Industrial Machine Learning for Enterprises

Deliverable D3.2

First version of 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:	D3.2
Nature of deliverable:	Report
Dissemination level:	PU
Internal version number:	1.0
Contractual delivery date:	2022-05-31
Actual delivery date:	2022-06-13
Responsible partner:	University of Helsinki

Contributors

Editor(s)	Mikko Raatikainen (University of Helsinki)
Contributor(s)	Juhani Kivimäki (University of Helsinki), Ville Kukkonen (Granlund), Janis Lapins (Spicetech), Lalli Myllyaho (University of Helsinki), Mikko Raatikainen (University of Helsinki), Harry Souris (SiloAI), Csarnó Tamás (Vitarex), Dorian Knoblauch, Jürgen Großmann (Fraunhofer Fokus).
Quality assuror(s)	Donny Thomas Daniel (Siemens), Johan Himberg (Reaktor)

Version history

Version	Date	Description
1.0	22-06-13	Version for publication

Abstract

This document describes the initial version of the methods and techniques. On the one hand, the implementation of the methods and techniques will be detailed later, and, on the other hand, the methods and techniques will be refined and extended, possibly adding new methods and techniques. Moreover, the methods and techniques will be tested in the use cases of the IML4E project. The methods and techniques covered are model cards, ML monitoring, ML debugging, ML model confidence calibration, ML testing with VALICY and quality assurance.

Keywords

MLOps, model cards, monitoring, debugging, testing, confidence calibration, quality assurance.

Executive Summary

This document describes the initial version of the methods and techniques. The methods and techniques covered are model cards to capture information about deployed ML models in ML-based systems; monitoring for drifts in features and applying feature importance and explain ability for deciding when to retrain models; ML debugging based on error diagnosis approach to narrow down what causes the error; ML model confidence calibration for probabilistic confidence estimations; node co-activation monitoring in neural networks to detect errors; advances in ML testing with VALICY; and quality assurance. In this document, we shortly summarize the key principles of the methods and techniques. The methods and techniques are now largely at their initial stages and relatively isolated. The future work includes developing the methods and techniques to be more mature, and integrate them with IML4E MLOps pipeline. Moreover, these methods and techniques will be realized and also evaluated in the case studies of the IML4E project.

Table of contents

1 Introduction

1.1 Role of this Document

The purpose of this document is to describe the first version of methods and techniques for advanced model engineering of the IML4E project. This document describes the initial version of the methods and techniques. On the one hand, the implementation of the methods and techniques will be detailed later in deliverable "D3.3 First version of tools for advanced model engineering", and, on the other hand, the methods and techniques are refined and extended, possibly adding new methods and techniques, in "D3.4 Second version of methods and techniques for advanced model engineering". Moreover, the methods and techniques will be tested in the use cases of the IML4E project. The document focuses on ML model engineering and quality assurance parallel with the data engineering-focused deliverable "D2.2 First version of methods and techniques for data collection, processing, and valorisation".

1.2 Intended Audience

The intended audience of the present document is composed primarily of the IML4E consortium for the purpose of understanding methods and techniques and advancing ML model engineering. However, this document is public and can provide an overview of the advances in the IML4E projects. This document describes methods and 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
[D3.1]	Baseline methods and techniques for advanced model engineering

Table 1: Contractual documents.

2 Methods and techniques

2.1 Model cards as an approach to manage the deployment versions of ML models

Model card (Mitchell 2019) is an approach that aims to clarify ML models' intended use cases and avoid their misuse. Intended as documentation for a trained ML model, a model card contains information about the intended use case, benchmark evaluation in relevant conditions, or any other information that the model's creators consider necessary for the proper use. A model card, or at least a significant part of it, can be automatically extracted and generated. However, some parts might require an engineer to manually record the design decisions, justifications, or rationales. Therefore, the model cards can form an additional task for developers to tackle and consider in their daily routines. However, such information needs to be captured somewhere, and model cards have emerged as a promising approach.

Our approach is that a model card is largely automatically generated with its metadata and stored in a machine-readable format that can be used to generate different views, such as HTML for web-based usage. The model card contains sections for overview and model details, considerations for the use of the model (intended use cases, limitations and trade-offs, and ethical considerations), performance metrics and examples of the model prediction in the evaluation set, and training parameters. The data scientist, model engineer, software engineer, or anyone with required technical knowledge is requested to fill in the specific considerations of new model versions. At the same time, the other sections are automatically updated on every run of the pipeline. The approach is that information is captured as a part of natural workflows rather than an extra phase, thus respecting today's spirit of incremental development. In fact, this approach follows the "Everything-as-code" (EaC) paradigm so that information is captured in code assets. Although many software development and operation specialists have already adopted the practices of EaC, e.g., infrastructure as code, our approach extends EaC beyond technical scientists, engineers, and operators (for further details, see Stirbu (2022)). The concept of EaC is extendable and can capture, e.g., many non-functional or regulatory concerns requiring traceable information for safety assurance.

As a result, information about an ML model stored in a model card is conveniently accessible before the deployment of the ML model for verification purposes. In particular, a model card then capsulates the required information for regulatory processes, such as in the case of medical devices that are required to go through rigorous processes before being allowed to use. For any deployed ML model, a model card provides traceable information and justification for the deployed ML models.

2.2 Monitoring for drifts in features and applying feature importance and explain ability for deciding when to retrain models

For the purposes of monitoring machine learning products in production, enterprises are monitoring, among others, the input features that the production models are receiving. Enterprises are comparing the production input features with the features used for training the production model. The purpose of that comparison is to identify if the statistical distribution and characteristics of the production features vs. the training features are significantly different. The difference between the features of those datasets indicates that the model operating in production has been trained in an environment that is not close enough to the production. Thus, a retraining operation should start, and a new model version could be deployed. The process is summarized in Figure 1.

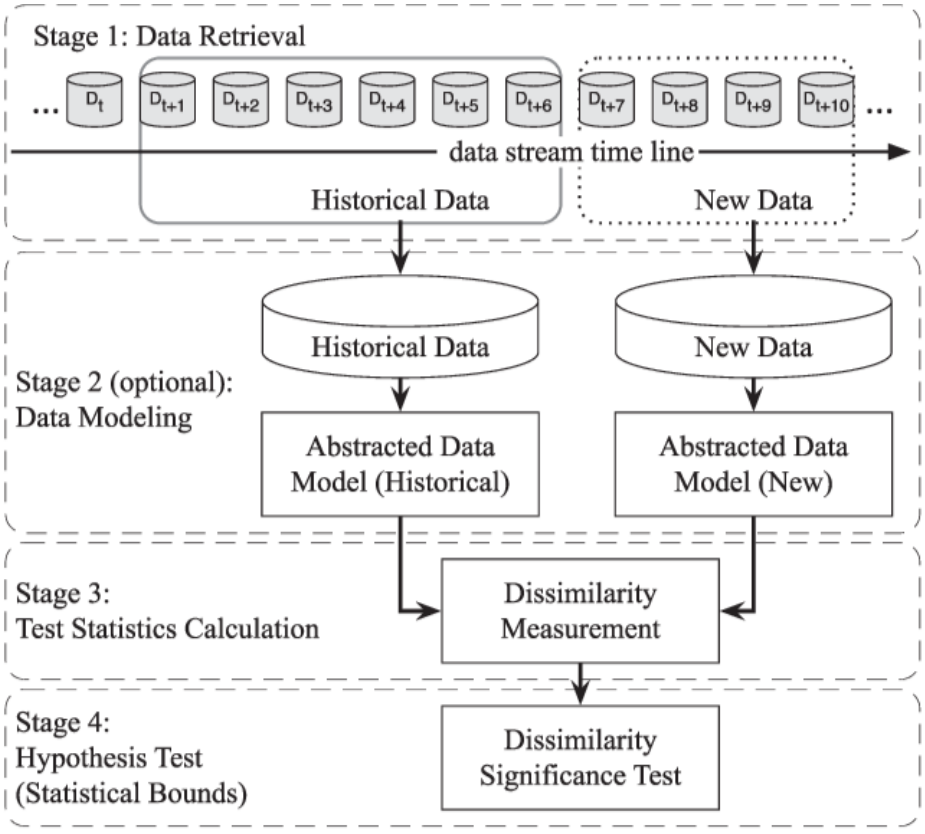

Figure 1: Metrics related with the statistical distance between the production (inference) input (new data in figure) and the data used in the training phase -historical data in figure (Lu et al. 2020).

To identify the distance between the historical and training data, and the production or new data, various techniques can be applied depending on the nature of the data. One area of interest when it comes to monitoring for drifts in the input features is that system controllers and operators are not in a position to decide and describe what is the impact on the decision ability of the models when particular features are drifting while other features are not. To address that, we propose the usage of explainability techniques and, in particular, the utilization of Shapley values that measure the contribution of each feature to the predictions of the models. Having that information, we are in a position to know how much the drifting features are affecting the decisions of the production models. Thus, we are in a better position to describe to the operators of the solution that the level that our solution is affected by the changes in the production environment.

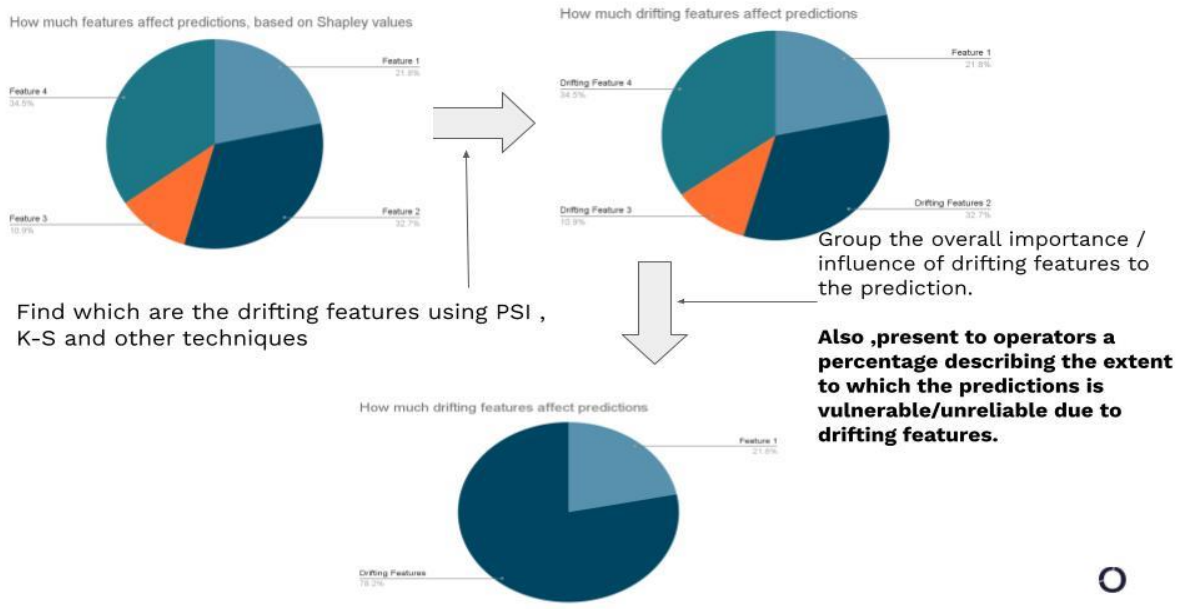

Figure 2: demonstrating how explainability and Shap values in particular can help on summarizing the influence of drifting features to models predictions

2.3 Monitoring for high number of ML-models & model cards

In applications with large quantities deployment instances for different customers, efficient monitoring is particularly important and challenging. That is, even the system software can be same but each system deployment is trained with customer-specific contextual data resulting in that each deployment instance having similar but somewhat unique ML model.

For this purpose, we will implement an automated process for storing model metadata alongside the serialized model and to build tooling that enables the contextualization and visualization of such metadata in model cards. Additionally, the tooling can be augmented with access to the feature store, the ability to test the models with given inputs, and visualizing historical features. Figure 3 illustrates the high-level overview of the data flow for the process and tooling for one customer – practically each customer are served with similar pipeline.

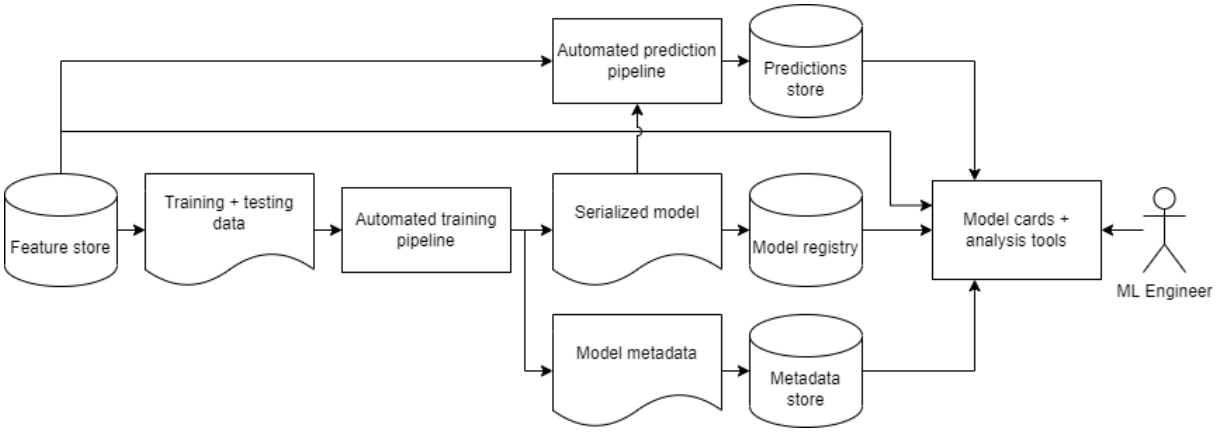

Figure 3: Overview of the data flow for model cards and model monitoring.

The metadata about the time series data to be stored and used in automatic or manual monitoring includes features such as:

- Summaries and statistics derived from the data used to train the model, such as minimum/maximum values and value bins or distribution descriptors.
- Different scores for the model.
- Used model hyperparameters.
- For time series applications, the start and end date of period used for training data.

For the metadata derived from input data, it is important to consider the difference between the features loaded from the feature store and the actual data that is used in the training process, as there may be filtering and different sampling methods applied.

To detect the need to retrain a model, some performance metrics are required. For this purpose, the predictions made by the models are connected to the monitoring tooling. This enables the analysis of historical predictions made by the models, and calculating error metrics when actual values for the predicted quantities are known.

2.4 Debugging

Unlike traditional software systems, poor quality in a machine learning (ML) model does not necessarily imply the presence of a bug. Instead, to debug poor performance in a model, one investigates a broader range of causes than would be the case in traditional programming. For example, here are a few causes for poor model performance (Google 2019):

- Features lack predictive power.
- Hyperparameters are set to nonoptimal values.
- Data contains errors and anomalies.
- Feature engineering code contains bugs.

To find out what causes the poor performance of the ML model, we use an error diagnosis approach. Our goal is to collect as much information about the model as possible, then narrow down what causes the error. The exact approach depends on the ML model type, but some of these techniques can be used for a wide variety of models. We can define main performance metrics which can be calculated during or after model training, such as accuracy, precision, and recall in a typical classifier. Logging the loss function during model training can be used to analyze train and test loss curves (bias vs. variance analysis). Some models will have commonly occurring error types, which can be identified by comparing model output on test data with ground truth.

We have been designing a process for debugging and error diagnosis in our use case of a human pose estimation problem. The main performance metric is called keypoint similarity (KS) (Ronchi 2017). For each keypoint on the human body, keypoint similarity measures how close the predicted keypoint is to the ground truth label. Closeness is defined by a Gauss function. Using KS, we can call a prediction correct if the KS value is above a certain threshold.

Typical pose estimation localization error types are the following: Jitter, Error, Swap, Inversion. Jitter refers to a small localization error around the correct keypoint location, while miss is a large error (above a certain threshold value). Inversion error means the ML model confuses different keypoints of the same person, while swap error refers to the model mixing up keypoints belonging to different people. We can identify different error types from KS values; therefore, we can calculate the frequency of each error type. With this information, we can also break down error types to each keypoint. We can also do sensitivity analysis on training data parameters, like how keypoint occlusion or image size, or other factors have an effect on model precision. All of this information is there to give insight to the ML engineers about what types of errors occur on the test dataset, and what might cause those errors.

During model training, each metric is logged using MLflow platform's tracking features. Automatically generated graphs can be saved as artifacts, as well as test image samples, where certain error types occurred. Automated report generation is also supported.

Some common error fixing techniques are:

- Change dataset / more data
- Treat missing values and outliers
- Hyperparameter tuning
- Longer training
- Change base model type
- Use of cross validation
- Use of data augmentation

2.5 Node co-activation monitoring in neural networks to detect errors

Machine learning (ML) models are statistical approximations, whimsical and capricious in nature, and often made for environments that evolve over time. In such approximations, a 99% accurate model -- something that is practically always correct -- is wrong 1% of the time. The trustworthiness of ML systems has been improved, for example, by establishing patterns for fault tolerance, but tools for measuring whether a model's results are and remain trustworthy can still be improved (Myllyaho 2022). Furthermore, such detection should ideally occur in real-time while the model is running. During computation runs, one approach to mitigating the risk and making the system more fault-tolerant could be monitoring the model's own inner structure.

The inner structure of a neural network -- a currently common ML technique -- is sometimes compared to the structure of biological neural circuitry (e.g., Abiodun 20018). Like biological neural circuitry and neurons, neural networks in computing consist of layers of interconnected nodes. These nodes are tiny computational units that receive an input and either activate and pass on output to the following nodes or remain dormant with an output of 0, having no effect on the computations made by the following nodes. We know from previous research on neural networks that after network training, specific groups of nodes tend to be responsible for specific outcomes and thus often activate concurrently (Pei 2017). For example, in an image recognition model, certain groups of nodes can be expected to activate when the image of a dog is shown, while at least a partially different group should activate for the image of a cat.

However, what if the activating nodes are suddenly ones that usually do not activate together and thus do not belong to any same group? Do these rare co-activations within a neural network indicate that the computation result is incorrect or that the input has never been seen before? If so, rare co-activations could be potentially be used to detect errors in neural networks.

To test our hypotheses, which are describe in more detailed in a publication under review, we trained four neural networks for an image classification problem. For every node in a neural network, we calculate how often each node activates concurrently with every other node using the training data: i.e., how probable it is that two nodes activate concurrently. This way, we obtain a metric of which nodes often contribute together to the network output and thus belong to one or more of the same groups. Using a separate test data set, we count the number of rare co-activations happening within the neural network for each input.

Rare co-activations are, on average, much more common in inputs from untrained classes than in inputs the model has been trained for. Thus, rare co-activations show good potential in detecting drift in incoming data: should the average number of rare co-activations increase, drift is most likely imminent. However, inputs from trained classes contained outliers with a high occurrence number of rare co-activations as well, and some untrained inputs have a low number of occurrences. Thus, detecting inputs that the model cannot handle and preventing them from being used further down in the system is more problematic. Considering the difference in the average number of occurrences, it may be possible to find systems and contexts where using it is feasible, but the system should be able to deal with some false positives and negatives. Additionally, rare co-activations tended to be more common in incorrectly predicted inputs than incorrectly predicted ones, but the difference was both smaller and statistically less significant. Thus, detecting single inaccurate predictions may not be feasible based on the number of occurrences alone, but the approach should at least be fine-tuned to find the most indicative co-activations.

2.6 Confidence calibration

The predictions of machine learning models always entail a degree of uncertainty. In the case of probabilistic classifiers, the predictions are given as vectors, which can be regarded as probability distributions over a discrete random variable, where each element of the output vector corresponds to a certain class. The predicted class will be the one with the highest value assigned by the model. This highest value is often called a confidence score. It would be tempting to make the interpretation that a confidence score expresses the probability of the predicted class being the correct one. However, this is not usually the case. Empirical analysis shows that the probabilistic predictions of machine learning models are often either underconfident or overconfident in this regard (Niculescu-Mizil 2005; Guo 2017). In contrast, a model whose confidence scores can reliably be interpreted as probabilities is called calibrated.

A calibrated classifier has many benefits over an uncalibrated one. If confidence scores can be interpreted as probabilities, then rules of probability calculus can be applied reliably. This is a useful property in many scenarios, such as when building complex AI systems, which need to take the predictions of several independent models into account in downstream tasks. Also, in any decision process where different errors produce different costs and risks need to be balanced, having calibrated confidence scores may prove to be useful.

Although research on confidence calibration has been published since 1950s, the research was originally focused on assessing the quality of probabilistic forecasts and automated decision making. Recently, there has been increasing interest in utilizing calibrated confidence scores in other downstream tasks as well, especially in the field of deep neural networks. This has given rise to applications leveraging confidence scores, such as Out-of-Distribution detection (Hendrycks 2017), selective classification (Geifman 2017) and failure prediction (Corbière 2019), which can all be regarded as forms of model monitoring.

Sometimes there is no straightforward or well-established way of producing probabilistic confidence estimations for the predictions of a given machine learning model. Such is the case with predictions made in a use case of the IML4E project, in which information is extracted from machine-readable PDF. Currently, the automated decision on whether predictions of the AI system can be trusted is based on a complex and manually tuned set of heuristics. Our approach is to design and implement a solution on how reliable confidence scores can be attached to the predictions of the existing AI system. Ideally, the confidence scores could then be used to replace the existing automated decision-making process and in model performance monitoring. Figure 4 gives an overview of the current version of the proposed solution.

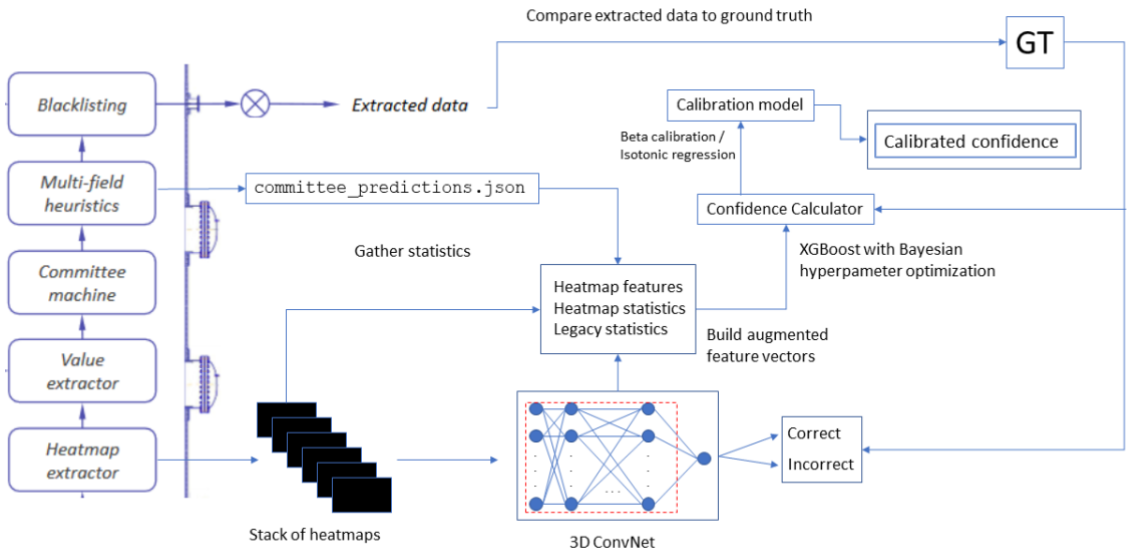

Figure 4: Overview of the proposed confidence calculator model.

2.7 Black box testing with VALICY

VALICY is an existing black box testing tool that is being further studied and developed within the IML4E project. The following describes the key advances so far.

Workflow. The workflow is being revised, and a re-structuring took place. Now the VALICY controller starts jobs and creates instances (AI model). While the earlier applied principles of levels still holds (job, instance, run), a new type of level is introduced for the nature of the instances: the level of evolution. This regards the nature of different instances created and their level of evolution. With each level comes an increase of smartness:

- The first level only regards the regular grid, low smartness
- The second level level has both an ai model and random selection of parameters with a configurable ratio of ai / random
- The third level, with increasing smartness and increasing number of evaluated parameter proposals, the information through the results about the multi-dimensional space becomes sufficient to only run artificial intelligence

In addition, the way to replace instances was completely redone. Instead of randomly generating the next AI model, an evolution of instances is initialized by an ML class such as an XGBoost classifier with a random (of a pre-selected) hyperparameter configuration. It has a certain scoring. After reaching a pre-set number of runs, this configuration is stopped and replaced by another child instance of the same type (XGB). To mimic evolution, a set of these hyperparameters is inherited by the child (somewhat an analogy to DNA), and the performance of this child is monitored. When the resulting prediction accuracy is worse than the one of its parent, the parent's configuration is varied again to form another child's configuration in combination. Once a child has two parents, only 2 out of 3 are taking for the inheritance of the configuration for the next child. This is depicted in Figure 5. By this approach, an instance evolution ensures that descendants are always more performant than successors or are replaced in the quest for survival of the fittest. The scoring of the instances is the accuracy of predictions. The uncertainty in a first approach is modelled to be *uncertainty=1-accuracy*. However, this does not regard the whole space since only the accuracies of predictions present can add to the uncertainty and is therefore only a temporal value. The overall uncertainty will be regarded in the next development step.

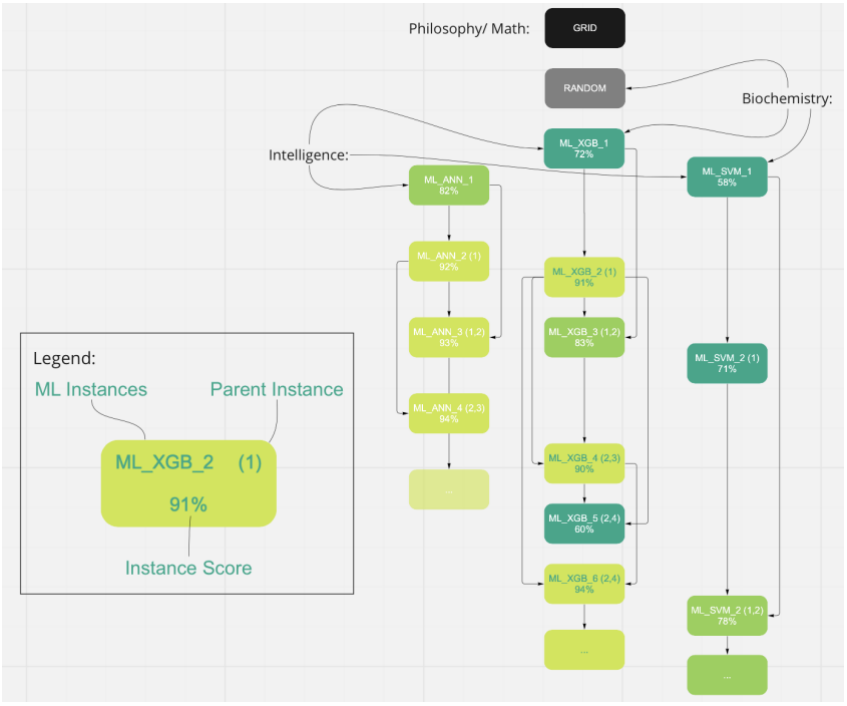

Figure 5: Levels of Evolution for instances and inheritance of hyperparameter configurations

As far as the workflow is concerned, the VALICY workflow, previously started on a server via Docker swarm, is also changed completely. While formerly, the VALICY controller started a docker swarm that started and replaced the instances depending on the scoring. It now just creates instance configurations and writes them into the VALICY database task queue. In the task queue, docker containers that run *instance_controllers* will start to process all proposed VALICY instance configurations without further intercomparison between instances. This approach, in combination with a complete re-doing and optimization of the code (aggregated database reads and writes), not only leads to a far more efficient code with regards to performance (number of proposals generated) but also eases up the debugging process since development and debugging in a docker swarm is rather exhaustive. The VALICY controller can now be debugged and run on a local computer since all the work except the evolution is done by the *instance_controllers*. Also, the *instance_controllers* can be debugged in a far more convenient manner.

Frontend. The web frontend has received some significant UX improvements, like the representation of discrete dimensions. Moreover, users can now store specific visualization configurations per job and reload them again. To support live surveillance of running instances, it is possible to update the result set represented in the current plot automatically by a fixed time interval.

The next steps include:

- Separate representation for the regular grid – can be turned on and off separately.
- Updating the color palette in the diagrams used to represent the different states, so that a more contrasting representation is shown.
- More fluently updating the available data, so there is less need for users to update manually.

API and Python integration: The REST-API used to communicate through the database with the VALICY backend has been further extended and optimized. The documentation of the current version is provided through a swagger platform under https://api.valicy.de/docs. In this course, the database accesses have been reconfigured and improved, so minimum necessary fields are queried, and the virtual table setup has better performance.

A separate Python module has been created, focussing on being easily deployable by data scientists. Additional endpoints have been added to the API so that it can best be used through this interface. It has already been applied in test environments and is further developed as needed. Our experts are looking forward to providing the module and assisting with the setup.

2.8 Continuous audit-based certification

ML-Systems are used for various use-cases in various industries, including highly regulated industries that require systematic evidence for achieving required quality goals or even certification. Since MLOps exists to deploy models at a high pace with high quality on automated bases, traditional point-in-time certification with a manual audit approach cannot keep up with this pace. Because retrained models are technically leading to a different Product, that require a recertification. So frequent changes to the deployed ML-System include significantly more certification-related manual overhead. Continuous audit-based certification (CABC) mitigates this by performing an automated audit, which leads to an automatically issued or revoked certificate.

The core of CABC is the automated compliance assessment of a product or system according to a set of given requirements based numerous quality attributes. Quality attributes describe an aspect of a system where a requirement defines a state in which an attribute should be much like an object of a class. For the implementation, we need to select suitable measurements and expected results for the selected quality attributes.

The difficulty is to find or produce the input for those quality measurements. Here comes in handy that MLOps is designed to maintain a high level of quality by explicitly or implicitly using tools that target automation in quality assessment and control. This means we can utilize their output or introduce other required tools to the MLops process.

Initially this demands a manual mapping between measurements tools and model quality measurements. As there are various frameworks, tools, and measurements suitable depending on the particular type of ML-Model

the right one need to be selected and adjusted. This leads to a vast variety of combinations and therefore manual work in selecting and adjusting.

To ease this process, we introduce a mapping methodology that makes it easy to implement measurements from scratch or use existing tools for providing information on quality. For having this standardized way of retrieving the data for the measurement we introduce an audit API, which functions as a clear separation between the test subject that is being audited and the party that performs the audit. The data transmitted mirrors the evidence in traditional certification. The measurement results will be evaluated off-premise to reach an independent assessment of the compliance with the quality goals. Depending on if the goals are met a certificate gets issued.

In the following we layout the methodology and the information retrieval process.

2.8.1 Methodology

In comparison to a traditional point-in-time certification, CABC needs adjustments to the roles and processes. Establishing trust in a traditional certification is achieved by introducing trustworthy third parties, usually organizations and humans. The challenge with CABC is to achieve the same grade of trustworthiness by mainly automated technical means.

A certification is usually used to demonstrate compliance to customers or authorities. A self-proclamation from an auditee does not generate the same amount of trust due to a conflict of interest as a third-party audit. For this reason, certification schemes often demand two and even three parties. Hence, we foresee for our CABC for MLOps the following parties:

- A Certification body defines the rules for the certification process. It lays out the criteria under which an audit is conducted and defines the form of the audit report. It is the certification body that according to the audit report issues or suspends a certification. Specific to CABC the certification body provides a registry of the ongoing certification process which serves potential stakeholders as a trusted resource for the defined scope as well as the current certification status.
- An auditing party conducts the audit under the rules of the certification body. It will verify the scope provided by the auditee for its suitability and its adherence to given requirements. It will verify the initial setup of the continuous auditing and facilitate the automated measurements and assessments at operation. The auditing party provides the means to receive the evidence from the auditee.
- The auditee is the owner of the ML-System. For establishing CABC the auditee needs to define the scope which includes selecting the required attributes and the frequency in which they get assessed. The auditee also needs to implement the technicalities in the MLOps process which means selecting the proper measurements for the evaluation of the quality attributes and making the actual technical implantation part of the MLOps process.

From a high-level view, CABC for MLOps can be separated into a “preparation phase” and the “operation phase”. The preparation phase is the initial manual setup and the operation phase is automated execution, see Fig. 6. It shows the three roles and their main activities during the two phases.

In the initialization phase, the proper operationalization of the selected set of quality requirements takes place. Key actions in this phase are the definition of the scope, the identification of the measurements associated with each quality attribute, the determination of the frequencies at which each quality goal should be checked, as well as the implementation of the mapping of evidence and quality measurement input. Mapping is leveraging the raw data produced in the MLOps process to usable measurement input via parsing, transforming or executing. In the continuous phase, the previous implanted measurements are performed in a continuous manner. Results of the measurements are interpreted and as part of an assessment lead to a certificate.

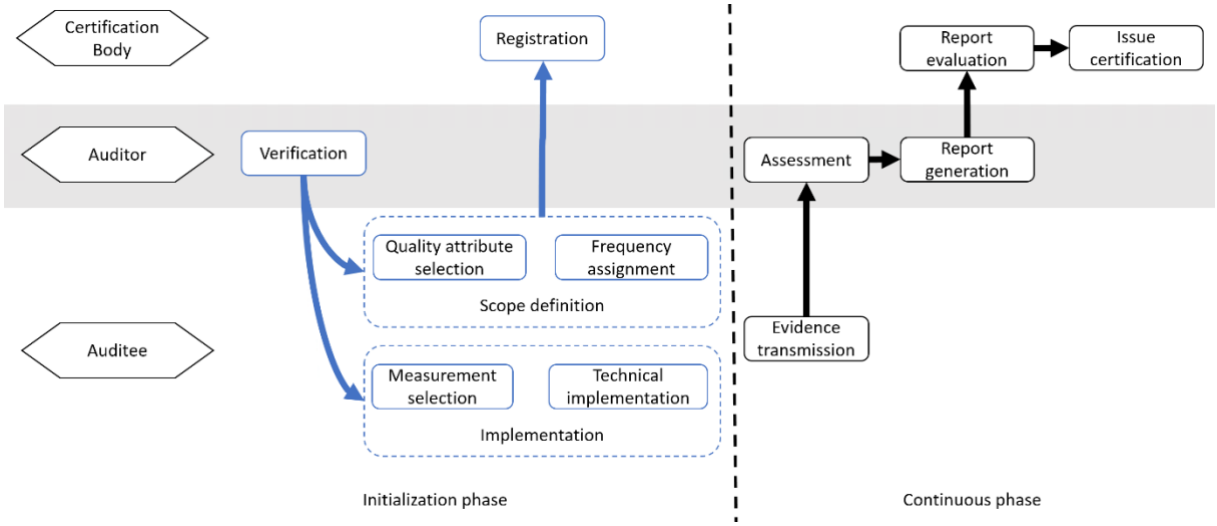

Figure 6: Roles and processes of CABC durring the initialisation phase and the continuous phase

2.8.2 Information retrieval

During each step of the MLOps process artifacts like logs, configurations, checkpoints, metadata, models, etc. get created.

An essential part of our concept is to extract the input parameters for the measurement from those artifacts. This can be done by several techniques like parsing a log or reading configurations and mapping certain statements to parameters. Such passive aggregation of information does not accommodate for the whole set of required parameters. Active counting, querying, testing and monitoring is necessary for obtaining information for instance data quality checks or robustness checks on the trained model. This process of passive and active information retrieval needs to be implemented specifically to the software stack in use. First experimental results are realized in MLflow in the form of components and plugins for the information retrieval. But for our approach to be more technology agnostic the Audit API layer is introduced. This REST-Endpoint receives data on the parameters needed for the metrics which is the value for the parameter as well as the raw data or if it is too large hashes to store as evidence. The received values enable the measurements which in their entirety then allow for an assessment of first one quality attribute and in combination of the quality of the whole ML-System. The result of this assessment is an automated audit report suitable for a certificate from an accepting certification body.

During the various steps of the MLOps process, different kinds of artifacts are generated. Those artifacts contain either directly valuable information for the quality assessment or they can be used as tools to generate this information. The mapping part of our framework facilitates the transformation of the raw data into the values and pieces of evidence transmitted to the Audit API, see Fig. 7.

It shows on top the MLOps process where each iteration produces raw-data. This raw-data is input either to active testing or directly to the extractor. The extracted information is mapped to the API.

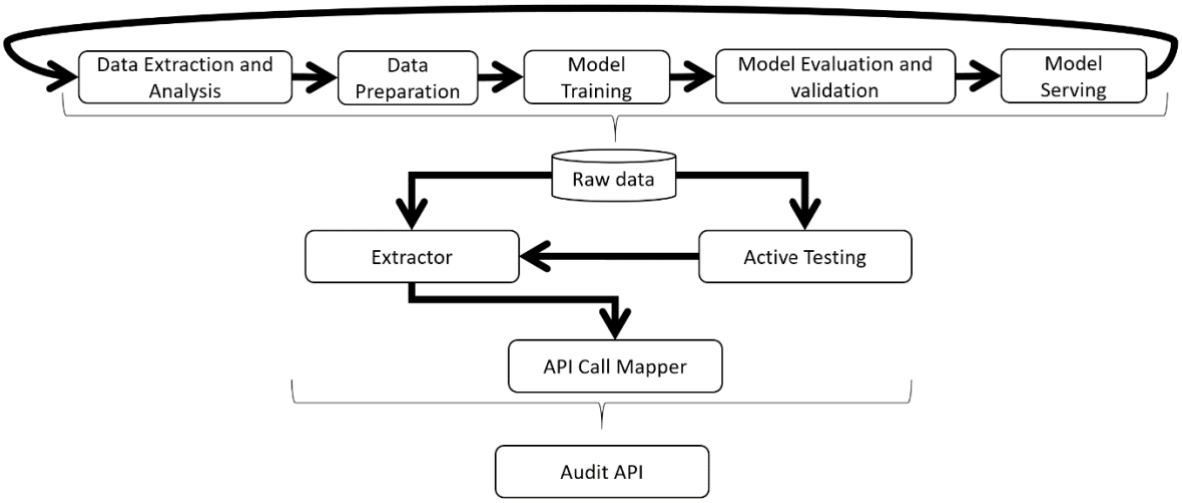

Figure 7: Mapping of the raw data to the input of the API calls.

3 Conclusions and outlook

This document has described the first, initial versions of methods and techniques for advanced ML model engineering and quality assurance of the IML4E project. The methods and techniques covered are model cards, ML monitoring, ML debugging, ML testing with VALICY, ML model confidence calibration, and continuous audit-based certification. The methods and techniques are now largely at their initial stages and relatively isolated. The future work includes developing them to be more mature, and better integrate with IML4E MLOps pipeline. Moreover, these methods and techniques will be realized and also evaluated in the case studies of the IML4E project.

References

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. Heliyon, 4(11), e00938.

Corbière, C., Thome, N., Bar-Hen, A., Cord, M., and Pérez, P. (2019). Addressing failure prediction by learning model confidence. In Advances in Neural Information Processing Systems.

Geifman, Y. and El-Yaniv, R. (2017). Selective classification for deep neural networks. In Advances in Neural Information Processing Systems.

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, (2019), Testing and Debugging in Machine learning, https://developers.google.com/machine-learning/testing-debugging/common/overview

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, page 1321–1330. JMLR.org.

Hendrycks, D. and Gimpel, K. (2017). A baseline for detecting misclassified and out-of-distribution examples in neural networks. In International Conference on Learning Representations,

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I.D. and Gebru, T., (2019). Model cards for model reporting. In Proceedings of the conference on fairness, accountability, and transparency (pp. 220-229).

Myllyaho, L. S., 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, 183, [111096].

Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. In Proceedings of the 22nd International Conference on Machine Learning, ICML '05, page 625–632, New York, NY, USA. Association for Computing Machinery.

Pei, K., Cao, Y., Yang, J., & Jana, S. (2017). Deepxplore: Automated whitebox testing of deep learning systems. In proceedings of the 26th Symposium on Operating Systems Principles (pp. 1-18).

Ruggero Ronchi, M., & Perona, P. (2017). Benchmarking and error diagnosis in multi-instance pose estimation. In *Proceedings of the IEEE international conference on computer vision* (pp. 369-378).

Stirbu, V., Raatikainen, M., Röntynen, J., Sokolov, V., Lehtonen, T., & Mikkonen, T. (2022). Towards multi-concern software development with Everything-as-Code. IEEE Software, (E-pub ahead of print). https://doi.org/10.1109/MS.2022.3167481