Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 130 (2024) 188–193

www.elsevier.com/locate/procedia

57th CIRP Conference on Manufacturing Systems 2024 (CMS 2024)

Efficient Deployment of Machine Learning Models in Manufacturing and Industrial Environments using ROS

Marvin Frisch[a,*], Jan Baumgärtner[a], Imanuel Heider[a], Alexander Puchta[a], Jürgen Fleischer[a]

[a]*wbk - Insitute of Production Science, Kaiserstrasse 12, 76131 Karlsruhe, Germany*

* Corresponding author. Tel.: +49-1523-9502621. *E-mail address:* marvin.frisch@kit.edu

Abstract

This paper presents a deployment concept that aims to overcome the challenges in the implementation of Machine Learning (ML) models in manufacturing and industrial environments. In these contexts, robots are not typically viewed as production machines. However, the potential for applying advanced techniques such as condition monitoring extends beyond production lines to encompass robotic systems. As a result, there arises a need for a modular solution that integrates into the existing ecosystem while accommodating the requirements of robotic environments. By embracing modularity and interoperability, our proposed deployment concept not only addresses the challenges specific to industrial robotics but also fosters a holistic approach to enhancing operational efficiency and performance in diverse manufacturing settings.

For this, an easily customizable and adjustable system that handles both data acquisition and data transfer is needed. By using the Robot Operating System (ROS) for all necessary data handling, we achieve a highly modular, efficient, and easy-to-use low-code deployment pipeline. Our approach splits the different processing steps into separate nodes and automatically sets up all necessary communication channels, achieving high interchangeability and a quick time-to-deploy. The approach is explained in detail and demonstrated for the real use case of deploying models to monitor handling robots.

Keywords: deployment pipeline; robotic systems; data processing

1. Introduction

The usage of Machine Learning (ML) methods in industrial manufacturing is still on the rise. With the possible use cases ranging from machine-specific tasks like predictive maintenance to workpiece-specific ones like quality prediction, there remains much untapped potential [1]. This holds especially true in the world of industrial robotics, where many of the advanced supervision methods already established for more traditional production machines are still in their infancy [2]. Since robotic systems are gaining importance in industrial contexts due to increasing demands for both flexibility and productivity [3], the potential benefits of implementing ML based prediction and fault detection methods are also increasing.

To unlock this potential, hurdles like data availability, suitable model design and the need for computing resources must first be overcome [4]. An additional challenge that needs to be addressed before using a newly trained model is its deployment [5]. Deployment describes the process between the development and execution of a piece of software [6]. It includes, but is not limited to, installing, integrating, activating, and updating the software [7]. In order to effectively deploy an ML model problems like data transfer to and from the model, environment management, and software testing need to be addressed [8].

A previous work presents a structure for a standardized ML data pipeline [9]. It separates the process steps that need to be close to the machine (data collection) and those that can be executed elsewhere (data processing, persistence, visualization) into separate containers and connects them using a message broker.

This approach is the first step towards unifying the deployment of ML methods. It succeeds in providing reusable building blocks for the separate process steps and an easily modifiable communication between them. However, there are improvements to be made. Using multiple separate data sources requires significant implementation efforts, as does introducing additional processing steps.

10.1016/j.procir.2024.10.074

To improve on those shortcomings, this paper presents an approach based on the Robotic Operating System (ROS). In robotic research contexts, ROS has become the de-facto standard messsaging service provider [10]. Its successor ROS2 increases its usability for industrial environments by improving upon factors like security, reliability, and support for large scale embedded systems [11]. We aim to utilize those improvements to achieve a highly modular low-code deployment pipeline for both robotic and non-robotic production equipment that can be easily introduced into existing robotic infrastructures.

2. State of the Art

Additionally to the previous work mentioned in section 1, numerous other approaches to ease and standardize deployment of ML models exist in the state of the art. Some notable examples are evaluated in the following.

2.1. General ML Pipelines

The Kubeflow Pipelines platform [12] enables the compilation of multiple components into an executable and containerized pipeline. This ensures cross-platform portability and efficient use of computing resources. While it provides a reliable way to implement the data processing steps, the data transfer from the machine to the model still requires significant coding effort.

In [13], a pipeline for the automation of data preparation and model training is presented. At the time of writing, it is only suitable for binary classification tasks, however, it is still in active development. Training data is provided in a file format, the pipeline is not suitable to be integrated into an existing data infrastructure. Its intended use case is the comparison and training and not the active deployment of models.

A bigger scope is presented in [14]. The pipeline proposed there encompasses not only the deployment of the model, but also the problem analysis, feature engineering, and model development. The presented steps needed to use a model, i.e. data extraction, data preparation, and model usage, are similar to those in [9]. Being a more structural analysis of the development of ML models, no sample implementation is provided.

General ML pipelines offer good ways to train models and get inference results. The deployment of the model, however, still requires manual adaptation to the environment at hand. In the following, existing approaches with the ROS environment in mind are evaluated.

2.2. ROS Deployment Infrastructures

[15] proposes a robot-centric deployment approach that is not unsimilar to ours. It uses ROS for communication with the robotic agent and splits the separate tasks within the pipeline into separate nodes. However, it focuses on image classification and how that can be used to infer control decisions like avoiding detected obstacles. As such, abstracting this approach for cases with non-image data or differing ML methods would require considerable development effort.

A similar approach is explored in [16], where in addition to image data, a separate track of nodes handles audio input in order to provide a holistic human-robot-interface. While integrating multiple different data sources, the developed pipeline is not meant to be adapted to different use cases and is instead focused on solving a specific problem.

In conclusion, according to our research there is currently no modular and scalable approach to provide a data pipeline for the deployment of generic ML models in the context of industrial robots. A generalized implementation that is compatible with existing infrastructure has the potential to save development effort over specialized solutions for every possible problem.

3. Proposed Deployment Pipeline Architecture

The goal of our development efforts is therefore to develop a generally applicable deployment pipeline that offers high modularity and scalability. In the following, the proposed pipeline and the design decisions made to achieve those objectives are presented. The level of detail examined increases from the big picture to a detailed examination of the individual elements.

3.1. The pipeline as a whole

Similar to the existing approaches shown in [9] and [15], we split up all functionality into separate independently executable programs. The primary building blocks of the pipeline consist of preprocessing, processing, and persistence. For this

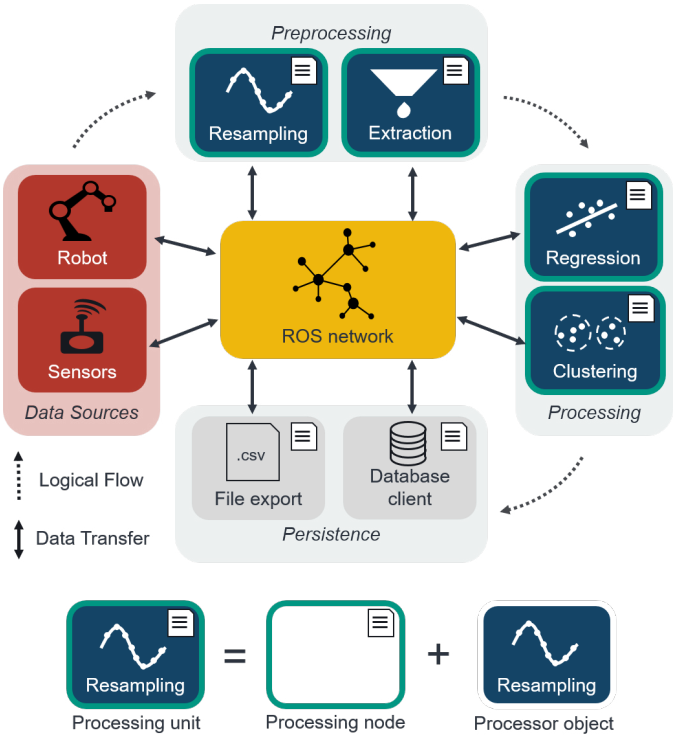

Fig. 1. Proposed Pipeline. The functionality implemented within the *processor object* depends on the use case, communication functionality is provided by the *processing node*. Own illustration based on [9].

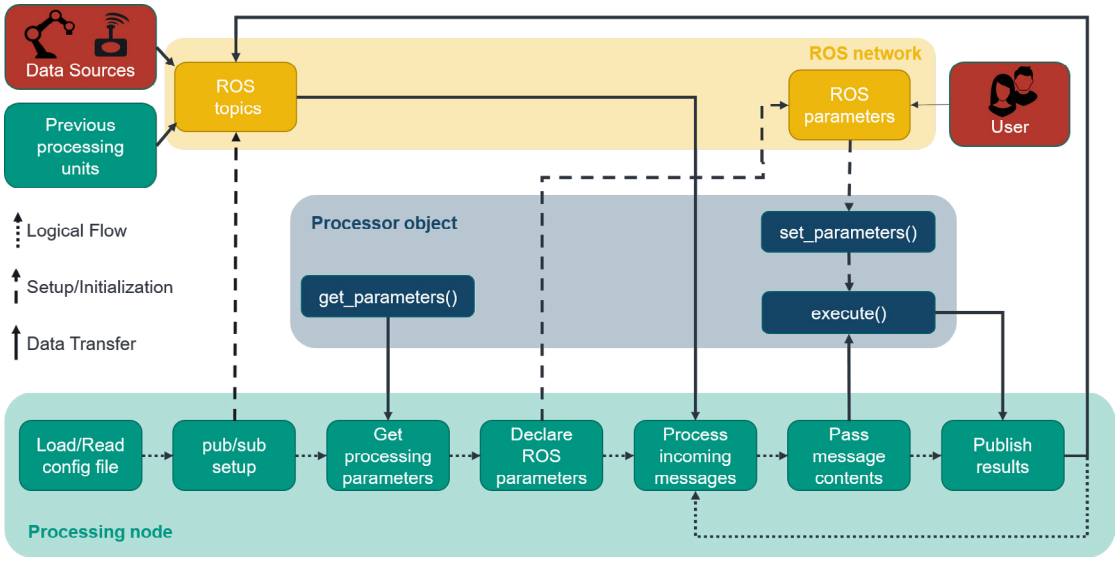

Fig. 2. Setup process and data path between the *processing node*, the *processor object*, and the ROS network.

first stage of development, it is assumed that all data sources, i.e. sensors and robots, are already integrated into a ROS network and no further integration is needed. The general structure of the pipeline is shown in Fig. 1.

The individual processing and persistence units are implemented as nodes within the context of ROS. This allows them to be executed independently of and in parallel to other nodes from any machine within the same ROS network. This network allows communication between any pair of individual units and is therefore able to handle all communication. A single data adaptation node running on a lightweight embedded system close to the monitored machine with the actual processing units running on a separate computer is as feasible a use case as all functionality being executed on an edge device with no external communication. Both cases are supported by our pipeline and functionality developed with one in mind can be used for the other without additional effort.

3.2. Detailed view of the individual units

This subsection aims to provide an overview of the functionality implemented within the separate nodes used for data (pre-)processing and data persistence. A specific unit for data visualization is not deemed necessary, since ROS specific tools like *rqt* allow for extensive visualization options.

3.2.1. Data processing

Modularity for the *processing node* is achieved by packaging all actual processing functionality into an exchangeable *processor object* that is defined at node creation. This object is expected to have an *execute* function, which is called periodically with a defined frequency. The gathered input data is transferred to this function and the output published backed into the ROS network, where it can be visualized or processed further. This structure allows using the concept of the *processing unit* for all tasks that fall within the preprocessing and processing blocks.

The *processor object* allows the definition of parameters. These are automatically linked to newly created ROS parameters, which enables the control of the unit functionality at runtime. For an example unit that provides resampling functionality, one possible parameter would be the sampling frequency. This allows dynamic changes to the provided output and a quick adaptation to new use cases. The setup process and data path between the *processing node*, the *processor object* and the ROS network are shown in Fig. 2.

The *processing node* uses publisher-subscriber patterns to transmit data, due to most sensory equipment providing data in this format. Additionally, this enables easy scalability by allowing multiple subscribers or publishers on the same topic. For example, if multiple different features need to be extracted from the raw source data, multiple *processing units* implementing separate extraction functionalities can be developed, improved and implemented independently, allowing a clear separation of tasks.

Generalizability is achieved by a dynamic creation of publishers and subscribers according to a config file. The structure of this config file for one of the encoder nodes in the exemplary use case (see section 4) is shown in Fig. 3. All inputs necessary for executing the processing function are defined under the *Inputs* header. For each, the ROS topic publishing the data is defined in the “Topic” field. ROS messages consist of multiple fields carrying separate pieces of data. They can also carry other ROS messages, generating a structure with arbitrary depth. The specific field the data relevant to the *processing unit* at hand can be found is specified under “Field”. To ensure that the message types of the topics can be handled correctly, they have to be imported at runtime. For this, the *Imports* header defines the packages and modules that need to be imported. The outputs of the processing function are published under the respective topics in the *Outputs* header. They are defined analogous to the inputs.

```
Inputs:
  Effort0:
    Description: "Effort0"
    Topic: "effort_split_lp"
    Field: ["effort0"]
    MessageType: "SplitEfforts"
Outputs:
  Loss:
    Description: "Encoder Loss"
    Topic: "encoder_loss_0"
    Field: ["loss"]
    MessageType: "EncoderLoss"
Imports:
  SplitEfforts:
    Package: "ros_ml_messages.msg"
    Module: "SplitEfforts"
  EncoderLoss:
    Package: "ros_ml_messages.msg"
    Module: "EncoderLoss"
```

Fig. 3. Exemplary config structure.

If already developed units are to be deployed into a differing environment with different communication standards (topic names, topic types) the effort of deployment is reduced to the adaptation of the unit individual config files. Optional descriptions for each input and output can help identify the appropriate data sources in scenarios where the user of the *processing unit* differs from the developer.

3.2.2. Data persistence

ROS-native tools like *rosbag* can be used to intuitively record and save messages published on relevant topics. Since the recorded *bags* are large in size and not suitable for training models, our pipeline provides a *recording node* that operates on the same config as the *processing node*. It does not process or publish data; it records the topics under *Inputs* and saves the relevant fields into a csv-file. This file can then be used for data analysis or model training. This naturally also works on existing *rosbags*. Additionally, nodes can be set up to record the topics into a time series database such as InfluxDB.

4. Exemplary Use Case

As the previous chapter described the design and structure of our pipeline, this section presents an exemplary practical use case that implements and utilizes it. The goal of this is to show the ease of the deployment process offered by our approach.

The presented use case concerns the monitoring of a handling robot during a reassembly task. The robot used is a Universal Robots UR5e (Datasheet available at [17]), the assembly group in question consists of a 3D printed replica of an electric hairpin motor. The setup is shown in Fig. 4 (a). The task for the robot is to take the lid of the motor and place it on the table they both stand on before picking it back up and placing it back. To avoid crashes and generate movement commands, the process is run in parallel on a digital twin (shown in Fig. 4 (b)) and movement commands as well as gripper actuation are received via a ROS2 interface and a publisher/subscriber and client/service setup, respectively. More information about the planing of the disassembly and the generation of the commands can be found in [18].

For monitoring and control purposes, the ROS2 interface publishes various data about the robot into the network. This includes but is not limited to the position, velocity and torque (called “effort” for generalizability reasons) for each joint. To detect deviations and errors during the reassembly we use torque data recorded during error-free runs to train an autoencoder for each joint. To record the data in the first place, the *recording node* mentioned in subsubsection 3.2.2 is used. The csv-file it exports contains the torque values for each joint for each sampling point during the reference runs (the *joint_states* topic the torque values are sent on is published with a frequency of 500 Hertz). Using this data, an autoencoder specific to each joint is trained. The structure of the encoders is identical; they have 50 input and 50 output neurons and reduce the data down to four neurons at the minimum.

To deploy the autoencoders, a pipeline of multiple different units is set up (shown in Fig. 5). The joint effort values are low-pass filtered and passed to the individual encoders. The calculated encoder loss values (that is, the mean squared error between the real signal and the signal reconstructed by the autoencoder) are then low-pass filtered again before checking if any of them are above a threshold. The low-passes are introduced to reduce noise and ensure that only real anomalies are detected. All individual processing steps are implemented using separate *processing units*, and parameters like the low-pass cut-off frequencies and the threshold value can be changed dynamically. In addition to the output of the threshold unit, tools like *rqt* can be used to monitor every data transfer between units.

To test both the pipeline and the developed autoencoders, the reassembly process is deliberately obstructed in multiple different ways. The resulting maximum and average encoder losses for Joints 5, 4, and 0 are shown in Table 1. The induced anomalies and their effect on the encoder losses are discussed in the following.

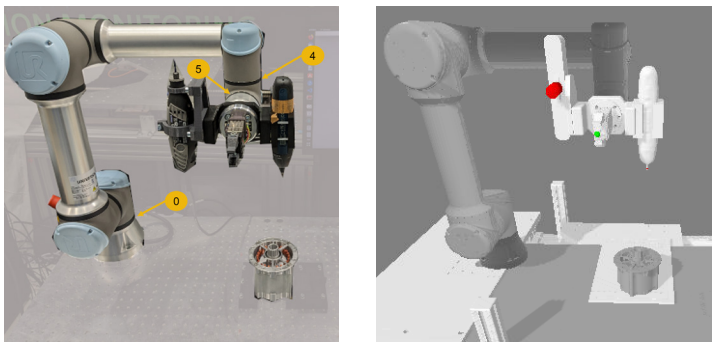

Fig. 4. (a) Experimental setup and robot joint numbers; (b) Simulated environment

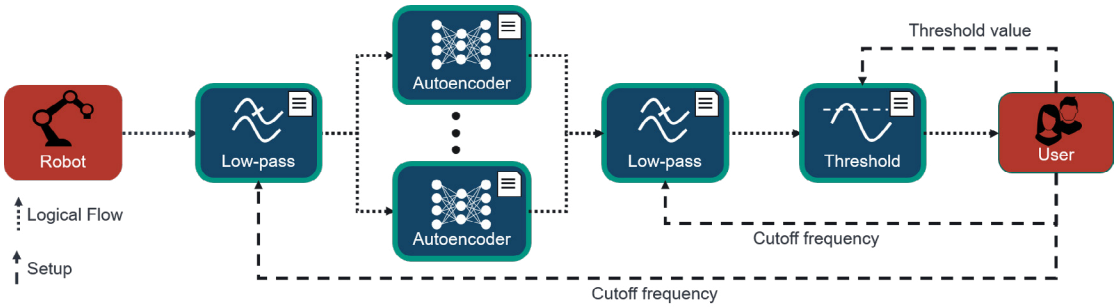

Fig. 5. Logical dataflow for the exemplary use case.

With no obstruction to the task, the maximum losses for joints 5 and 4 are nearly twice the maximum loss for joint 0, while the average losses are close to identical. A possible explanation is the interaction with the lid. Because the movement commands originate from a simulation not perfectly aligned to reality, the robot presses the lid lightly into the motor when replacing it, resulting in a momentarily higher torque.

For the first obstruction, the operator pushes manually against the robot's shoulder joint (joint 0) and subsequently blocks its movement as it tries to move the lid to the side. This results in a maximum encoder loss of around two times the unobstructed value.

For a second scenario, the lid of the motor is held in place as the robot tries to remove it from the housing. This results in the robot not being able to grip the lid, continuing the rest of the program with an empty gripper. Ultimately, this results in the robot crashing into the lid as it tries to place the lid it should be carrying onto the housing. This leads to a maximum encoder loss of nearly five times the unobstructed value for joints 4 and 5, with joint 0 not being affected.

Table 1. **Maximum** (and average) encoder loss for joints 5, 4, and 0 during each scenario.

Scenario	Joint 5	Joint 4	Joint 0
No obstruction	**0.36** (0.05)	**0.31** (0.06)	**0.17** (0.06)
Blocked shoulder	**0.38** (0.06)	**0.34** (0.07)	**0.33** (0.08)
Blocked lid	**1.52** (0.10)	**1.48** (0.11)	**0.16** (0.05)

The obstructions lead to significant increases in the encoder losses for both cases, but the magnitude of the impact differs. This could be addressed by using different threshold values for each joint, requiring only a slight modification to the threshold units *processing object*.

5. Summary and Outlook

In this paper, a pipeline for the efficient deployment of ML models in industrial environments is presented. It achieves high modularity and ease of use by offering an automated setup of communication channels based on intuitively structured config files. It is generally applicable to a variety of use cases by not restricting the form, structure, amount, or frequency of data processing. While the development and training of new models still require development effort, the proposed pipeline helps to get both new models into existing environments and existing models into new environments.

Future work will expand the test domain to non ROS-native environments like traditional machine tools. For that, a data adaption node that reads data from the communication protocol currently at use and publishes that data to the ROS network is needed. Additionally, the pipeline will be used to deploy to more complex use cases to key out limitations and potentials for improvement.

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] A. Dogan, D. Birant, Machine learning and data mining in manufacturing, Expert Systems with Applications 166 (2021) 114060. doi:10.1016/j.eswa.2020.114060.

[2] X. Wang, M. Liu, C. Liu, L. Ling, X. Zhang, Data-driven and Knowledge-based predictive maintenance method for industrial robots for the production stability of intelligent manufacturing, Expert Systems with Applications 234 (2023) 121136. doi:10.1016/j.eswa.2023.121136.

[3] G. Fragapane, D. Ivanov, M. Peron, F. Sgarbossa, J. O. Strandhagen, Increasing flexibility and productivity in Industry 4.0 production networks with autonomous mobile robots and smart intralogistics, Annals of Operations Research 308 (1) (2022) 125–143. doi:10.1007/s10479-020-03526-7.

[4] T. Wuest, D. Weimer, C. Irgens, K.-D. Thoben, Machine learning in manufacturing: advantages, challenges, and applications, Production & Manufacturing Research 4 (1) (2016) 23–45. doi:10.1080/21693277.2016.1192517.

[5] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch, H. H. Olsson, Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions, Information and Software Technology 127 (2020) 106368. doi:10.1016/j.infsof.2020.106368.

[6] A. Dearle, Software Deployment, Past, Present and Future, in: Future of Software Engineering (FOSE '07), IEEE, Minneapolis, MN, 2007, pp. 269–284. doi:10.1109/FOSE.2007.20.

[7] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. Van Der Hoek, A. L. Wolf, A Characterization Framework for Software Deployment Technologies:, Tech. rep., Defense Technical Information Center, Fort Belvoir, VA (Apr. 1998). doi:10.21236/ADA452086.

[8] A. Posoldova, Machine Learning Pipelines: From Research to Production, IEEE Potentials 39 (6) (2020) 38–42, conference Name: IEEE Potentials. doi:10.1109/MPOT.2020.3016280.

[9] I. Heider, H. Yu, N. Krischke, B. Wirth, A. Puchta, J. Fleischer, KI-Einsatz in KMU: Einstiegshürden ausräumen [Clearing entry hurdles for AI deployment in SMEs – Artificial intelligence for German SMEs], wt Werkstattstechnik online 113 (07-08) (2023) 282. doi:10.37544/1436-4980-2023-07-08-16.

[10] I. Malavolta, G. Lewis, B. Schmerl, P. Lago, D. Garlan, How do you architect your robots?: state of the practice and guidelines for ROS-based systems, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice, ACM, Seoul South Korea, 2020, pp. 31–40. doi:10.1145/3377813.3381358.

[11] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot Operating System 2: Design, architecture, and uses in the wild, Science Robotics 7 (66) (2022) eabm6074, publisher: American Association for the Advancement of Science. doi:10.1126/scirobotics.abm6074.

[12] Kubeflow, Kubeflow Pipelines Documentation, online. Accessed 26-Feb-2024 (Apr. 2020).
URL https://www.kubeflow.org/docs/components/pipelines/

[13] R. Urbanowicz, R. Zhang, Y. Cui, P. Suri, STREAMLINE: A Simple, Transparent, End-To-End Automated Machine Learning Pipeline Facilitating Data Analysis and Algorithm Comparison, in: L. Trujillo, S. M. Winkler, S. Silva, W. Banzhaf (Eds.), Genetic Programming Theory and Practice XIX, Genetic and Evolutionary Computation, Springer Nature, Singapore, 2023, pp. 201–231.

[14] D. Kreuzberger, N. Kühl, S. Hirschl, Machine Learning Operations (MLOps): Overview, Definition, and Architecture, IEEE Access 11 (2023) 31866–31879. doi:10.1109/ACCESS.2023.3262138.

[15] M. G. Sarwar Murshed, J. J. Carroll, N. Khan, F. Hussain, Efficient Deployment of Deep Learning Models on Autonomous Robots in the ROS Environment, in: M. A. Wani, B. Raj, F. Luo, D. Dou (Eds.), Deep Learning Applications, Volume 3, Advances in Intelligent Systems and Computing, Springer, Singapore, 2022, pp. 215–243. doi:10.1007/978-981-16-3357-7_9.

[16] Y. Mohamed, S. Lemaignan, ROS for Human-Robot Interaction, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 3020–3027, iSSN: 2153-0866. doi:10.1109/IROS51168.2021.9636816.

[17] Universal Robots, UR5e Datasheet, online. Accessed 13-March-2024 (2023).
URL https://www.universal-robots.com/media/1807465/ur5e-rgb-fact-sheet-landscape-a4.pdf

[18] M. Hansjosten, J. Fleischer, Disassembly Graph Generation and Sequence Planning Based on 3D Models for the Disassembly of Electric Motors, in: T. Bauernhansl, A. Verl, M. Liewald, H.-C. Möhring (Eds.), Production at the Leading Edge of Technology, Springer Nature Switzerland, Cham, 2024, pp. 448–457. doi:10.1007/978-3-031-47394-4_44.