely correctly only a certain portion of the time (Ramanathan et al. 2016). Therefore, we focus on only issues stemming from the ML model rather than the system in general or to software around the ML model specifically. The figure below is a result of an interview study about fault tolerance (Myllyaho et al. 2022) depicting the different origins of misbehaviour in a ML system. Poor input means that the input data is somehow broken or incomplete. Concept drift means that input changes over time to something else than was expected originally, such as different distribution. Faulty deployment means that the input is broken by the system before it enters the model, or that the model's output is broken in the system, for example, by a buggy interface. Buggy or inaccurate model refers to issue in the ML model itself while misuse of model's results means that the ML model is used incorrectly or misunderstood in the ML system.

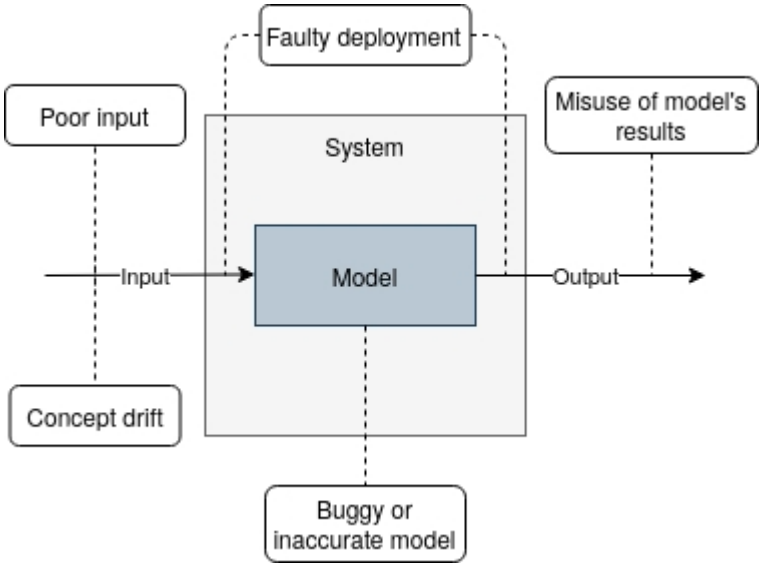

Figure 2 – Concepts of misbehaviour and their placement in relation to the system (Myllyaho et al. 2022).

4.3 Debugging

Unlike traditional software, which contains a large amount of code, machine learning tasks use a relatively small amount of code but a large amount of data. Just like in traditional program development, where developers look for bugs in the code, machine learning developers look for problems in their data and in their models. Given the frequency of program errors and the potentially serious consequences, debugging is very important when developing an ML model. However, there are fewer tools available for debugging and fixing errors in the data than for traditional code debugging.

Machine learning systems are often used as a black box. This means that while a model can provide accurate predictions, we cannot necessarily unambiguously explain the logic behind these predictions. Machine learning models usually do not provide an explanation for their decisions. Because of this, debugging the output of machine learning models is a difficult task.

Machine learning model debugging aims to find and fix problems in Machine learning systems. In addition to newer innovations, the common practice borrows from model risk management, traditional model diagnostics, and software testing. Model debugging attempts to test ML models like code and to probe sophisticated ML response functions and decision boundaries to detect and correct accuracy, fairness, security, and other problems in ML systems (Google 2022b).

The following are commonly used debugging techniques:

- **Sensitivity analysis**, sometimes called “What-if” analysis is a statistical method, which is able to estimate how sensitive a model is to changes in input parameters from their nominal values. We can simulate data that represents interesting scenarios, then see what kind of predictions the model makes in those scenarios. Because it’s difficult to know how a nonlinear ML model will react to new data that it did not see during training, it can be useful to conduct sensitivity analysis on all of our important ML models (Kenton 2022).
- **Residual analysis** is a numeric evaluation method, which measures the difference between the known true outcome and what a ML model predicted to be true. Residual plots can be very helpful in determining where the ML model is making a mistake like misclassifying data points. Residuals are crucial when it comes to evaluating the quality of a model. When residuals are zero, it means that the model prediction is perfect. More or fewer values mean less accuracy of the model.
- **Benchmark models**: A benchmark model is the most easy-to-use, stable, reliable, transparent, and interpretable model that you can compare your model with. A benchmark model can be a simple linear model, decision tree, or a previously existing and well-understood ML model. It's best practice to

compare the newly developed ML model's performance to a known benchmark model. Benchmark models can be useful debugging tools. You can ask questions like: “what predictions did my ML model get wrong that my benchmark model got right, and why?”.

- **Model assertions** help improve or override model predictions in real-time. In the context of machine learning modelling, a business rule can help the model meet its specified goals by detecting when it fails predictions or behaves unexpectedly. ML developers can specify constraints on model outputs, e.g., cars should not disappear and reappear in successive frames of a video. Model assertions can be exact or “soft,” i.e., probabilistic. Model assertions can be used in ML debugging like in runtime monitoring, in performing corrective actions, and in collecting “hard examples” to further train models with human labelling or weak supervision (Kang et al. 2018).
- **Metamorphic Testing.** The concept of Metamorphic testing (Chen et al. 1998) tries to solve the problem of testing systems that their output cannot be predefined for a given input. This problem is described as the test oracle problem (Barr et al 2015). In Metamorphic testing, practitioners define relations between inputs and outputs. Those defined relations are called Metamorphic relations (Zhang et al. 2019). Once those relations are defined practitioners generate test cases with modifications to the corresponding inputs and they expect relevant updates in the outputs. If the outputs do not follow the expected update pattern, then practitioners should mark the testcase as failed. Metamorphic testing finds usage in machine learning systems, due to the fact that in machine learning system the test oracle problem is met. Several studies and publications have been conducted the last years on the usage of Metamorphic testing in ML systems.

4.4 Black Box Testing with VALICY

With the increasing use of AI in applications there is an increasing need for novel validation techniques. For example, one specific case where existing software development techniques cannot be fully applied anymore is the autonomous vehicle. Especially, the NCAP approach (Euro NCAP 2022) assumes a definition of edge cases, i.e., the most critical cases to investigate. But this approach assumes there is an identifiable most critical case that must be excluded. Criticism of this method comprises the over-focusing on specific presumed “most critical” conditions to be fulfilled, while other cases might get too little attention. While the existing approach of the V-model described in ISO 26262 part 6 (ISO 26262-6) is comprehensive for software development in the application at the moment, there is the need to be able to estimate boundaries of the system in combination with AI as it introduces additional uncertainty that is not covered by the V model and must be evaluated (Salay et al. 2017).

With regard to this development, the need arises to be able to evaluate critical cases that depend on a multitude of input parameters and on one or more output target vectors. This is especially important regarding certification done by a safety audit. To be able to do so, there is an increasing need for virtual validation systems. At Spicetech, which is an IML4E project member, such a virtual validation tool – VALICY - is introduced and further developed within this project. VALICY consists of four major components:

- The VALICY core: The VALICY core is a Python-based AI framework that makes use of well-established machine learning and artificial intelligence libraries like sklearn, KERAS/Tensorflow and XGBoost. There are three levels in VALICY, a job level with all configurations for the test parameters as well as result parameters (upper, lower bounds, failure threshold)
- The VALICY database: To organize the data coming from and going to outside applications (black box) VALICY has an elaborate data model that manages all the job data, data to be validated by the black box with a proposed result from VALICY and the corresponding black box
- The VALICY API: VALICY has a REST-API based on SWAGGER UI that enables data transfer to and from VALICY to outside applications
- The VALICY frontend: in order to give a visual representation of the result, VALICY has a visualization frontend where three or more input parameters can be plotted with different projection techniques to

give an easily understandable visual representation of multi-dimensional parameter spaces. For a black box test configuration, the VALICY API, hosted and described on https://api.valicy.de/docs, has to be integrated. Spicetech can support the establishment of this integration.

VALICY operates in two modes, a black box testing mode with feedback from the black box and a mode for evaluation of real test data without timely feedback. The first mode including a coupled black box application that is continuously provided new parameter combinations to be tested as well as an expected VALICY result to be compared is mainly focused on in this project.

There are three levels of VALICY, a job level, the instance or AI level, and the run level. While all the major configurations are done on the job level, i.e., boundaries of the parameters under consideration as well as the threshold of the result, the instance level executes the predictions and each of these predictions form a run.

After configuration of job details, VALICY starts the virtual validation runs by first iterating through the regular grid of test parameters (min and max, or more if there are further subdivisions configured) makes guesses (initially no sophisticated result expectable) for the outcome, makes it available for export and expects feedback from the external black box application and evaluates if the VALICY proposal for the outcome was true or false. This is done on the instance level where different AI models are randomly configured with respect to the hyperparameters. The default mode is three different instance configurations which equals three different AI models and configurations that are competing for the best result (best prediction of black-box behavior). This is what we call the instance or AI level

After iterating through the regular grid, VALICY, or more exactly, the instances that could see the feedback from the black box have a first impression of the decision space and start making its own proposals for parameter combinations and the expected black box outcome. During each run VALICY aims to come as close to the decision surface as possible while still making correct predictions. The parameters are used for a black box run and the results are compared and transferred back to via the API to evaluate instance performance. If an instance performs well, it can predict a lot of results correctly until the decision surface becomes too complex for the AI model. When reaching a threshold of a pre-set value (e.g., 4) of wrong predictions in a row is exceeded, VALICY replaces the AI model with a newly configured model. A new instance trains on all the previously run results and has to prove its capability of predicting new critical parameter combinations close to the decision surface. This workflow is visualized in Figure 3.

The open challenges in VALICY are

- The scoring system for the instances and runs. The runs are the results of VALICY and are given a score that is higher the closer it was to the presumed decision surface. To account for the increasing complexity to identify critical parameter combinations of failure, there is an exponential decay term multiplied with the score, analogous to the radioactive decay. This ensures that earlier results are of less importance than later ones. However, this scoring system is under test, at the moment, and will be described in detail in a later deliverable.
- Determination of remaining uncertainties. The remaining uncertainties of areas not tested are estimated using appropriate mathematical functions and are under validation, at the moment. A detailed description will be presented in the next deliverable.

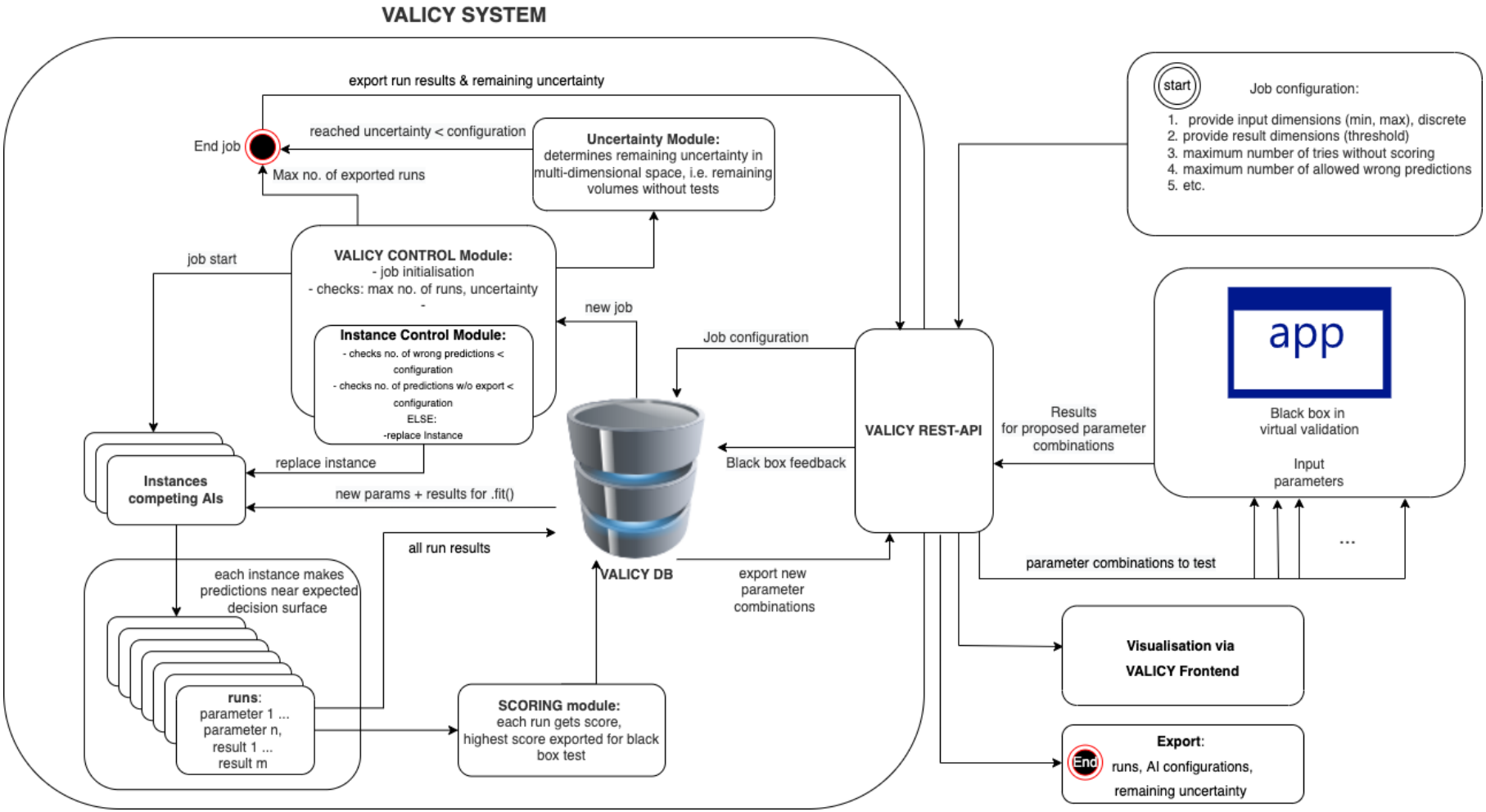

Figure 3 – General overview of the VALICY black box testing workflow

4.5 ML Model Verification

Whilst testing of ML systems is aiming at finding defects of their implementation, the purpose of verification is to provide a provable guarantee that the system will not misbehave under a broad range of circumstances. In contrast to classical software systems, which are verified against their specifications, due to the nature of ML methods and their unstructured operational environments, creating a specification for them is a tedious task and is still a subject of active research (Houben et al 2021).

One of the main requirements used in specifications of ML systems is their robustness against the so-called adversarial examples. Since their discovery in classification tasks of computer vision and NLP (Szegedy et al 2014), most of research has been focusing on these two domains, with former being much more popular than the latter. The problem is stated as follows: small and often hardly perceptible perturbations of an initially correctly classified object (image or text) may lead to its misclassification by the same NN. These perturbations might originate in sensor's malfunction, environmental conditions, etc. or they can be crafted on purpose, in which case they are referred to as adversarial attacks. Researchers have proposed various methods for adversarial defences and robust training approaches. However, they have usually been broken by yet stronger adversarial attacks. To make an end to this arms race, part of the research community has focused on formal verification methods, capable of providing a provable guarantee of a classifier's robustness. (Huang et al 2020) An algorithm that takes a NN, an object to classify, and a distance (a.k.a. magnitude of perturbation or robustness radius) as an input and outputs "not robust" if the NN is not robust to changes of a given object within given distance is called robustness verification algorithm. A verification algorithm that outputs "not robust" if and only if the NN is not robust is called complete. (Li et al 2020)

The most of complete verification approaches focus on feed-forward ReLU NNs (Li et al 2020). These types of NNs can be modelled as Boolean satisfiability problem (SAT), satisfiability modulo theories (SMT), mixed-integer linear programming (MILP), etc., and solved either by off-the-shelf solvers or by their NNs adapted versions (Huang et al, 2020). The former two approaches overapproximate a NN and iteratively tighten the approximation until either the property has been satisfied or discharged. MILP approaches allow to directly model a ReLU NN without any approximation. Unfortunately, the problem the abovementioned algorithms are attempting to solve is NP-hard. Thus, even though they use strategies like branch-and-bound to reduce computational complexity by skipping parts of solution space that cannot be optimal, they do not scale for state-of-the-art NNs. (Li et al, 2020)

In this case verification problem can only be solved with relaxations, meaning the algorithm returns a rather conservative or pessimistic certification. For the input ranging within the examined radius of perturbations, linear relaxation methods approximate the output of each neuron by linear lower and upper bounds and propagate them through the network layer by layer. If the output of the overapproximated model is only the true class of the examined object, the actual NN is robust. An alternative approach is to relax robustness verification to a semidefinite programming (SDP) problem, which can be solved in polynomial time. (Li et al, 2020)

The common drawback of these approaches is that they are often too conservative on naturally trained NNs. Hence, various robust training methods have been suggested to allow resulting NNs to be certified with nontrivial robustness guarantees. Inspired by adversarial training approaches, which are looking for such parameters of a NN that minimize maximal loss from misclassification of perturbated objects; robustness training searches for parameters that maximize the robustness bounds found by verification approaches. Some methods combine this objective with the regular training loss to balance standard top-1 accuracy and robustness accuracy. (Li et al, 2021)

Even though more suitable for mid-size NNs than complete verification approaches, the relaxation-based methods fail to deliver tight lower bounds for large NNs. In this case probabilistic verification approaches come in handy. What distinguishes them from previously discussed deterministic approaches is that they can mistakenly verify an NN to be robust against certain perturbations of a particular object, while in fact it is not. The probability of such a mistake is a hyperparameter of this procedure and can be limited to a very low level, e.g., 0.1% per object. These approaches are model-agnostic and thus scale well. (Li et al, 2021)

In practice, one of the best-performing probabilistic verification methods is randomised smoothing. It uses a NN's (a.k.a. base classifier's in this context) robustness to random noise to create a so-called smoothed classifier, which in turn is certifiably robust to adversarial perturbations. When queried at image x, the smoothed classifier returns whichever class the base classifier is most likely to return for random corruptions of x. Since it is

impossible for one algorithm (smoothed classifier) to predict the output of another algorithm (base classifier), smoothed classifiers often use Monte Carlo simulation, a.k.a. random sampling, of the base classifier in combination with a hypothesis test, to estimate the "winner-class" with the required confidence (e.g., 0,1% error probability). If the randomized classifier classifies the perturbated objects correctly (I.e., "winner-class" is the correct class of the non-perturbated object,) robustness radius of the smoothed classifier is calculated based on the properties of the distribution (e.g., Gaussian, Laplacian, etc.) which has been used to generate random noise. These approaches can certify state-of-the-art NNs for ImageNet dataset within acceptable time, e.g., that ResNet-50 is robust on 37% of test images within a L2 radius of 1.0. To give a sense of scale, a perturbation with this L2 radius could change one pixel by 255, ten pixels by 80, 100 pixels by 25, or 1000 pixels by 8. (Cohen et al. 2019)

4.6 ML System Validation

In terms of validation, it is often more relevant to discuss about ML system validation as noted above rather than only ML model validation. A systematic literature review (Myllyaho et al. 2021) identifies and synthesizes various ML system validation methods as summarized in Table 2 below. The validation methods are divided into classes in which simulation and trial include three and two subclasses, respectively.

Validation method	**Description**
Simulation:	
Fully virtual simulation	The deployment environment of the system is replicated with a virtual simulator
Hardware-in-the-loop simulation	A virtual simulator that contains also some non-virtual components
System-In-the-loop simulation	The system in an artificial environment
Trial:	
Trial in a real environment	The system is used as it would be used in the final deployment environment
Trial in a mock environment	The system is used in an environment that replicates an actual environment
Model-centred validation	Validation focusing solely on validating the model
Expert opinion	The system is assessed against expert's opinion

Table 2 – Validation methods for ML systems. Adapted from (Myllyaho et al. 2022).

5 Model Monitoring and Maintenance

5.1 Model Monitoring

Model monitoring or continuous validation observes a deployed system and is applied to ensure that the AI system also works in situations unseen during development or in training data, essentially testing that the AI system operates as intended, which is the definition of validation (Myllyaho et al. 2020). There is, however, no established conceptualization and terminology. For instance, Zhang et al. (2020) use the term “online testing” and Breck et al. (2017) use the term “monitoring testing”, whereas Myllyaho et al. (2021) use the term continuous validation, which is also adopted here. Continuous validation differs somewhat from the traditional way of seeing validation as something usually applied to the end product (IEEE 1991). Many AI systems continue to change their behaviour well after deployment or -- as discussed above -- are difficult to validate to a satisfactory degree before deployment, thus challenging the idea of an unchangeable end product to a degree. Considering this, post-deployment methods to ensure desired functionality and requirements are met are reported as continuous validation. While fault-tolerance measures, and other safety or quality assurance are included in continuous validation, monitoring is probably the most widely known and used concept of continuous validation.

A taxonomy for continuous validation is presented in Table 3 (Myllyaho et al 2021). The taxonomy is further developed and refined for model monitoring and fault tolerance and validated in a set of of interviews in (Myllyaho et al 2022). The table 4 below summarizes the results of the study.

Validation method	**Description**
Output and input restrictions	Hard limits given to the system input and output
Failure monitor	System’s malfunction detection
Safety channel	Backup component that takes over if the primary components are compromised
Redundancy	Critical components are duplicated in the system
Voting	Different components perform the same task and vote on the action to be taken

Table 3 – Continuous validation methods for ML systems monitoring in literature. Adapted from (Myllyaho et al. 2021).

Pattern	Variant	Pros	Cons	When to use
Input checker	Hard limits	Efficient in enforcing business rules and preventing poor quality data entering the model. Computationally light. Low cost.	Requires very special knowledge of the model. May solve only small problems.	When accuracy of single outputs is vital, and holey or out-of-range inputs cause problems.
	Novelty inputs	Shows holes in -- and can be later utilized as -- training data.	Difficult to say when a novel input is a problem.	When training data is difficult to gather otherwise.
Input distribution observer		Indicates changes in operations environment, and possible need of retraining.	Does not prevent single errors from happening.	In naturally evolving or changing input distributions. When input sources are prone to problems.
Output checker	Hard limits	Efficient in enforcing business and safety rules, and in spotting broken outputs. Computationally light. Low cost.	Requires very specific knowledge of the domain and system. Careless use leads to limiting results.	When business rules or safety regulations dictate a range of acceptable results, or unacceptable outputs can otherwise be recognised for certain.
Output distribution observer		Indicates changes in operations environment, and possible need of retraining.	Does not prevent single errors from happening.	In naturally evolving or changing input distributions. When input sources are prone to problems.
Model observers	Resource consumption	Could indicate suboptimal development in a continuously learning model.	Better suited for development phase and monitoring HW problems.	When testing the system after model deployment.
	Activation observers	Has potential in spotting erroneous input-output pairs.	Knowledge claims based on our data cannot be made.	
Redundant models	Recovery blocks with divergent models	Offers potentially effective fall-over possibilities.	Requires knowledge on when output is erroneous to be applied efficiently. Requires several models. Computationally heavy.	When high dependability is required, and other fall-over solutions are too simple for the problem.
	Input switch	Allows the best suited model to be used for each input.	Requires an enormous amount of knowledge about input space and used models. Requires several models.	When inputs can be in several forms or types, or inputs contain several sub-problems.

	Multi-armed bandit	Raises tolerance against changes in data distribution. Allows safer introduction of new models.	Requires several models.	In naturally evolving or changing input spaces. If product maturity has introduced several iterations of models.
	Voting	Potential to eliminate surprising input-output pairs.	Difficult to implement. Requires several models. Computationally heavy.	When sufficient data science skills are present in the project.
Fall-over options		Allow simpler, more predictable outcomes when errors are detected. Either brings in the user or developer to solve the problem or handles situations consistently. Low cost.	Results may not be as sophisticated as with finer solutions or models.	Often, if not always, as the last resort.

Table 4 – The patterns for fault tolerance (Myllyaho et al. 2022

5.2 Model Monitoring of High Number of Different Models

Model monitoring practices and research appear to generally focus on cases involving one or a handful of trained models. However, there are situations where many models need to be trained and used. Reasons for this may have to do with either latent variables that are difficult or impossible to capture, or considerations such as data confidentiality and edge deployments. As a practical example of latent variables, and a running example for the discussion in this subsection, let us consider predicting the heating energy consumption of a building based on the time and outdoor temperature. Buildings and their use vary a lot, with latent variables such as occupancy schedules that may be difficult to capture as inputs but can be learned by models.

For situations involving hundreds or thousands of models, open questions remain about methods and techniques for model monitoring. Approaches for managing these models could include fostering better understanding of the deployed models by using so-called model cards (Mitchell et al. 2019) as metadata descriptions for the models and exploiting the shared qualities of models such as grouping based on metadata or shared inputs. For example, in the building heating energy prediction example, buildings could be grouped based on their location, type of heating system, or function. Additionally, input data can be monitored and if a change in distribution happens, models which share same input can be checked to see if there is a drift.

Model cards (Mitchell et al. 2019) are short documents which provide data on specific trained model. They summarize the models with attributes such as:

- Model details: Training date, model version, algorithm
- Training and evaluation data: Data on which model was trained and evaluated, sample of data, features
- Metrics: Performance measures during evaluation, thresholds, uncertainty
- Intended use of the model: Purpose and object, constraints in which it works
- Ethical considerations
- Caveats and recommendations

Such model cards can be featured in the model monitoring dashboard and can also be used as a source of model related metadata needed for further analysis, for example, setting up the thresholds for detecting problems with a specific model. Comparing the latest error distributions against the known error distributions from the evaluation phase enables detecting if the models have drifted.

5.3 Model Maintenance

Model maintenance and maintaining ML production systems are challenging undertakings (D. Sculley et al. 2015). Only lately models have started to operate in production across multiple industries and there is not a standardized approach on how to maintain such of deployments.

A lot of practitioners base their maintenance operations in the continuously monitoring of the models and on triggering re-training operations when models are decaying. Specifically, the largest cloud providers in their AI solutions suggest model re-trainings when drifts are detected and as it is mentioned in a Google's article (Google 2022a) ".... to maintain your model's accuracy in production, you need to do the following:

- Actively monitor the quality of your model in production
- Frequently retrain your production models
- Continuously experiment with new implementations to produce the model

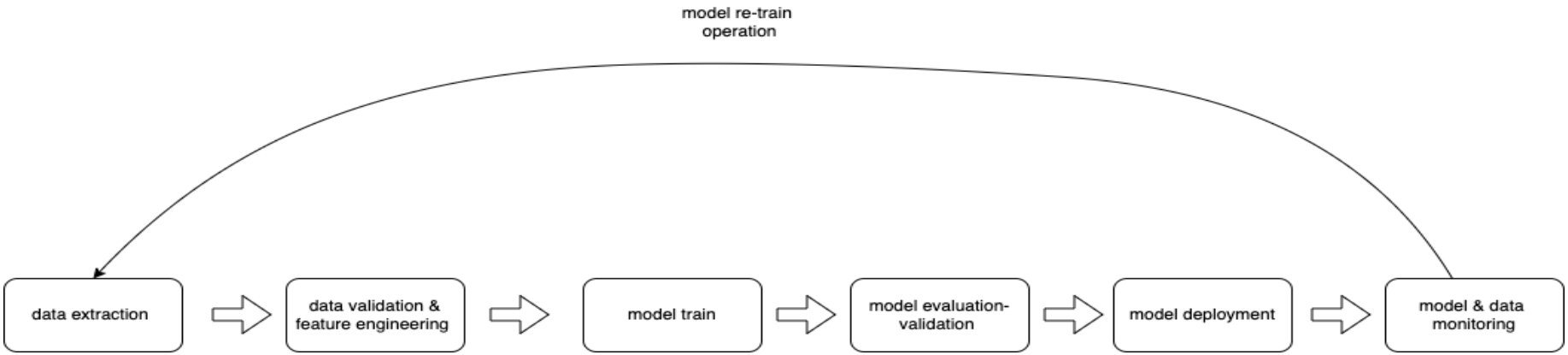

Figure 4 – High level view showing the sequence of different steps that make up an ML pipeline and how those steps are connected.

However, the challenge still remains on what metrics to use to detect bias and drifts to different machine learning use cases using different kind of data (tabular vs image data for example) and what should be the corresponding threshold values for those metrics so as to generate alerts and notifications to the ML monitoring and maintenance organisation. In addition, to continuously retrain model just because new data arrived may not be always the optimal scenario (Xin et al. 2021).

Some common metrics enterprises use to identify model decays and trigger more intelligent model updates are:

- Metrics related to the accuracy of the deployed model. For example, organizations are monitoring F-score in binary classification problems, and they deploy new re-trained versions of models when accuracy drops under a particular threshold.
- Metrics related with the statistical distance between the production (inference) input data and the data used in the training phase. By monitoring those metrics organizations monitor for data drifts which reveal how much outdated the model has become in contrast to the production (real word) data. The following figure describes a framework on how the statistical distances can be measured.

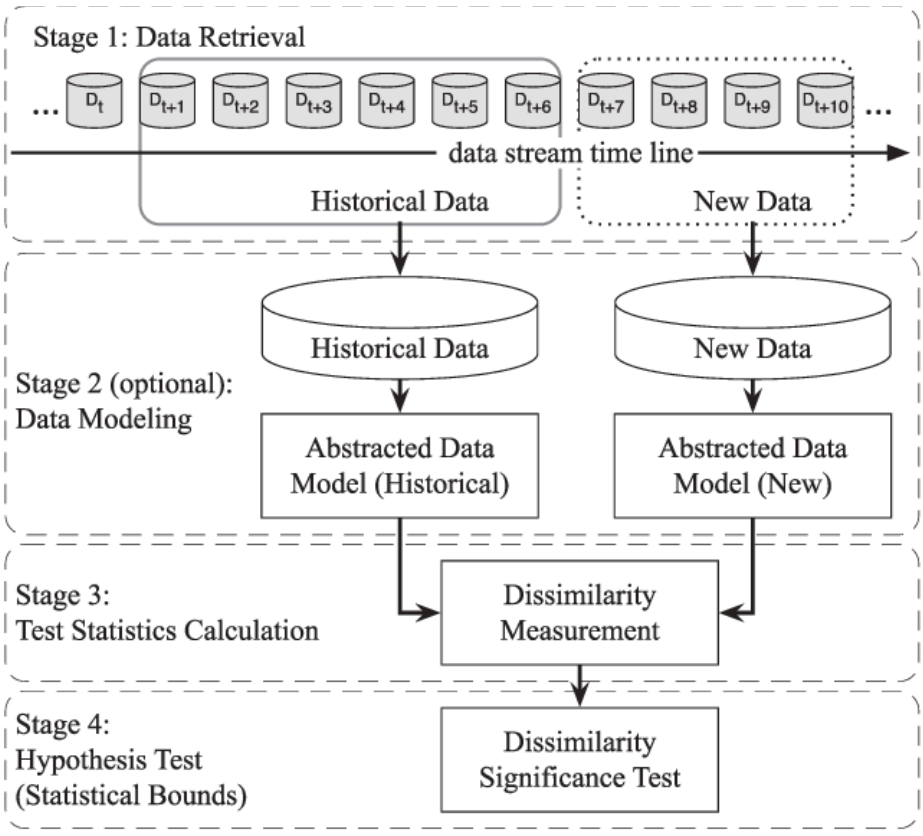

Figure 5 – Metrics related with the statistical distance between the production (inference) input data and the data used in the training phase (Lu et al. 2020).

One interesting type of data drift is the concept drift. Concept drift is related with the changes in the statistical properties of the target variable. In other words, with changes related with the distribution of the predictions. For example, organizations monitor if over a period of time one target class is met significantly more or less than what happened in a previous time period the model used to operate. Several statistical approaches and tools have been used in the area of detecting concept drifts with for example Kolmogorov–Smirnov test to be one of them.

In the following architecture steps 5, 6 and 7 are important for setting up the fundamentals of monitoring and maintaining ML solutions. Pipelines at step 5 should deploy model binary files together with information about the model version and information regarding the data sets used in training. Pipelines at step 6, 7 should save model predictions together with the input production data and together with the model and data versions that were related to the deployed model binary files that did the prediction. If we group all those information, then we are in position to run a variety of statistical tests and analysis of different performance metrics and hence we are in position to trigger model retrain operations when they are needed.

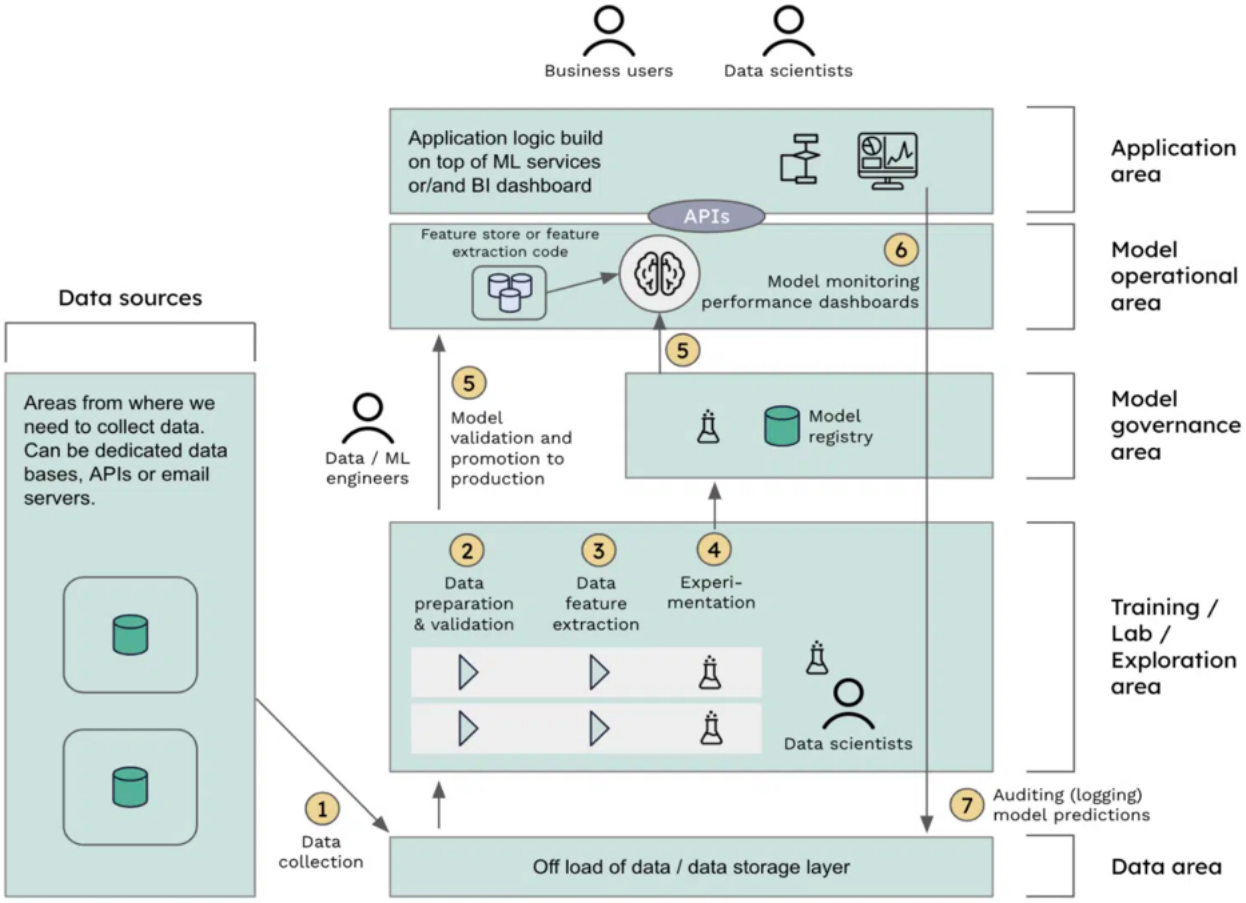

Figure 6 – A view of the MLOps architecture.

6 Conclusions and Baseline

This document has covered the technological landscape of the MLOps activities within IML4E project. In this section, we provide a summary by identifying the areas in which IML4E aims to primarily make novel contributions.

With respect to model reuse, the IML4E project aims to specifically address two topics. First, using, documenting, and storing an ML model for use in the long term, such as relacing the deployed model with another existing model version. Second, using the same model in different deployment instances, i.e., when the same software and model is used for several customers or different edge nodes. Both cases require efficient MLOps management processes for transparency and traceability.

Towards this end, the IML4E will focus on two basic scenarios of ML systems. First, the systems that have their own model, evolve on their own, and are deployed, e.g., as one digital service to the cloud. Second, systems that have large number of deployment instances that can be then deployed, e.g., to edge devices, and the instance have some differences, such as training data used.

Specific focus in the IML4E project will be on various verification and validation activities covering and differentiating both the ML model and ML system viewpoints. On one hand, the IML4E project aims to extend the conceptual understanding and characterize the root causes of misbehaviour as well as develop the existing approaches further and develop novel approaches. A quintessential part of this undertaking will be empirical testing.

Likewise in the case of monitoring and maintenance, the IML4E project will characterize and develop different approaches. The specific focus will be on the seamless integration as a part of entire MLOps architecture and processes. This architecture should enable the storing of the production data together with the related ML predictions together with the corresponding model and data versions that were used. Storing and grouping that information enables model performance monitoring both at the time a prediction is taking place but also in a batch fashion evaluating the model performance and the statistical changes over a period of time.

References

Avizienis, A., Laprie, J.C., Randell, B. and Landwehr, C., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE transactions on dependable and secure computing, 1(1), pp.11-33.

Barr, E.T., Harman, M., McMinn, P., Shahbaz, M. and Yoo, S., 2014. The oracle problem in software testing: A survey. IEEE transactions on software engineering, 41(5), pp.507-525.

Bourque, P. & Fairley, R. E., ed. 2014, SWEBOK: Guide to the Software Engineering Body of Knowledge , IEEE Computer Society.

Breck, E., Cai, S., Nielsen, E., Salib, M. and Sculley, D., 2017, December. The ML test score: A rubric for ML production readiness and technical debt reduction. In 2017 IEEE International Conference on Big Data (Big Data) (pp. 1123-1132). IEEE.

Chen, T.Y., Cheung, S.C., Yiu, S.M. 1998. Metamorphic testing: A new approach for generating next test cases", Technical Report HKUST-CS98-01 , Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong, arXiv:2002.12543.

Cohen, J., Rosenfeld, E. and Kolter, Z., 2019, May. Certified adversarial robustness via randomized smoothing. In International Conference on Machine Learning (pp. 1310-1320).

Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B. and Vanschoren, J., 2019. An open source AutoML benchmark. arXiv preprint arXiv:1907.00909.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.

Google, 2022a, MLOps: Continuous delivery and automation pipelines in machine learning, https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Google, 2022b, “Testing and Debugging” https://developers.google.com/machine-learning/testing-debugging

He, X., Zhao, K., Chu, X., 2021. AutoML: A survey of the state-of-the-art. In Knowledge-Based Systems, Volume 212, 2021.

Hinton, G.E. and Sejnowski, T.J. eds., 1999. Unsupervised learning: foundations of neural computation. MIT press.

Houben, S., Abrecht, S., Akila, M., Bär, A., Brockherde, F., Feifel, P., Fingscheidt, T., Gannamaneni, S.S., Ghobadi, S.E., Hammam, A. and Haselhoff, A., 2021. Inspect, understand, overcome: A survey of practical methods for ai safety. arXiv preprint arXiv:2104.14235.

Hrycej, T., 1992. Modular learning in neural networks: a modularized approach to neural network classification. Wiley

Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M. and Yi, X., 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review, 37, p.100270.

IEEE. 1991. IEEE standard computer dictionary: A compilation of IEEE standardcomputer glossaries, IEEE Std 610

ISO 26262. 2011. Road vehicles — Functional safety — Part 6: Product development at the software level

Kenton, W. 2022, Sensitivity Analysis. https://www.investopedia.com/terms/s/sensitivityanalysis.asp

Kang, D., Raghavan, D., Bailis, P., & Zaharia, M.A. 2018. Model Assertions for Debugging Machine Learning. Preprint. https://cs.stanford.edu/~matei/papers/2018/mlsys_model_assertions.pdf

Knight, J., 2012. Fundamentals of Dependable Computing. CRC Innovations in Software Engineering and Software Development: Boca Raton, FL, USA.

Li, L., Qi, X., Xie, T. and Li, B., 2020. Sok: Certified robustness for deep neural networks. arXiv preprint arXiv:2009.04131.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J. and Zhang, G., 2018. Learning under concept drift: A review. IEEE Transactions on Knowledge and Data Engineering, 31(12), pp.2346-2363.

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I.D. and Gebru, T., 2019, January. Model cards for model reporting. In Proceedings of the conference on fairness, accountability, and transparency.

Myllyaho, L, Raatikainen, M, Männistö, T, Nurminen, J K & Mikkonen, T. 2022, On Misbehaviour and Fault Tolerance in Machine Learning Systems', The Journal of Systems and Software, vol. 183.

Myllyaho, L, Raatikainen, M, Männistö, T, Mikkonen, T & Nurminen, J K. 2021, 'Systematic literature review of validation methods for AI systems', The Journal of Systems and Software, vol. 181, 111050.

Ramanathan, A., Pullum, L.L., Hussain, F., Chakrabarty, D. and Jha, S.K., 2016.. Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision. In 2016 Design, Automation & Test in Europe Conference & Exhibition (pp. 786-791). IEEE.

Russell, S. and Norvig, P., 2002. Artificial intelligence: a modern approach.

Salay, R., Queiroz, R. and Czarnecki, K., 2017. An analysis of ISO 26262: Using machine learning safely in automotive software. arXiv preprint arXiv:1709.02435.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.F. and Dennison, D., 2015. Hidden technical debt in machine learning systems. Advances in neural information processing systems, 28, pp.2503-2511.

Szegedy, C. et al. (2014) 'Intriguing properties of neural networks', arXiv:1312.6199 [Preprint]. Available at: http://arxiv.org/abs/1312.6199).

Wang, Y., Xiong, R., Yu, H., Zhang, J. and Liu, Y., 2018. Perception of demonstration for automatic programming of robotic assembly: framework, algorithm, and validation. IEEE/ASME Transactions on Mechatronics, 23(3), pp.1059-1070.

Xin, D., Miao, H., Parameswaran, A. and Polyzotis, N., 2021, June. Production Machine Learning Pipelines: Empirical Analysis and Optimization Opportunities. In Proceedings of the 2021 International Conference on Management of Data (pp. 2639-2652).

Yao, Q., Wang, M., Chen, Y., Dai, W., Yi-Qi, H., Yu-Feng, L., Wei-Wei, T., Qiang, Y., Yang, Y., 2018. Taking human out of learning applications: A survey on automated machine learning. https://arxiv.org/abs/1810.13306

Zhang, J.M., Harman, M., Ma, L. and Liu, Y., 2020. Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering.

Zhou, Z.H., 2016. Learnware: on the future of machine learning. Frontiers Comput. Sci.10(4), 589–590

Zöller, M., Huber, M., 2021. Benchmark and Survey of Automated Machine Learning Frameworks. Journal of Artificial Intelligence Research 70. pp. 409-472.