Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 127 (2024) 122–128

www.elsevier.com/locate/procedia

10th CIRP Conference on Assembly Technology and Systems (CIRP CATS 2024)

Towards a Testing Framework for Machine Learning Model Deployment in Manufacturing Systems

I. Heider[a,*], J. Baumgärtner[a], A. Bott[a], R. Ströbel[a], A. Puchta[a], J. Fleischer[a]

[a]*wbk Institute of Production Science, Kaiserstr. 12, 76131 Karlsruhe, Germany*

* Corresponding author. Tel.: +49 172 141 1977 ; fax: +0-000-000-0000. *E-mail address:* imanuel.heider@kit.edu

Abstract

The deployment of machine learning models in manufacturing systems presents unique challenges, necessitating robust testing procedures to ensure reliable and efficient operation. This paper proposes an automated testing framework specifically designed to address these challenges, focusing on verifying the correct utilization of data sources, validating model functionality, and assessing the compatibility of the target machine with the deployed model. By automating the testing process, this framework aims to enhance the reliability and effectiveness of machine learning model deployment in manufacturing systems. Through a comprehensive literature review, the paper explores existing methodologies and identifies gaps in current practices. The proposed framework incorporates various test types, including unit tests, integration tests, regression tests, and performance tests, each tailored to the specific requirements of manufacturing systems. Experimental results demonstrate the framework’s effectiveness in detecting errors and failures during the deployment process. Overall, this research contributes to advancing the field of machine learning deployment in manufacturing systems and provides practical insights for practitioners seeking to optimize the reliability and efficiency of their deployed models.

Keywords: Type your keywords here, separated by semicolons ;

1. Introduction

The digitization of the production landscape has led to the proliferation of data-driven solutions in manufacturing systems. New machines are equipped with ever more sensors collecting data at high frequencies, this increasing availability of data has enabled the development of advanced machine learning (ML) models. But especially in small and medium-sized companies, most machines do not have these so-called Industry 4.0 capabilities [4]. This has led to a retrofitting market, with a plethora of companies offering solutions for the integration of additional sensor technology and software tools for data connection to machines and their controls [13]. These new systems can also enable the deployment of machine-learning models. However, most of these systems are vendor-specific and are neither transferable nor scalable. Current development trends, therefore, try to automate the integration into a seamless process. However, in even in these cases, the main focus lies with established ML application areas (i.e. the domain of ‘Big Tech‘) where computing power and data are readily available. They do not consider the specific requirements of machine learning models in a manufacturing context. This paper aims to close this gap by proposing a testing framework specifically designed to allow for risk-free deployment of machine learning models into manufacturing systems. The framework is designed to be easily integrated into existing development processes and to be scalable to different machine-learning models.

10.1016/j.procir.2024.07.022

2. State of the Art

When deploying machine learning models in manufacturing systems, two factors need to be considered: First, stable execution of the model on the target machine must be ensured, and second, the model must be able to handle the data from the target machine.

Especially in systems that have limitations in terms of computing power, an examination of these aspects is relevant. In the context of handling and assembly systems, a deployment close to the machine on industrial PCs or even embedded systems is conceivable. If it is not possible to clarify without doubt in advance whether the target system can meet the requirements of the model, a testing framework is indispensable.

2.1 Running the model on the target machine

The first step in deploying a machine learning model is to ensure that the model can run on the target machine. This is one of the tasks typically handled by machine learning deployment frameworks. One key problem that machine learning deployment frameworks seek to solve is the seamless integration of models into production environments. Traditional machine learning frameworks focus primarily on model training and evaluation [15], often overlooking the complexities of deploying models in a scalable and efficient manner. These are two essential prerequisites for the deployment in manufacturing contexts especially when deploying to edge devices with limited computing power. Deployment frameworks bridge this gap by providing components and APIs for model serving, enabling easy integration with existing systems and applications. The most popular frameworks include:

- TensorFlow Serving
- TorchServe
- MLflow
- ONNX Runtime

TensorFlow Serving is an open-source framework developed by Google for deploying machine learning models in production environments [15]. It provides a flexible architecture for serving models in a variety of formats, including TensorFlow, TensorFlow Lite, and TensorFlow.js. TorchServe is the equivalent of TensorFlow Serving for PyTorch models [6]. A library-agnostic framework is represented with MLflow, an open-source platform for managing the machine learning lifecycle, including model deployment [19]. It provides a REST API for deploying models in a variety of formats, including TensorFlow, PyTorch, and scikit-learn. All of these frameworks mainly deal with the deployment of models on servers and are not specifically toward deployment on edge devices or embedded systems. For this purpose, ONNX Runtime is a better fit. The ONNX Runtime is an open-source framework developed by Microsoft for deploying machine learning models in production environments. It specifies a standard format for representing machine learning models, enabling interoperability between different frameworks and hardware platforms. The possibility to export models of different ML frameworks into the ONNX format as well as the lightweight structure of the ONNX Runtime allows its use in manufacturing contexts. For example, Beckhoff Automation integrates the ONNX Runtime into some Industrial PCs as standard and provides for the integration of the ML workflow into the PLC (Programmable Logic Controller) [3]. In summary, it can be said that there are many frameworks for deploying machine learning models on the target machine. However, these frameworks assume that data is readily available.

2.2 Handling the data from the target machine

In truth, connecting data sources to machine learning models in manufacturing systems can be challenging. The data interfaces and application programs on controllers may be outdated and difficult to work with. Additionally, there are multiple communication standards in use, which can complicate the process [13]. Different approaches to solving these problems exist, with some requiring additional hardware. In the following, only approaches that do not require additional hardware are discussed.

Commercial solutions for industrial data acquisition exist, such as PTC’s KEPServerEX and Ignition by Inductive Automation [12, 10]. These applications provide interfaces for conventional communication standards and PLCs, primarily targeting SCADA applications. On the other hand, open-source libraries like Apache Software Foundation’s PLC4X

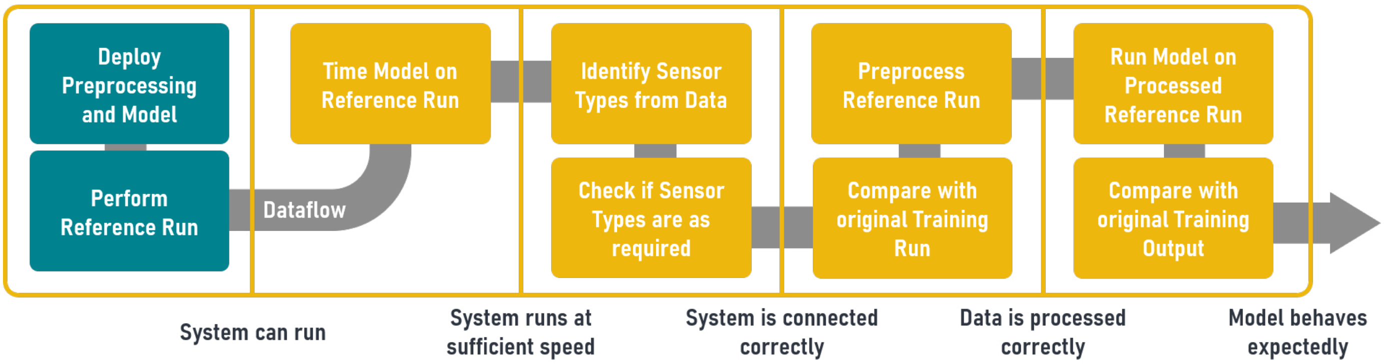
Deploy Preprocessing and Model
Perform Reference Run
Dataflow
Time Model on Reference Run
Identify Sensor Types from Data
Check if Sensor Types are as required
Preprocess Reference Run
Compare with original Training Run
Run Model on Processed Reference Run
Compare with original Training Output
System can run
System runs at sufficient speed
System is connected correctly
Data is processed correctly
Model behaves expectedly

Figure 1 The proposed testing framework

and Streampipes, as well as the open62541 implementation of OPC UA [5, 18, 17], enable connectivity for various communication standards and streaming data handling.
While newer machines utilizing protocols like OPC UA or MQTT tend to have consistent signal designations, this may not be the case for retrofitted brownfield machines.
This means that the data sources must be configured for each machine individually, and the deployment process needs to check whether the correct data sources are used.
Even if all data sources are properly configured, they still need to be connected to the machine learning model. This is typically done using a data adapter. Data adapters serve as a middle layer between the machine-specific data sources and the machine-learning model. Typically, the data adapter deals with data acquisition and, if needed, data preprocessing and data transformation. For this work we will be using the holistic approach with the working title MyGateway presented in [9] The application can be understood as a framework of wrappers for data source-specific libraries addressing the ingestion of data from different sources and its provisioning to different sinks. An advantage of this approach is the ability to quickly incorporate new protocols. It provides templates into which the relevant libraries, such as the above-mentioned PLC4X, can easily be integrated. Open-source projects offering similar functionalities exist as well. InfluxData's TICK stack and the Apache project Streampipes are noteworthy examples [1, 18]. The TICK stack, which is built around the InfluxDB time series database, focuses on ingesting and persisting time series data. Apache Streampipes also covers these functionalities (InfluxDB being among the tools used by Streampipes), but in addition, offers a low-code environment in which users can build their own data processing pipelines.

3. Proposed Framework

To develop the proposed testing framework, a solution-neutral representation of the ML pipeline is needed. The ML pipeline can be divided into three parts:

- Data ingestion
- Data preprocessing
- Model execution

Each of these parts is a potential source of errors. While the first part of the ML pipeline is handled by the data adapter, the second part is handled by a preprocessing pipeline with the third part being handled by the ML model. Grouping the data adapter and preprocessing pipeline we can define two components that need to be tested: how the data arrives at the model and how the model processes the data. The full framework is shown in Fig. 1. Each stage gate consists of a test suite that checks the respective component. The aim of the test suite is not only to provide a go / no go decision but also to provide information about the source of the error. As such the tests have multiple stages systematically narrowing down the source of the error along the machine learning pipeline. The first stage starts with traditional deployment frameworks which solely verify the model's flawless execution. After these tests, we know that the system can run. The second stage tests whether the hardware is capable of running the pipeline with sufficient speed, i.e. if the model output occurs within an acceptable period. The basis for the following tests is a reference control program. This reference control program is part of the validation set during the training of the machine learning model. During deployment, the reference control program is executed on the target machine. The third stage then uses the recorded data from the reference control program to check whether the data adapter has been connected to the correct data sources. Once this has been verified, the fourth stage checks whether the data preprocessing pipeline correctly transforms the data. Finally, a fifth stage checks whether the model behaves as expected on the reference control program on the target machine. If all of these tests are passed the system is ready for deployment. Each test is explained in detail in the following subsections. This deployment procedure rollout of machine learning models can be grouped into three stages. The first stage is the development of the machine learning model. The second stage is the deployment stage in which tests are run to verify the correct deployment of the model. The third stage is the actual rollout of the model on the target machine. A graphical representation of the stages is shown in Fig. 2.

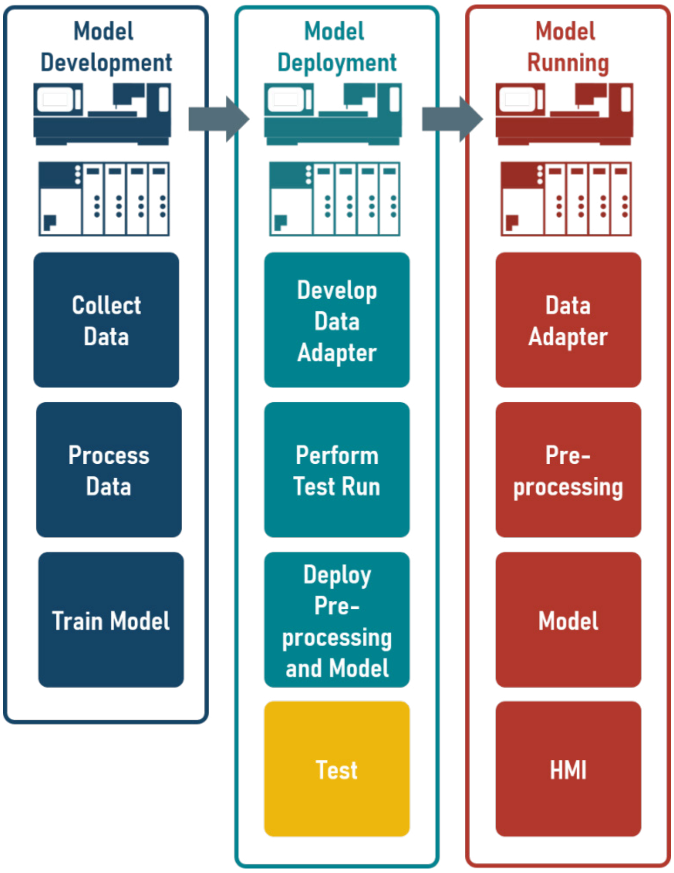

Figure 2 The stages of deployment of a machine learning model using the proposed framework

3.1 Testing the Data Pipeline

An evident requirement for an ML pipeline is that the model must be able to process its output. At this point, no attention is paid to the correctness of the output. This merely verifies that the incoming data can be processed. This is checked using existing frameworks by simply checking whether the model runs without errors. The more serious case is when the model runs without errors but produces wrong results due to wrong data. This can be caused by a variety of reasons, for example:

- The data is not in the expected distribution
- The data sources are not configured correctly

The first problem is caused by the fact that the model was trained on a different data distribution than the data from the target machine. Most target machines are bound to have a slightly different data distribution than the training data. This phenomenon is known as domain shift [16]. This problem can be detected through a comparison of the data distribution of the training data and the data distribution on the target machine. This requires a certain amount of data from the target machine, which is not always immediately available. For this reason, we propose a reference run designed in such a way that it generates data with the same structure as the training data. Such a reference run is to be performed on the target machine. The difference between the two distributions can then be used to detect a possible domain shift. A simple metric to determine the similarity between two distributions is the mean squared error [2]:

$$Domain\ Shift = \frac{1}{n}\sum_{i=1}^{n}\|x_i - y_i\|^2 \quad (1)$$

Where x_i is the i-th value of the reference distribution and y_i is the i-th value of the target distribution. While this approach is sufficient for simple data streams, many machine learning models in manufacturing aggregate data from different sensors, possibly providing measurements in different units. While it is possible to normalize the data before training the model, this is not always done. In this case, one needs to take into account that the different sensors have different amplitudes. This can be done by using a weighted mean squared error:

$$Domain\ Shift = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^T W (x_i - y_i) \quad (2)$$

Where W is a diagonal matrix encoding of weights for each sensor. Note that this domain shift is to be computed for each sensor j separately. The total domain shift is then the sum of the domain shift for each sensor:

$$Domain\ Shift = \sum_{j=1}^{m}\frac{1}{n}\sum_{i=1}^{n}\left(x_{i_j} - y_{ij}\right)\quad W_j\left(x_{ij} - y_{ij}\right) \quad (3)$$

The weights are determined based on the sensor range as well as its measurement unit (the latter is relevant if different sensors measure the same physical quantity at different scales). This can be thought of as a kind of normalization of each sensor.

Such a normalization requires knowledge of the sensor range. Although it is possible to determine the sensor range using the data from the reference machine run, this data does not necessarily reflect the full range of the sensor. To solve this issue, we require metadata for each data source in the data adapter. This metadata should contain the sensor range and the physical unit of its measurements. Apart from the sensor range and the unit, the metadata must further include information on exactly which signal is being captured by the sensor. For example, a CNC controller typically allows for the logging of data on power consumption, speed, and current for every axis. A unique identifier for each of these signals thus has to include information on the axis it belongs to.

While such an identifier can help detect improperly configured data sources, it does not cover all possible error sources related to the identification of a data signal. In practice, the data adapter is often configured using a configuration file that might contain errors. For this reason, we propose a second test that will check that all sensors are properly identified and connected. Here we will investigate each data source individually.

First, it must be verified that the correct data sources are connected to the pipeline. If the model expects an input vector consisting of power consumption, speed, and current, the data sources for these values must be used in the correct places. A correct selection of data sources can be determined through a parameter identification approach. In this case, we propose the analytical parameter identification method presented in [7]. The method uses data from a reference machine run to identify the machine's parameters. It establishes the identity of the parameters through the characteristics of their signal curves. A unique movement sequence is derived for each axis of a machine. These runs are then carried out on the machines to be examined. A specific motion sequence for all axes is specified in the NC code. The comparison of signals retrieved from the NC control and the expected path of the axes is used to identify the data sources containing the axes' positions. This conclusion is shown in Fig. 3.

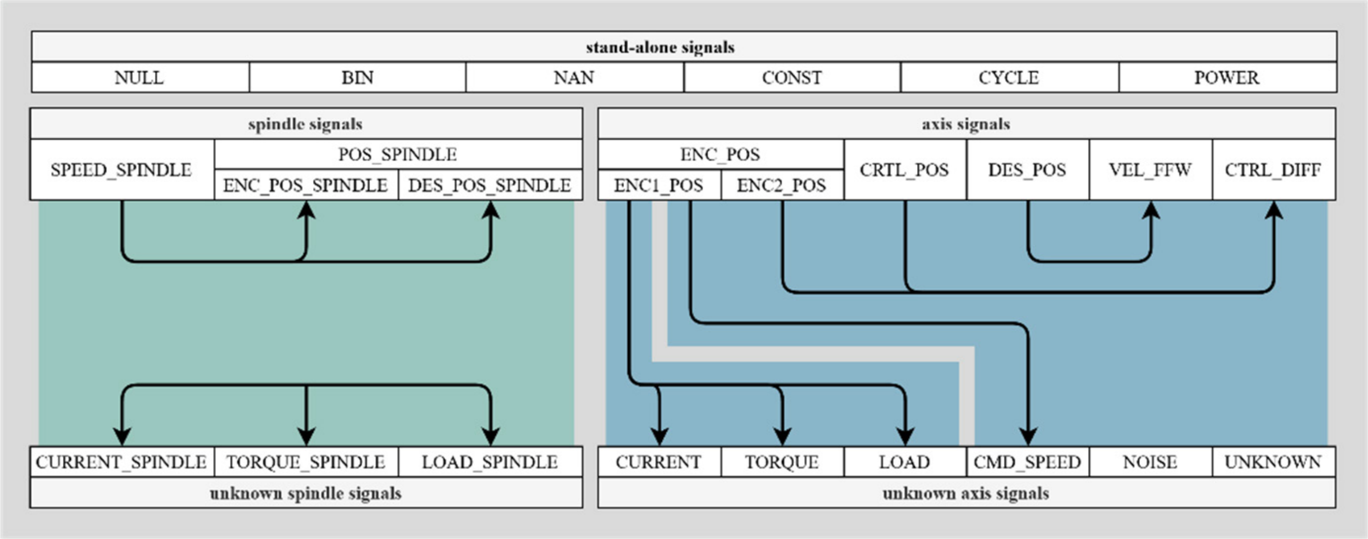

Figure 3 Processes of reference runs and a signal identification [7]

The signals occurring during the reference runs can be recorded via MyGateway. These recordings represent the input of the analytical approach to parameter identification. Within the framework of the 3 pre-processing stages, trivial signals (constant, zero, boolean, etc.) as well as signal classes required for later identification stages, such as position signals, are determined. Based on this, all signals related to the spindle are put into one class and all signals belonging to axes are determined in a two-stage approach as illustrated in Fig. 3. The spindle and axis identification stages make use of analytical relationships. On the one hand, the signals' classes - i.e. spindle or axis - are established. Then, the assignment to their corresponding axes is carried out via the information from the reference runs. Thus, for the respective machine axes, the associated axes and their classes are given, which can be checked against the input requirements of the model. After this test, we can verify that the correct data sources are used in the correct places. This, however, does not mean that the data sources have been configured correctly. Often data is preprocessed before it is fed into a machine-learning model. It must be ensured that the data is preprocessed in the same way as it was during model training. If one has access to the raw data as well as the preprocessed data used in the training phase, then this can be done by preprocessing this raw data using the preprocessing pipeline on the target machine. Ideally, the data preprocessed on the target machine should be identical to the data processed during the initial model training phase. If one only has access to the preprocessed training data (and not the raw data), then this can be done by preprocessing raw data on the target machine after each sensor has been identified. The preprocessed data should now be similar to the preprocessed training data. Each comparison can be done using the same weighted mean squared error as in equation (2) but applied to each sensor individually. If the data is not preprocessed in the same way, then the difference between the two should be large. The magnitude of an acceptable or unacceptable difference is at the discretion of the developer and can vary greatly depending on the application.

4. Testing the Machine Learning Model

Even when the data adapter is working correctly and the domain shift is within acceptable boundaries, the model output can still be inadequate. Apart from the obvious case of a model producing bad results, another important consideration is the frequency at which the model can run. In most manufacturing scenarios the model will run in a loop, where it will be executed at a certain frequency based on the cycle time of a production process. If the model takes longer to execute than the cycle time of the machine, then the model will not be able to keep up with the machine. This means we need to test whether the model can run at the required frequency. This can be tested by running the validation data through the model and measuring the time it takes to execute.

4.1 Testing Model Behaviour

Inherent in model predictions is an uncertainty field around the corresponding prediction, since the input data and model are subject to uncertainty. Existing approaches of uncertainty quantification in machine learning are however unsuitable as a test of model behavior because at least a deeper understanding of the model is required. Additionally, they often use a complex variational process between data and network architecture. Thus, these approaches do not apply to proprietary black box models. Due to the existing research gaps in the area of uncertainty determination of black box models, we propose only a plausibility check for the predictions as a final test of the model behavior. For this purpose, the covariance can serve as a representation of the relations between the input domains.

$$Relation\ between\ inputs = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n-1} \quad (4)$$

Here, x_i is also the i-th value of source domain Ds and yi the i-th value of the target domain for the j-th sensor. The covariance provides a rudimentary description of the difference between the two domains. This relation can now be used by introducing a variation parameter to obtain similarly distributed parameters $y*_i$.

$$Relation\ between\ inputs = \frac{\sum_{i=1}^{n}(x_i-\bar{x})\left(a(y_i-\overline{y})\right)}{n-1} \quad (5)$$

Here a is normally distributed noise with a magnitude equal to the covariance between the input domains. This effectively maps the input uncertainty of the initial data to the expected output uncertainty of the neural network. If a network performs outside of these bounds, then it is likely that the network is not behaving as expected. However, this evaluation is only a plausibility check and cannot replace a final comprehensive uncertainty analysis.

5. Sample Implementation

Since the framework is designed to be general purpose, it can be implemented in different ways. This section will describe a possible implementation of the proposed testing framework for a use case in additive manufacturing. A possible setup is adapted from [8] and shown in Fig 4. This use case constitutes a specific condition check for the Arburg freeformer. Due to the complexity of the process and the wide variety of available materials, it is important to ensure proper conditions before a printing job is started. A subfunction that enhances operational readiness is the detection of an offset of the discharge unit. Under particular circumstances, inexperienced operators may succeed in causing a misalignment of the discharge unit. ML-based detection of such an offset is desirable. In this case, the volume flow of discharged material serves as an indicator of a possible offset. This offset detection method uses a reference run akin to the one described in Section 3 A singlelayer test part is printed. The signal curve during this reference run can be consulted if the identity of the required parameters (here exclusively the volume flow) is not known in advance. The data adapter is implemented using the MyGateway framework [9] and uses Arburg's proprietary OPC UA server to access the data. Although the data adapter is logically responsible for sending data to the preprocessing stage, a message parser is typically used in practice to parse and send the data. In this case, the RabbitMQ message broker is used to send the data to the preprocessing stage. The preprocessing stage is implemented as a custom Python class object offering RabbitMQ interfaces and metadata regarding the required data types. This serves as a first safety check to ensure that the data adapter is sending the correct data. From the preprocessing stage, the data is sent to the machine learning model which is in this case an autoencoder. The model is packaged in the same Python class as the preprocessing stage and offers the same interfaces. This allows the model to be tested in the same way as the preprocessing stage. The proposed testing framework can now be implemented using the Pytest framework [11]. Using Pytest fixtures the data adapter and the machine learning model can be instantiated and tested. This requires stored data of the reference run on the new machine as well as the training data. While it is possible to have the test trigger a reference run, for safety reasons it is often better to trigger the reference run manually. The test now accesses the data from an InfluxDB database [14] and performs the checks described in the previous section. Since the tests are implemented using the Pytest framework, they can be run automatically using a CI/CD pipeline. If issues occur during testing, the logs of the test can then be used to identify the problem. Once all checks are successful, the model can be deployed to the machine. To make this as smooth as possible all components are packaged in docker containers. The model can then be hooked up to a visualization tool such as Grafana to monitor the model's performance. A useful side effect of the sensor selfidentification is that the data visualization can also be configured automatically.

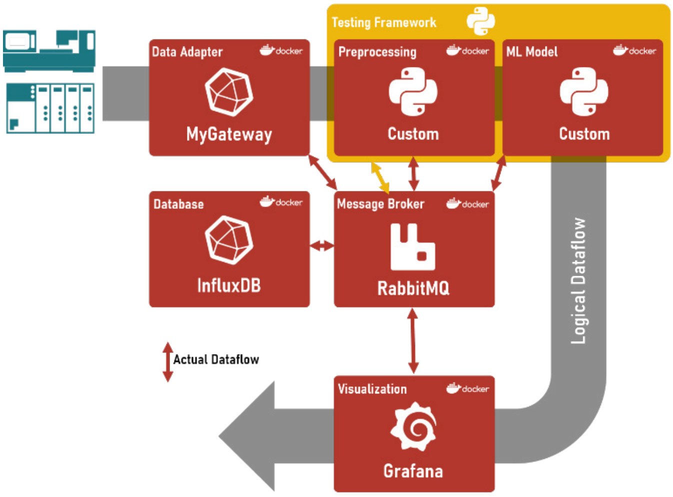

Figure 4 Sample implementation of the proposed testing framework

6. Conclusion

In this paper, we have presented a framework that checks whether the requirements toward a successful deployment of ML models in manufacturing contexts are met. The framework not only checks whether the model can run on a target system but also checks the correctness and/or plausibility of input data as well as model output. This was done by testing the data adapter and the machine learning model separately. All in all the deployment of machine learning models in manufacturing is a complex task where few tools exist to help developers and users. Standardized testing frameworks like the one presented in this paper can help to make the deployment of machine learning models in manufacturing more robust and reliable. But to improve deployment on the shop floor, wider adoption of such frameworks is needed. We hope that our work has helped to argue the need for such frameworks and that it will help to improve the deployment of machine learning models in manufacturing.

Acknowledgements

This publication is based on the results of the AutoLern research and development project. This research and development project is funded by the German Federal Ministry of Education and Research (BMBF) within the funding measure ProLern (Funding Number: 02P20A025) and managed by the Project Management Agency Karlsruhe (PTKA). The authors are responsible for the content of this publication. The authors of this paper thank the Ministry for the funding.

References

[1] Gunnar Aasen. 2017. Introduction to influxdata's influxdb and tick stack. (Sept. 2017). https://www.influxdata.com/blog/introduction-to-influxdatas-influx db-and-tick-stack/

[2] Kavutse Vianney Augustine and Huang Dongjun. 2009. Image similarity for rotation invariants image retrieval system. In 2009 International Conference on Multimedia Computing and Systems. IEEE, Ouarzazate, Morocco, 133–137. doi: 10.1109/MMCS.2009.5256716..

[3] Beckhoff Automation. 2023. Twincat 3. machine learning- und neural network inference engine. (June 2023).

[4] Antonella Biscione, Chiara Burlina, and Annunziata de Felice. 2023. Knowledge flows and innovation: a pseudo-panel approach. Applied Economics, 0, 0, 1–16. eprint: https://doi.org/10.1080/00036846.2023.2207812. doi: 10.1080/00036846.2 023.2207812.

[5] The Apache Software Foundation. 2017. Plc4x: the universal protocol adapter for industrial iot. (2017). https://plc4x.apache.org/.

[6] The Linux Foundation. 2020. Torchserve master documentation. (2020). https: //pytorch.org/serve/.

[7] Philipp Gönnheimer, Robin Ströbel, and Jürgen Fleischer. 2023. Analytical approach for parameter identification in machine tools based on identifiable cnc reference runs. In Production at the Leading Edge of Technology. Mathias Liewald, Alexander Verl, Thomas Bauernhansl, and Hans-Christian Möhring, (Eds.) Springer International Publishing, Cham, 494–503. isbn: 978-3-031-18318- 8.

[8] Imanuel Heider, Huitian Yu, Nikolai Krischke, Benjamin Wirth & Jürgen Fleischer (2023). KI-Einsatz in KMU: Einstiegshürden ausräumen / Clearing entry hurdles for AI deployment in SMEs – Artificial intelligence for German SMEs. In wt Werkstattstechnik online (Vol. 113, Issues 07–08, pp. 282–287). VDI Fachmedien GmbH and Co. KG. https://doi.org/10.37544/1436-4980-2023-07-08-16

[9] Jonas Hillenbrand, Philipp Gönnheimer, Eduard Gerlitz, and Jürgen Fleischer. 2021. Design and implementation of a holistic framework for data integration in industrial machine and sensor networks. Procedia CIRP, 104, 1771–1776. 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0. doi: https://doi.org /10.1016/j.procir.2021.11.298.

[10] LLC Inductive Automation. 2022. Inductive automation. ignition: user manual. (2022). https://docs.inductiveautomation.com/display/DOC81.

[11] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. 2004. Pytest. (2004). https://github.com/pyt est-dev/pytest.

[12] Sebastian Krüger. 2022. Leveraging kepware in iiot and how to mitigate its current technical limitations. (June 2022). https://learn.umh.app/blog/leveragi ng-kepware-in-iiot-and-how-to-mitigate-its-shortcomings/.

[13] Juergen Lenz, Thorsten Wuest, and Engelbert Westkämper. 2018. Holistic approach to machine tool data analytics. Journal of Manufacturing Systems, 48, 180–191. Special Issue on Smart Manufacturing. doi: https://doi.org/10.1016/j.j msy.2018.03.003.

[14] Jalal Mostafa, Sara Wehbi, Suren Chilingaryan, and Andreas Kopmann. 2022. Scits: a benchmark for time-series databases in scientific experiments and industrial internet of things. In Proceedings of the 34th International Conference on Scientific and Statistical Database Management (SSDBM '22) Article 12. Association for Computing Machinery, Copenhagen, Denmark, 11 pages. isbn: 9781450396677. doi: 10.1145/3538712.3538723.

[15] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflowserving: flexible, high-performance ml serving. (2017). arXiv: 1712 . 06139 [cs.DC].

[16] Felix Ott, David Rügamer, Lucas Heublein, Bernd Bischl, and Christopher Mutschler. 2022. Domain adaptation for time-series classification to mitigate covariate shift. In Proceedings of the 30th ACM International Conference on Multimedia (MM '22). Association for Computing Machinery, Lisboa, Portugal, 5934–5943. isbn: 9781450392037. doi: 10.1145/3503161.3548167.

[17] Julius Pfrommer. 2017. Semantic interoperability at big-data scale with the open62541 opc ua implementation. In Interoperability and Open-Source Solutions for the Internet of Things. Ivana Podnar Žarko, Arne Broering, Sergios Soursos, and Martin Serrano, (Eds.) Springer International Publishing, Cham, 173–185. isbn: 978-3-319-56877-5.

[18] Dominik Riemer and Philipp Zehnder. 2023. Apache streampipes documentation. (June 2023). https://streampipes.apache.org/docs/user-guide-introductio n/.

[19] Matei Zaharia et al. 2018. Accelerating the machine learning lifecycle with mlflow. IEEE Data Eng. Bull., 41, 4, 39–45.