IoTwins
BIG DATA OPTIMIZE INDUSTRY AND SERVICES

Grant Agreement N°857191

Distributed Digital Twins for industrial SMEs: a big-data platform

DELIVERABLE 3.5 – EDGE-BASED EVENT GENERATION SOFTWARE MODULE

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement № 857191

Document Identification

Project	IoTwins
Project Full Title	**Distributed Digital Twins for industrial SMEs: a big-data platform**
Project Number	857191
Starting Date	September 1st, 2019
Duration	3 years
H2020 Programme	H2020-EU.2.1.1. - INDUSTRIAL LEADERSHIP - Leadership in enabling and industrial technologies - Information and Communication Technologies (ICT)
Topic	ICT-11-2018-2019 - HPC and Big Data enabled Large-scale Test-beds and Applications
Call for proposal	H2020-ICT-2018-3
Type of Action	IA-Innovation Action
Website	iotwins.eu
Work Package	WP3 - AI services for distributed digital twins
WP Leader	UNIBO
Responsible Partner	UNIBO
Contributing Partner(s)	SAG, SAGOE
Author(s)	Andrea Borghesi (UNIBO)
Contributor(s)	Schuele Tobias (SAG), Kintzler Florian (SAGOE)
Reviewer(s)	Baptiste Morisse (THALES), Filippo Mantovani (BSC)
File Name	D 3.5 – EDGE-BASED EVENT GENERATION SOFTWARE MODULE
Contractual delivery date	M30 – 28 February 2022
Actual delivery date	M32 – 15 April 2022
Version	1.1
Status	Final
Type	DEM: Demonstrator, pilot, prototype
Dissemination level	PU: Public
Contact details of the coordinator	Francesco Millo, francesco.millo@bonfiglioli.com

Document log

Version	Date	Description of change
V0	20/01/2022	First draft
V1.0	22/03/2022	Internal review completed
V1.0	06/04/2022	Document sent to the consortium for approval
V1.1	15/04/2022	Final Version

Table of Contents

1 Introduction

In this document we describe a software module developed to generate events on the edge devices. To be more precise, in this case with the term "event" we may refer to a variety of different occurrences, ranging from status updates on critical components of an industrial pipeline, to alarm signals revealing faults or anomalous situations, or simply messages reporting non-critical information which can be nonetheless useful for helping system administrator, machine operators and facility managers. We assume that an event is described by a timestamp (describing *when* the event happened) and an associated value *(what* has happened).

In this document we describe the architecture, composed of different sub-services, which can be adapted for the purpose of event generation. In particular, the architecture has been devised to be into components which require cloud capability and those which instead are to be run on the edge. In the following sections we first describe the architecture of the event generator module and provide more details about its implementation in two exemplary testbeds. Then, we consider the case of performing data analytics through Machine Learning (ML) models on the edge.

2 Software module description

We start by providing a general overview of the software module architecture, as shown in Figure 2.1. The architecture is composed by three main components (not all on the edge): 1) the IoT device (identifiable as the "IoT Device" rectangle in Figure 2.1), from which the data is monitored and collected, 2) the cloud computing resources (the "Cloud") box in Figure 2.1), where a historical data set is kept and where the deep learning (DL) models are trained, and 3) the edge device where the actual events are generated (the "Edge" box in Figure 2.1).

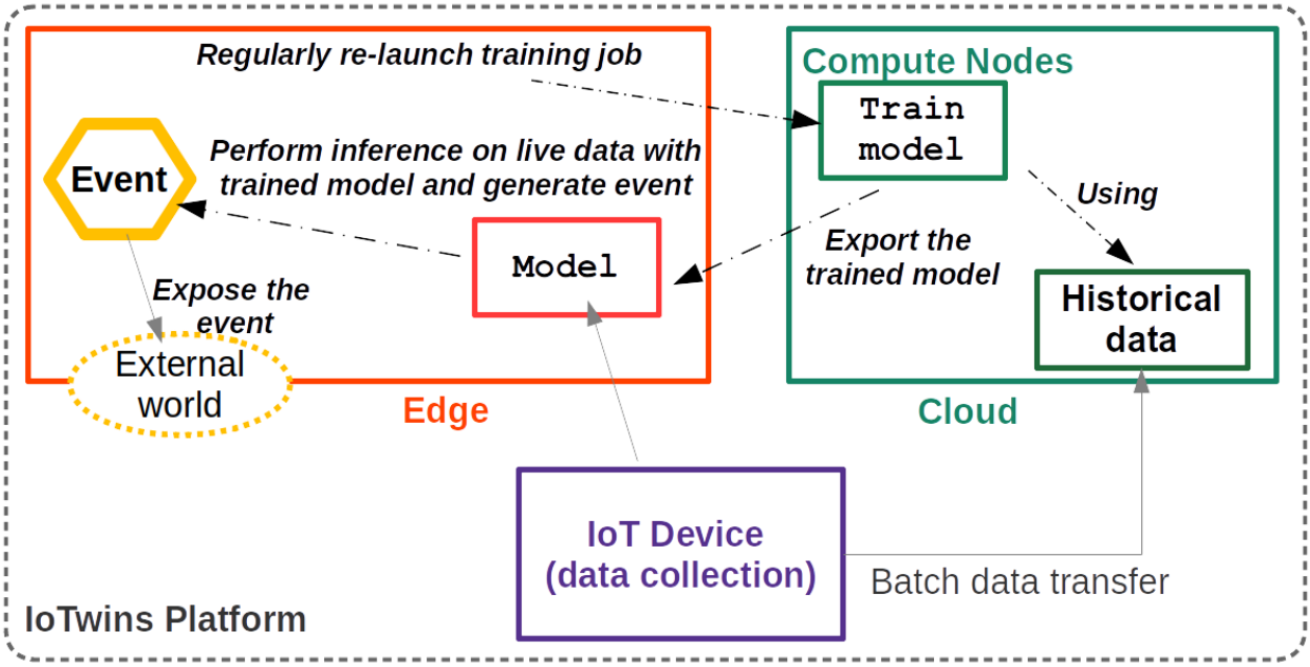

Figure 2.1 – General Edge-based Event Generation Architecture

The IoT device is often directly connected to the industrial components to be monitored and typically has a very limited computing capability. Its task mostly consists of monitoring the target component state and

exposing the relative data, for instance by sending this data to other, more sophisticated devices more apt at analyzing the data stream. In addition to data transferring through network protocols, the IoT device might perform simple pre-processing operations on the live data, such as batch creation (gather groups of data points before sending them to external world as data packets) or window-based aggregation (e.g., computation of mean and standard deviation over fixed-length time windows). The data from the IoT device can be sent either in batches, with an inherent temporal delay due to the creation of the batch after a sufficient number of data points are available (for example, when transferred to the cloud resources), or as a data stream which can be fed live to the edge components; in general, the data stream is not constituted by a continuous stream of data points but rather by smaller (but still discrete) packets of data (with time windows smaller than the batch transfer to the cloud).

The cloud layer is where the data coming from the monitoring devices can be stored for historical tracing and analysis. Historical data sets can reach large dimensions (easily GBs and TBs) and represent a huge wealth of information waiting to be mined. Thanks to this big data describing in detail the target industrial systems it is possible to create accurate models for the system behavior, thus obtaining the desired digital twins. For example, the large data sets collected by the monitoring infrastructure deployed on the IoT devices can be used to trained Deep Learning (DL) models for forecasting and estimating various targets (e.g., predicting the energy consumption of IT components in a supercomputer or the flow of people passing through turnstiles in a stadium hosting a public event, or again forecasting the status of the components in an industrial plant). Taking advantage of such big data requires significant computational resources and this is the reason why it takes place on cloud infrastructure or High-Performance Computing facilities. The type of model to be created strongly depends on the particular testbed; however, notwithstanding the context, these models (Deep Learning, simulation, optimization, etc.) are trained/created/refined on the cloud, possibly (likely) using large historical data. After the training, such models are firstly validated directly on the cloud and then can be exported to the edge – to be used on new data without requiring additional training. The training or model creation phase can be done either on a regular basis or in an asynchronous manner, depending on the requirements.

The edge component is where the events are actually generated. Typically, this is done by loading in the RAM memory of the edge an already trained model, which is then used to perform *inference* on live data coming from the IoT devices (possibly stored in a relatively small-sized storage space local to the edge component). By executing the cloud-trained model on the edge the events are produced; the inference mechanism must be continuously run on the live data as to create a stream of events. For this to be done the software module devoted to this task must be a *long-term running* module which must be kept alive at all time and relaunched when needed – from the development point of view, the event generator must be a software *daemon* running on the edge; this can be obtained in several different manners, according to the edge hardware characteristics and operating system (when/if available) and on the chosen programming language to create the software module. A crucial aspect for the employment of the trained model on the edge is the actual computational resources available in the edge. This is a very hard-to-generalize point, as there exists a wide range of devices which fall in the definition of “edge”, often depending on the specific characteristics of the considered testbed. It is thus impossible to provide generic yet useful guidelines, except the consideration that, broadly speaking, the edge devices do not offer sufficient computational capabilities to execute too complex models. For instance, the edge might lack GPUs or might have a limited RAM size, thus greatly limiting the size of the models which can be run on it. A mechanism to mitigate this issue when dealing with Machine Learning models is discussed in Section 3 of this document.

2.1 Current testbed deployments

In this section we provide more details on two event generators which have actually been deployed with the collaboration of the IoTwins partner. These examples have an illustrative purpose as well, to better understand the general architecture previously described. We will consider TB04 and TB06. Other testbeds have adopted similar solutions, either fully or partially.

2.1.1 TB04 - GCL Predictive maintenance and production optimization for closure manufacturing (beverage and spirits caps)

In the TB04 the main task is to forecast the Remaining Useful Life (RUL) of an industrial component. In this case the event generated at the edge is the RUL estimate. A similar approach is being studied for deployment in TB09 as well. The TB04 involves the GCL industrial device from an industrial plant for a selected injection moulding machine. Among the various hypothetical failure events examined, it was decided to move towards a dedicated event of breakage which cyclically, but unpredictably, occurs on a defined plastic injection moulding machine. The anomaly that has been decided to prevent is an abnormal wear of the spindle bearing coating (worm screw that allows the mobile table to close by acting on the brace of the toggle). In this case, we want to be able to predict the RUL of the overall component, that is the time left before another anomaly (of the type above described) arises. The RUL prediction must be available on the edge to allow operators to perform corrective actions; hence, in this case the edge-based generated event is the RUL estimation of the industrial component.

For this purpose, a DL model has been trained using High Performance Computing (HPC) resources provided by one of the partners of the consortium (CINECA), in particular using a HPC system called Marconi100[1]. Once the DL model has been trained, the next step is to deploy it and use it to perform *inference on the live data*, thus creating the desired RUL signal. The general architecture described in Figure 2.1 is specialized according to the characteristics of the testbed; the resulting architecture is portrayed in Figure 2.2. The data is generated from the industrial device (the injection moulding machines) and it is regularly sent to the HPC nodes of CINECA supercomputers to create an historical data set; the data has the format of multi-variate time-series. In this case we do not employ cloud resources but rather an HPC system. As edge, there is a Virtual Machine with limited computational capability hosted on HW resources again provided by CINECA; the VM has a mount point to the distributed file system employed in Marconi100 thus the HPC and the edge components shared a distributed data resource (DRES in the picture).

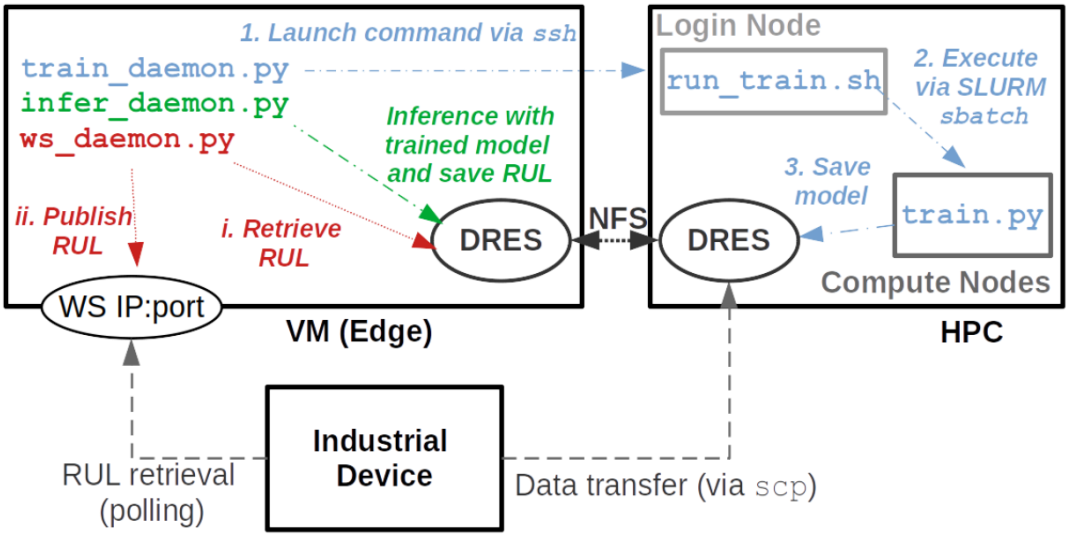

Figure 2.2 – TB04 Edge-based Event Generation Architecture

[1] https://www.hpc.cineca.it/hardware/marconi100

The training phase takes place on HPC resources. We start by pre-processing the collected data to remove clearly useless values (e.g., constant values for all the monitored period) and then we prepare the data for the actual task of RUL estimation; the key aspect is building the RUL label. We adopt the classical method of segmenting the time-series in runs-to-failure, i.e., periods of time where a component starts as perfectly healthy and goes towards the end of its useful life (where RUL is 0). This can be done as we have information describing the state of the component at every time step. We then trained a DL model, namely a 1-dimensional convolutional neural network trained in a supervised fashion: it relies on the presence of a label associated with the data (time-series) collected from the target system. To effectively train the ML model, the training data (the data used to teach the DL models) must contain some run-to-failure events, that is periods of time where a failure/a problem/an anomaly happened; there must be at least one run-to-failure, but a larger amount is welcome and could greatly improve the accuracy of the service. We implemented several types of neural network for handling the time series and predicting the RUL (after the data have been pre-processed): Recurrent Neural Networks (RNNs), 1-dimensional convolutional networks (CNNs), and Temporal Convolutional Networks (TCNs). After an empirical preliminary evaluation we opted for the 1-dimensional CNN as it provided the best trade-off between prediction accuracy and model inference time. The training of the DL model takes place on the computing nodes of Marconi100, by submitting the “train.py” job using the SLURM workload manager; the training job is started by launching the “run_train.sh” on the login node of Marconi. Once the RUL estimation model is trained it is saved in the DRES shared by the VM and the HPC system.

The edge is a virtual machine with Ubuntu 20.04 installed. On the edge component, there is a set of Python daemons which run at different frequencies. Every 2 months the “train_daemon.py” script launches a training phase on the HPC system, using the new data collected in the two-months time-span. If an already trained model is available, it does not get discarded but its weights get updated according to the new data; a new training process is performed anyway. Then, there is the “infer_daemon.py” script which runs every 5 minutes: this is the crucial software component devoted to the event generation on the edge. This script loads the trained DL model and performs *inference* on the live data collected from the IoT device. The inference operation generates the Remaining Useful Life prediction based on the data collected in the previous 5 minutes. The 5-minutes has been identified in collaboration with domain experts, according to the expected dynamics of the RUL evolution in the target industrial component. The output of the inference (the RUL prediction) is saved in a text file on the edge component. Finally, the “ws_daemon.py” is a Python script that is activated every 5 minutes and reads the RUL prediction and then publishes it through a web service. The web service IP address and port are reachable by enabled IPs; in this way the RUL estimation is exposed on the edge and can be accessed by the machines operators. Details of the RUL estimation and validation results will be provided in D4.4. The 5 minutes event-generation interval (the RUL prediction) has been identified via discussion with the domain experts. In practice, it is a very high rate more than sufficient to provide timely warning to the machine operators, as the typical incubation period for a failure is in the order of several (tens) of hours. Hence, we opted for a very fine-grained event-generation rate to be very conservative; at the same time, a 5-minutes inference interval is well supported by the edge infrastructure and does not result in excessive computational overhead.

2.1.2 TB06 - Large IT infrastructure management (EXAMON)

In the case of TB06, we are dealing with a large supercomputing infrastructure. In this case the definition of edge cannot really be applied, at least not in the general sense of computing device with limited capabilities: even the single computing nodes composing the supercomputer are made of high-end HW components (e.g., in terms of number of cores, RAM, HW accelerators, etc.). However, it is possible to consider the single computing nodes as the edge parts of the overall system, while the entire data center obviously corresponds

to the cloud part. In this case, the services/computations to be run on the edge must be as lightweight as possible, not to interfere with the typical workload of the computing nodes (which are devoted to executing the paying users' applications).

In the DAVIDE supercomputer[2] the edge component was available in the form of embedded boards installed in each supercomputing node, on top of the other computational components. The embedded boards served to provide finer-grained monitoring information than those obtained with off-the-shelf components. In that circumstance, we adopted a similar scheme for generating an anomaly signal describing the health status of the nodes. We trained a set of node-specific DL models trained in a semi-supervised fashion exploiting the supercomputing resources – after training, the DL models were deployed in the embedded boards with limited computational capabilities and used to analyze live data and produce the anomaly signal. The good performance of the results in terms of accuracy and low overhead revealed the feasibility of this approach in the supercomputing context[3]. In this case the edge-generated event is an alarm signal that reports the status of the supercomputing node, using a deep neural network trained on historical data and fed with historical data collected by the EXAMON monitoring infrastructure. In Figure 2.3 we can see the architecture of the edge-event generator.

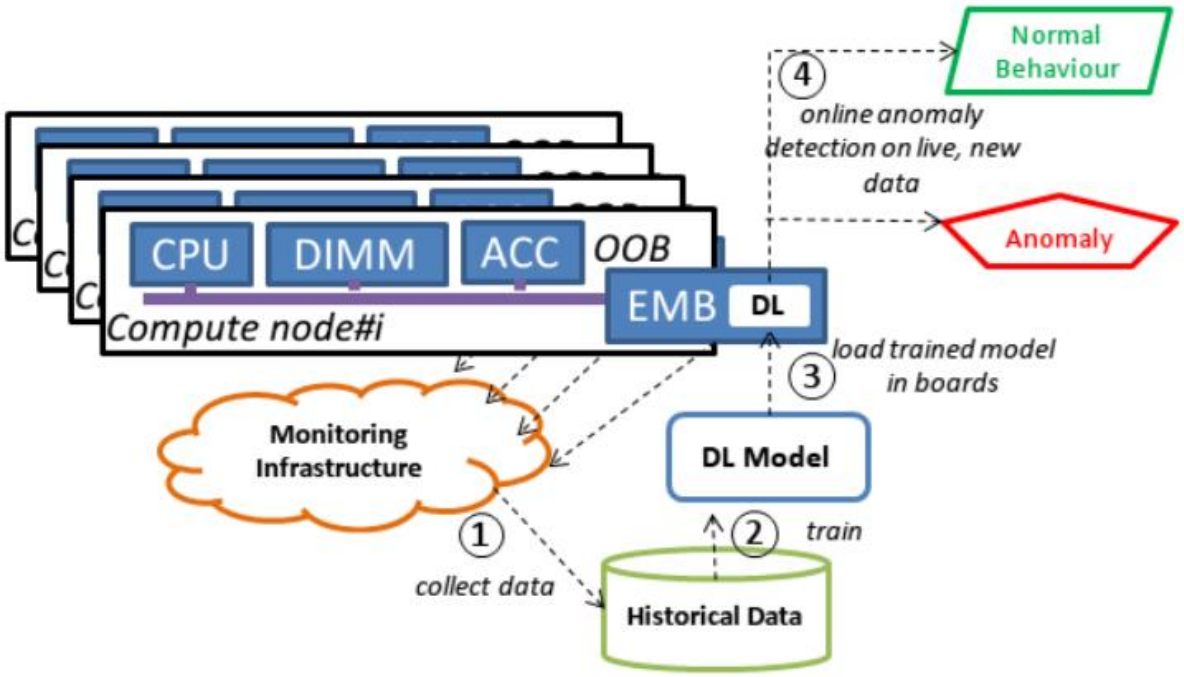

Figure 2.3 – TB06 Edge-based Event Generation Architecture

Data collected with the monitoring framework is fed to a DL model in order to train it to detect anomalies. During the training phase the model encounters only examples describing a system under normal conditions. After the training phase, the DL models are loaded on the embedded monitoring boards (EMB in the figure); when new measurements arrive, the trained DL model takes them as input and can identify anomalies, triggering an alarm for system admins. Since the embedded boards do not offer great computing capability, the DL model must be lightweight and generate low overhead. The benefits of edge computing (placing the anomaly detection module in the embedded boards) is twofold: 1) the board can directly read the out-of-

[2] Ahmad WA, Bartolini A, Beneventi F, Benini L, Borghesi A, Cicala M, Forestieri P, Gianfreda C, Gregori D, Libri A, Spiga F. Design of an energy aware petaflops class high performance cluster based on power architecture. In2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) 2017 May 29 (pp. 964-973). IEEE.

[3] Borghesi A, Libri A, Benini L, Bartolini A. Online anomaly detection in HPC systems. In2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS) 2019 Mar 18 (pp. 229-233). IEEE.

band sensors measurements, reducing the access time (furthermore, in-band monitoring is not allowed on many systems since it could affect stability); 2) online inference could not be easily performed on the computing nodes of a HPC system since it would subtract resources from users and complicate the scheduling process.

The DL model is an autoencoder neural network trained in a semi-supervised fashion, I.e., using only examples belonging to the normal state of the computing node during the training phase. The main idea of our method relies on the autoencoder capability to learn the typical (normal) correlation between the measures and then consequently identify changes in this correlation that indicate an abnormal situation. We train the autoencoder with data corresponding to the normal state and minimize the reconstruction error; this error is called training error. In this way it learns the normal correlations between the features from the monitoring infrastructure. After this first training phase, we feed the autoencoder with new data unseen before – this is generally called inference in DL terminology – and we then observe the reconstruction error. If the new data is similar to the data used as input (if it respects the normal correlations) then the error will be small and comparable to the training error. If the new data correspond to an anomalous situation, the autoencoder will struggle to perform the reconstruction, since the learned correlations do not hold. Hence, we identify anomalies by observing large reconstruction errors during the inference phase. We train an autoencoder for each supercomputing node, using the HPC resources to train them. After the training, the neural networks are loaded in the embedded devices (each model is loaded in the embedded boards hosted by the corresponding node) and used for the inference – and the generation of the events representing the anomaly signal. The trained autoencoders are fed with the live data coming from the monitoring infrastructure; each batch of live data (around 6-7 kilobytes) is processed in 11ms, thus generating the anomaly signal event with minimal overhead.

3 Data Analytics at the Edge

In this section we describe edge-specific solutions for data analytics, in particular focusing on embedding Machine Learning techniques considering the peculiar hardware limitations of edge devices.

3.1 Embedded ML Ops

Figure 3.1 shows the classical machine learning lifecycle. The objective of this cycle is to create a model (usually an executable neural network) that is deployed on some edge device, solving some problem like classification of images to detect defective products in a factory. Training a neural network often requires a large amount of resources, for which reason this is usually done in the cloud where the required resources are available.

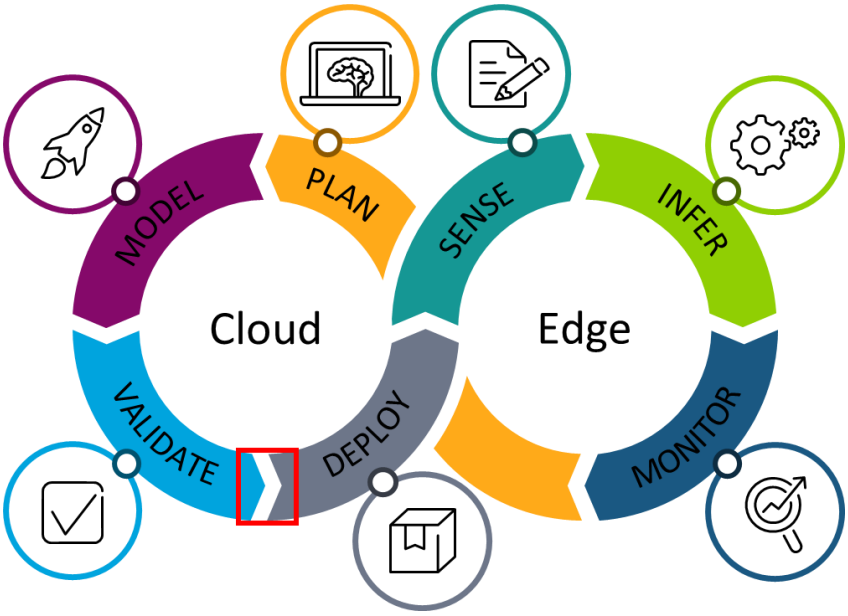

Figure 3.1 - Machine Learning Lifecycle

For the most part, this lifecycle is covered and automated by frameworks such as Kubeflow[4] in conjunction with deep learning frameworks like Tensorflow[5] and Pytorch[6]. However, the more specialized use cases get, the more constrained target devices are, hence tools are missing for realizing the whole lifecycle. Especially the boundary between the validation and the deployment step (marked with a red box in the above figure) is usually a manual and time-consuming task.

As depicted in Figure 3.2, we observe a domain transition here. On the left side, there are data scientists who are familiar with artificial intelligence, scripting languages, and cloud environments. On the right side, embedded engineers, who are familiar with low-level technologies, are responsible for achieving satisfying performance on potentially resource-constrained target devices, e.g., by means of specialized accelerators and toolchains. These two domains are fundamentally different, and experts are required to bridge the gap.

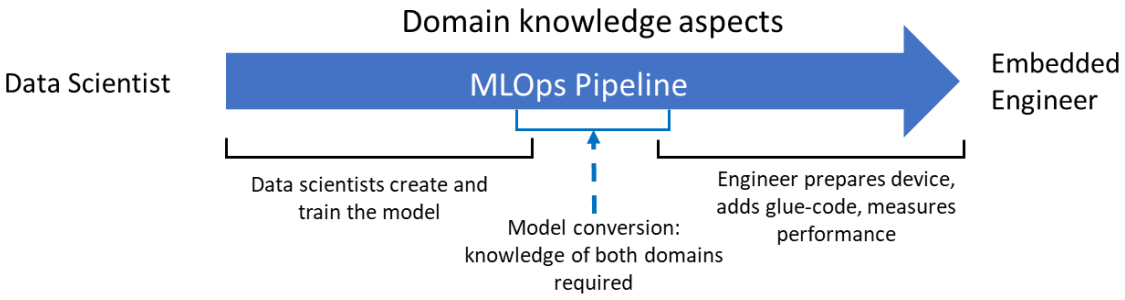

Figure 3.2 - Domain Transition

Additionally, the embedded domain is often very restrictive. Not everything available for the data scientist can be realized without additional effort on the target devices. For example, some features available during design time are missing in embedded inference frameworks, and thus cannot be converted automatically. We might even have highly resource-constrained devices that cannot load large models into main memory.

[4] https://www.kubeflow.org/
[5] https://www.tensorflow.org/
[6] https://pytorch.org/

If the data scientist does not already take that into account at design time, the machine learning lifecycle must be reiterated, which is again a manual and time-consuming task.

To simplify and automate this task, we developed an MLOps (Machine Learning Operations) framework as part of IoTwins that supports developers in creating an automated pipeline (see Figure 3.3) for transforming a model from the data science domain to a model suitable for execution on an edge or IoT device. Using the provided pipeline, the generated models can be optimized, profiled, and verified. This way, the data scientist immediately and iteratively receives feedback on whether a given model executes on the target device with the expected performance and correctness results. Long development steps between lifecycle iterations are avoided.

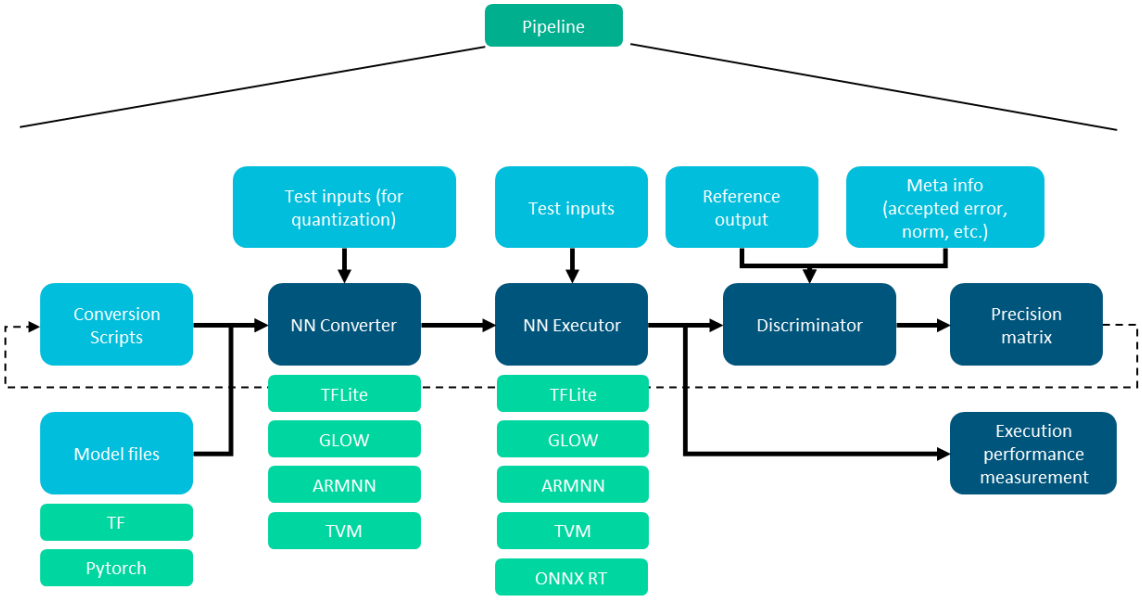

Figure 3.3 - MLOps Pipeline

GitLab[7] is a very popular and well-known tool for realizing traditional DevOps pipelines. Whenever code is changed, executables are built and tested. A key design decision of MLOps has been to not create a new DevOps framework, but instead to provide tooling for realizing pipelines in existing ones. Most developers are already familiar with frameworks like GitLab and can start immediately (Figure 3.4 shows a sample pipeline visualized in GitLab). Additionally, components are designed to be independent from the concrete pipeline framework and deployed in containers, so that at a later point in time, a port to a different DevOps framework, if necessary, is possible with very low effort.

[7] https://about.gitlab.com/

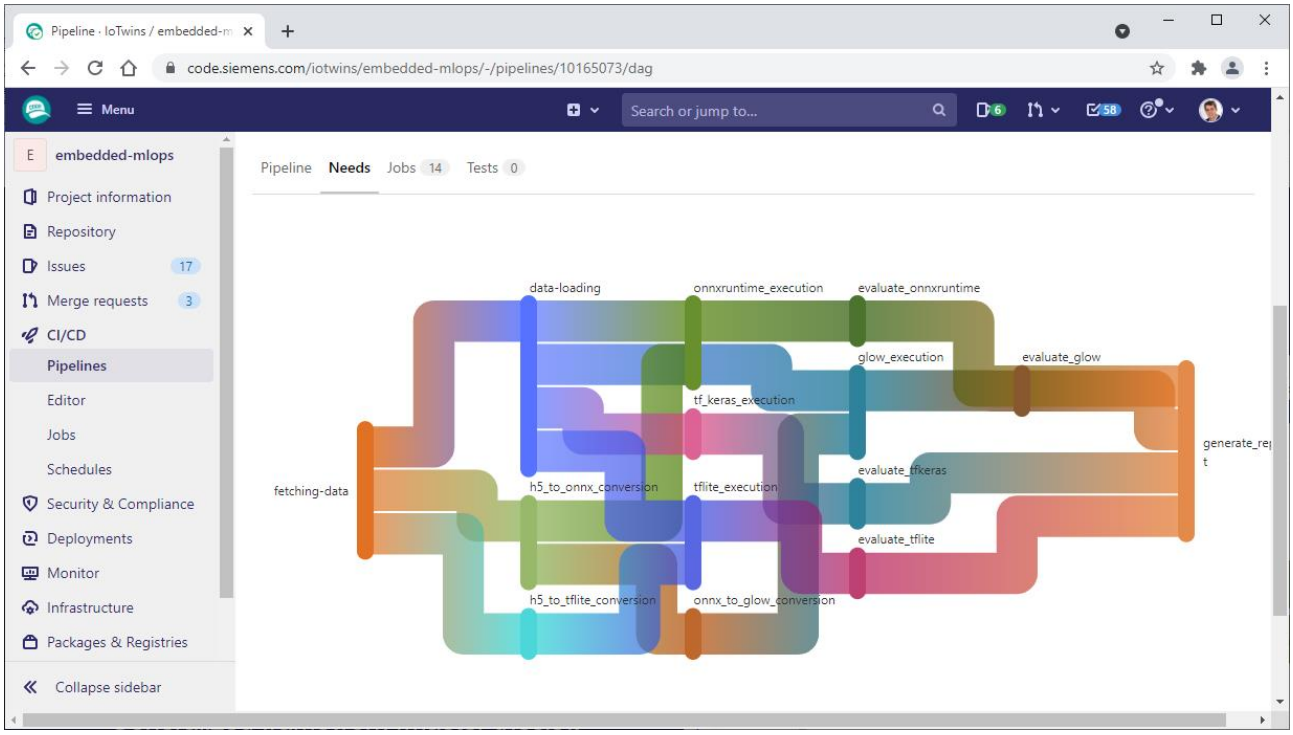

Figure 3.4 - GitLab Pipeline

For speeding up the development, a set of use cases from different domains have been implemented and are provided as examples. New use cases can be realized quickly by forking and adapting existing ones. The framework utilizes templates and hooks to enable customizations with little effort. The final pipeline is executed whenever changes to the model are made by the data scientist. This way, different models are generated, verified, and executed on the target device. Immediate feedback is provided—see Figure 3.5 for a sample report created in the verification step.

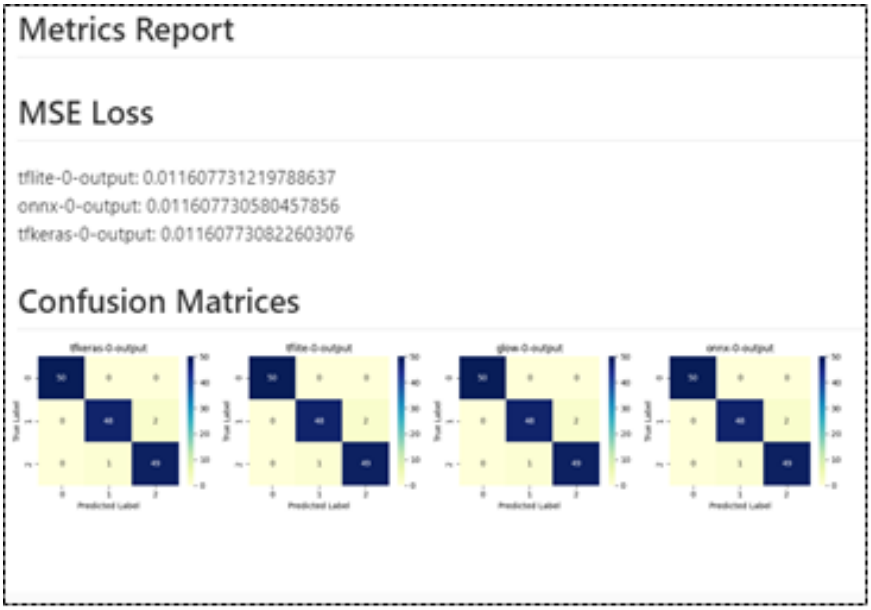

Figure 3.5 - Example Report

In summary, we significantly lower the hurdle for data scientists to develop models for constrained edge and IoT devices by covering white-spots in the machine learning lifecycle. The framework is easy to use and realized in a manner data scientists are mostly familiar with.

3.2 Context Aware Monitoring

CCAM was originally developed to provide a generic method to monitor a black-box system for determining in which working state it is and whether it functions correctly or malfunctions[8]. For this purpose, CCAM observes the system's in- and outputs to determine in what states the corresponding signals are. CCAM avoids a large computational footprint by making all decisions using confidence values and only contextual knowledge. Because CCAM works on the principles of self-awareness and benefits from a hierarchical agent-based architecture, it was implemented in the Research on Self-Awareness (RoSA) framework[9]. Roughly speaking, CCAM consists of one signal state detector for each monitored signal and one additional system state detector. The system state detector abstracts the system's operation based on all present signal states.

For IoTwins two works[10] were published in which the concept of CCAM was enhanced and a novel Smart Grid monitoring methodology was presented for low voltage grids to be able to handle the growing amount and complexity of existing data (see Figure 3.6).

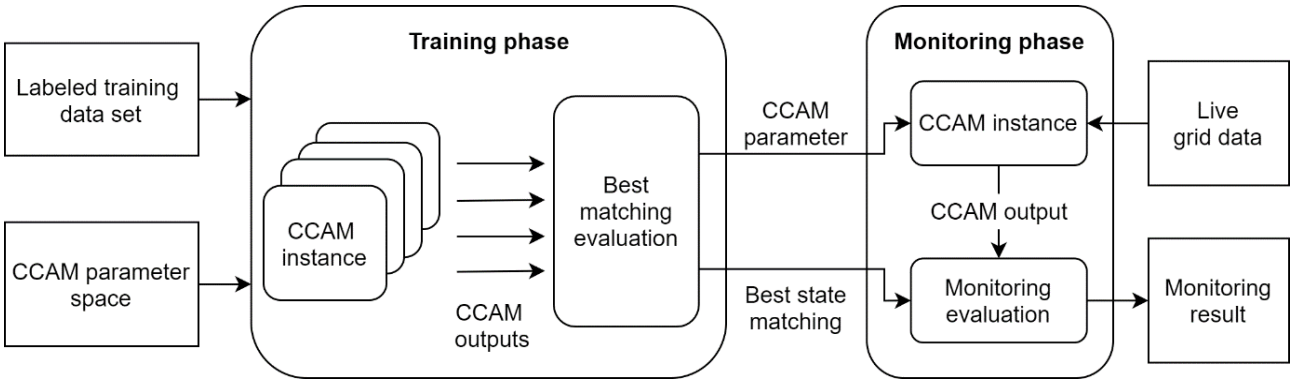

Figure 3.6 - Concept of the proposed Smart Grid monitoring methodology

[8] Götzinger, Maximilian, et al. "Model-free condition monitoring with confidence." International Journal of Computer Integrated Manufacturing 32.4-5 (2019): 466-481.

[9] Götzinger, Maximilian, et al. "RoSA: A Framework for Modeling Self-Awareness in Cyber-Physical Systems." IEEE Access 8 (2020): 141373-141394.

[10] Hauer, Daniel, et al. "Context Aware Monitoring for Smart Grids." 2021 IEEE 30th International Symposium on Industrial Electronics (ISIE). IEEE, 2021.

Hauer, Daniel, et al. "A Methodology for Resilient Control and Monitoring in Smart Grids." 2020 IEEE International Conference on Industrial Technology (ICIT). IEEE, 2020.

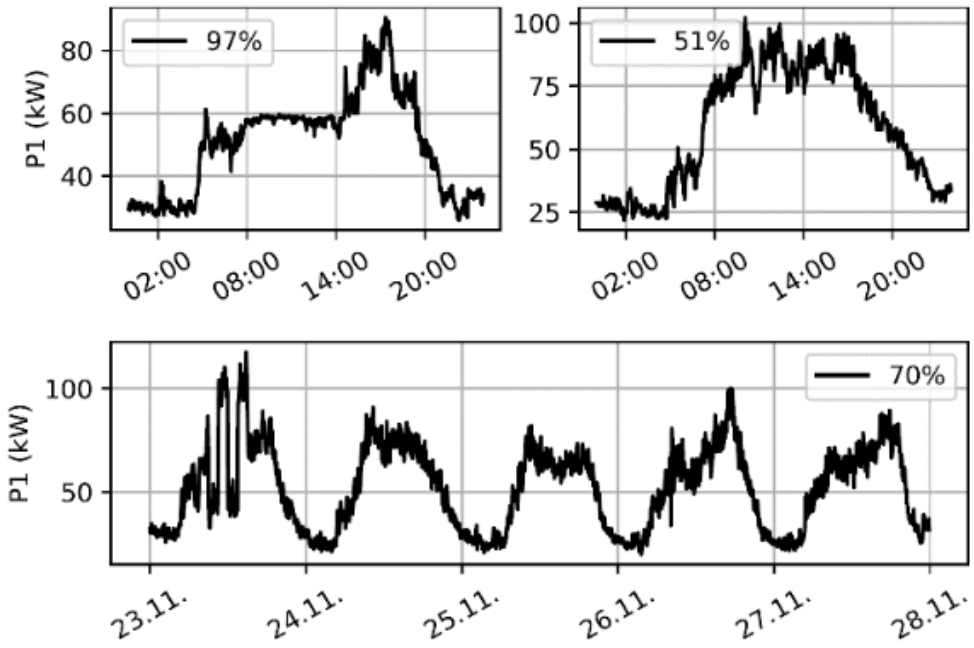

Figure 3.7 - Evaluation results: Upper left: best matching day (97%); Upper right: worst matching day (51%); Lower area: Battery event (70%) and following days

During a training phase (on cloud level), multiple CCAM instances with different parameters are evaluated based on a training data set (e.g., favored daily load profile). As a result, the best CCAM parameter set as well as the best state matching are calculated and used for the monitoring phase (on edge level). From now on, live grid data can be monitored by the single CCAM instance, and every new label under investigation (e.g., a day) will be evaluated. This enables the grid operator to detect anomalous behavior (e.g., a maintenance event) of specific grid segments among numerous others, enabling fast and targeted intervention. Figure 3.7 shows one tested scenario, where the daily load-behavior of one substation was monitored. Due to an installed battery, peakloads above 60 kW should be prevented. As most of the time the battery was fully discharged during the day, the dominant behavior can be seen in the upper left image in Figure 3.7. We trained our CCAM-based monitoring system with this profile and monitored the upcoming days. Battery failures or events (e.g., battery maintainance events) can be succesfully detected as deviations from the trained profile.

3.3 Change Point Detection

Yamanishi et al published works[11] in which they proposed a framework for statistical outlier detection. It assumes that the given data source is non-stationary, and every data is independently drawn from it. A Gaussian mixture model is developed as a statistical model for continuous variables, while a histogram density as that for discrete variables. Outliers in the time series are detected relative to this learned statistical model.

[11] Yamanishi, Kenji, et al. "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms." Data Mining and Knowledge Discovery 8.3 (2004): 275-300.
Yamanishi, Kenji, and Jun-ichi Takeuchi. "Discovering outlier filtering rules from unlabeled data: combining a supervised learner with an unsupervised learner." Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. 2001.

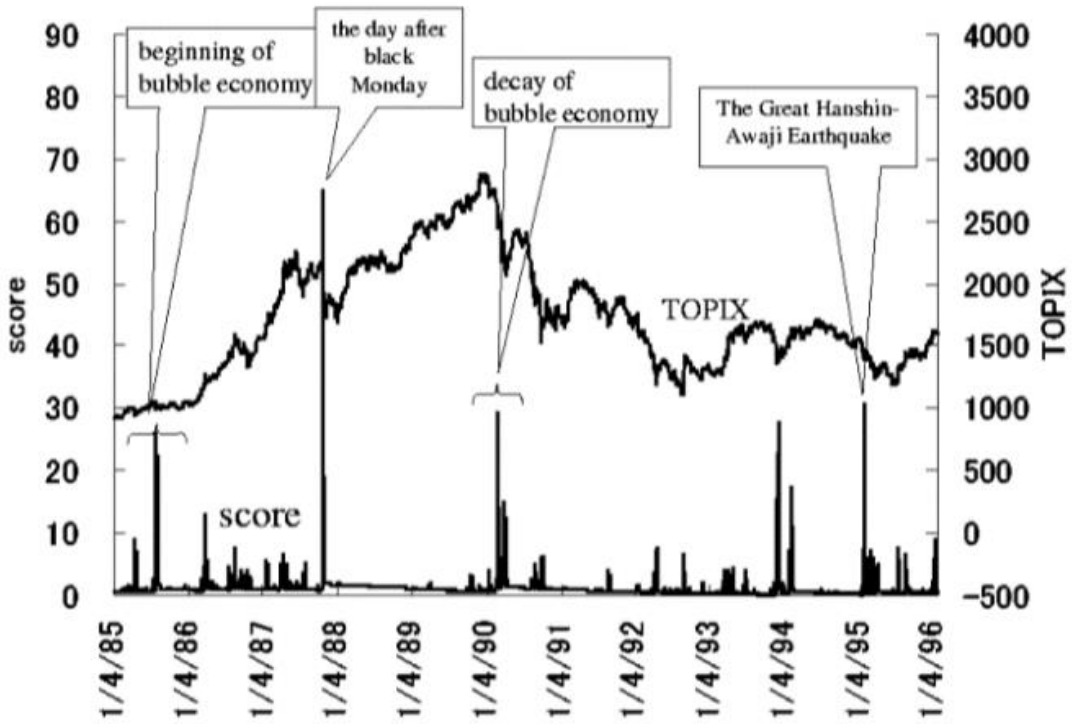

Figure 3.8 - Changepoint detection example using sequentially discounting auto-regression model estimation (analyzing the Tokyo stock price index) [7]

In later works[12] this framework is extended toward these features: 1) dealing with time series data, and 2) detecting change points in a data stream. The result is a framework for detecting outliers and change-points from non-stationary time series. Figure 3.8 has illustrative purpose as it shows the results from the example analyzing the Tokyo stock price index, taken from the original paper describing the change-points method. The change-point score represents the deviations of the current data point from its expectation based on the learned model. The higher the value, the more likely the data point is a changepoint. As it can be seen in the figure, high changepoint scores indeed indicate a change in the observed time series. The textboxes additionally list known historical influences on the stock prices.

[12] Yamanishi, Kenji, and Jun-ichi Takeuchi. "A unifying framework for detecting outliers and change points from non-stationary time series data." Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. 2002.

Figure 3.9 - Changepoint detection for the active power of one simulated household

For IoTwins the framework was implemented and enhances for an entirely different industrial use case: the monitoring of energy grids on the edge level and the cloud level (Testbed 7). The time-series data (e.g., power or voltage) is monitored and use the calculated changepoint detection score (CPD) for the detection of events of interest in monitored grid stations. Figure 3.9 shows exemplary results for the monitoring of the active power of one simulated household. The sudden change around 3pm can be clearly detected as peak in the CPD. Besides that, minor changes in the load profile (e.g., at 7 am) also lead to a higher CPD score.

4 Conclusions

In this document we described software tools designed to work on the edge and to generate events there. The notion of “event” is really wide, and its actual specifics are strictly dependant on the use case. We started by providing a general overview of how such tools have been developed, providing then a couple of examples of deployments performed with the collaboration of testbed partners. In these deployments the two types of events were considered, a Remaining Useful Life estimate and an anomaly signal; both these events are generated on the edge but rely on deep learning models trained on historical data using cloud and high-performance computing resources. Finally, we discuss how to perform analytics on the edge, taking into account the computational resources available on such kind of dispositive and showing how to overcome these limitations through the usage of a software module called MLOps.