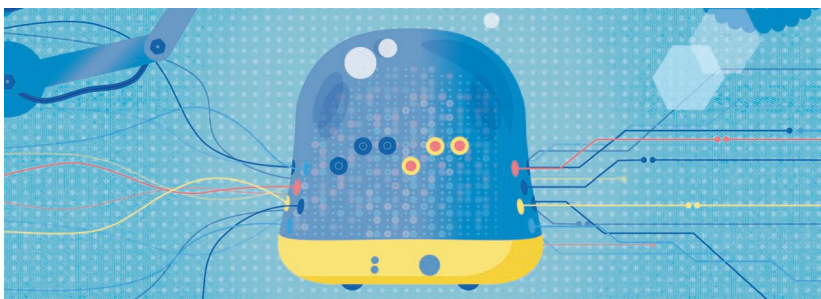


MLOps for Developing Machine-Learning-Enhanced Automotive Applications

Experience at the Bosch Cross-Domain Computing Solutions Division

André L. Ferreira¹, Bosch Car Multimedia S.A. and Universidade do Minho

João M. Fernandes², Universidade do Minho



// We highlight the challenges faced by a development unit at the Solutions Division at Bosch Car Multimedia to establish development of a machine-learning-enhanced systems (MLOps) process, providing lessons learned and takeaways to the community based on an internal validation study. //

AT THE CROSS-DOMAIN Computing Solutions Division (XC) at Bosch Car Multimedia S.A., we design and produce many software-intensive systems and components for the automotive industry. These systems increasingly use AI, more specifically deep learning (DL) and machine learning (ML) algorithms. These DL/ML-enhanced systems are contributing to the development of advanced driver assistance systems that are incorporated into the vehicles to improve the safety of drivers, passengers, and road users. Also, these technologies are being used for developing proof-of-concept systems that monitor vehicle usage over their lifetime. Examples are as follows:

- scratch detection in the surface of the cars, based on audio data
- detection and classification of the occurrence of impact events that generate small damages to the vehicle structure

Digital Object Identifier 10.1109/MS.2024.3415750

Date of publication 21 June 2024; date of current version 11 December 2024.

- acoustic anomaly detection, based on audio data, that triggers an event when abnormal sounds are detected inside the vehicle¹
- violence detection inside the car, namely violence actions among passengers inside a vehicle cabin.²

The main challenge faced when developing ML-enhanced systems is not only to develop the best possible AI models, but also to define a development process and infrastructure that enable experimenting, evolving, and delivering the models over the entire development lifecycle.³ Software engineers face significant difficulties whenever ML components are added to software systems. These problems include the need to manage the experimental and iterative nature of the development of ML models and the integration of these models into software systems in an economical way.

The application of DevOps methods to ML-based solutions and software system development has become more widespread.⁴ However, there are not many findings in the scientific literature about development and deployment of ML-enhanced systems (MLOps) and its effective implementation. This article shares our experience in engineering ML-enhanced systems for the automotive industry and draws attention to essential success elements that have shown their value. We provide software engineers, ML engineers, and development teams with timely and engaging inputs from the field.

Challenges When Developing Automotive ML-Enhanced Systems

AI permits computers to perform complex tasks that we associate with

humans, such as predicting, diagnosing, planning, and recognizing.⁵ AI describes methods that let computers simulate human decision-making to complete challenging tasks. Currently, many AI systems are based on ML techniques.

An ML-enhanced system is a software-intensive system that includes data and components that implement algorithms mimicking learning and problem-solving. Those systems have different characteristics than traditional software systems.⁶ The functionalities of an ML-enhanced system are enabled by ML components (e.g., for autonomous driving). Typically, the ML models are components that do not work alone and that need to be smoothly incorporated into the architecture of the system. This creates new challenges that software engineers must address.

Traditional software systems execute a set of instructions that were predefined by the programmers, while an ML-enhanced system makes decisions based on past data and probabilistic outcomes. Thus, the specific characteristics of ML require the traditional software approaches to be adapted, so that one can efficiently build, maintain, and evolve those systems. In particular, DevOps, a popular approach in industry, needs to be extended into DataOps or MLOps to cope with the specific requirements of ML-enhanced solutions.⁵

The creation of ML-enhanced systems differs from other kinds of systems in that they are strongly dependent on data that are used to develop the corresponding ML models. Often, it is challenging to predict at the outset of a project if a suitable cost-effective solution will be eventually achieved.

Another way that ML development differs from traditional software development is that designing and programming code is not the main focus. Usually, an appropriate ML algorithm or model for a given task or problem is not found on the first attempt. ML development requires an iterative process due to its experimental nature.⁷

To industrialize automotive ML-based systems in a cost-effective way, a reliable and systematic process that offers support for the full lifecycle of these systems is needed. As for any ML-based system development, dealing with data and finding strategies to increase data management efficiency become mandatory.

In the automotive industry, novel ML systems often rely on equipment and sensors that are part of the vehicles. In such cases, the binding of these ML-based systems to the vehicle hardware is strong and cannot be neglected. The diversity of sensors and vehicle components and their placement in the vehicle are constraints when developing the ML model.

The iterative nature of model development is an intrinsic ingredient of ML. Adopting a data-centered approach to development means looking for the best algorithm/model by maximizing the amount of data that is currently accessible and, preferably, expanding it. This also applies in the automotive industry, where efficient data management is crucial.

Specific challenges arise when deploying ML models in the automotive context, as delivering those models to vehicles may not be as straightforward as in other application domains. Not only must software be installed in vehicles, but effective management of software updates must also be guaranteed.

MLOps in the Automotive Industry

ML capabilities may be added to a system in several ways, including software systems with ML components and ML frameworks, tools, and libraries that provide ML functionalities. A common pattern has

increasingly relevant to ensure specifications are fulfilled in a cost-effective manner. In ML, iterative development is the norm and is stressed even more. Many iterations of development are needed for algorithms/models and automation in ML becomes more relevant when compared to traditional

The goal of MLOps is to create highly automated development pipelines that promote efficiency, reproducibility, and traceability in development by integrating ML and data engineering into the DevOps landscape.

been observed: While creating and implementing ML systems may be comparatively quick and inexpensive, maintaining them over time is challenging and costly because of technical debt. Challenges when developing traditional software systems still exist when developing ML-enhanced systems, in addition to a new set of issues unique to ML.⁸

The creation of ML systems and their rapid introduction into production are the ultimate objective of all industrial ML development initiatives. However, it is still difficult for software professionals to standardize and operationalize the software process for ML systems. Many development endeavors fall short of expectations, specifically when they start to be overly costly to maintain.

To tackle the uncertainties associated with the iterative nature of software development, automation in software quality assurance is

software development. Poor performance and extremely high development costs can be the consequences of not having a simplified procedure that fosters experimentation.

MLOps offers concepts, best practices, and a suitable development culture to create cost-effective, high-quality, and reliable ML systems. The goal of MLOps is to create highly automated development pipelines that promote efficiency, reproducibility, and traceability in development by integrating ML and data engineering into the DevOps landscape.⁹

Reproducibility and traceability of experiments in developing ML models are most desired in the automotive context. The capacity to reproduce an ML model is not only desirable from a process quality perspective, but also fundamental to enable a systematic approach to improve that model.

MLOps stands for the collection of techniques and tools for the deployment of ML models in production. MLOps automates ML pipelines by applying DevOps practices. Its core principle is to automate and monitor every stage of the ML pipeline. This creates a development environment that allows for workflow optimization and the avoidance of implementation and deployment errors and mistakes.¹⁰ MLOps is defined as a paradigm, including aspects like best practices, concepts, as well as a development culture when it comes to the end-to-end conceptualization, implementation, monitoring, deployment, and scalability of ML products.¹¹

Establishing (and formalizing) an MLOps process is not straightforward, as it entails many challenges,¹² which are discussed next.

To build ML models, one needs to use data with high quality, as they have a considerable impact on the quality of the results. It is also important to consider explainability whenever delivering an ML system that must be trusted. Explainability is considered the first step toward creating a sustainable and reliable MLOps platform.

There is a need to experiment faster and deploy ML solutions quickly. This is only possible with an automated ML pipeline because it reduces the experimentation time. Many companies already have an established process and best practices for automatically developing ML models, but they still struggle in migrating from monolithic ML programs to component-oriented ML pipelines. An MLOps process must be formally represented to allow its automation. Warnett and Zdun report that, when compared with informal textual descriptions and informal graphical representations, the use of semiformal diagrams for describing

MLOps systems improves understandability and does not significantly increase task duration (and thus hinder understanding).¹³

To compare deterioration on the results of ML models in production, it is critical to comprehend performance and metrics for ML systems. Observability focuses on checking data integrity, identifying shifts in data distribution, like drifts in ML classifiers, and using nonparametric techniques to assess the distribution of model prediction confidence on the changes.

An MLOps workflow typically follows seven major steps¹⁰:

1. *data extraction* to integrate relevant data from various sources
2. *data analysis* to understand data from the datasets
3. *data cleaning, transformation, and feature engineering* to split data into training, validation, and test sets having appropriate formats
4. *model training* to train ML models and save the best performing trained model starting from different algorithms and parameter settings
5. *model validation* to interactively evaluate the model quality on the test data and identify whether it satisfies the quality criteria based on performance metrics
6. *model serving* to deploy models in the target environments integrated with other software components
7. *model monitoring* to detect model degradation through usage, input data, and performance analyses.

The extent to which a given company is able to automate the seven-step MLOps workflow is related to its

MLOps maturity level. According to John et al.,¹⁴ software companies go through four stages as they systematically adopt MLOps practices: 1) automated data collection (steps 1–3), 2) automated model deployment (steps 1–6), 3) semiautomated model monitoring (steps 1–7), and 4) fully automated model monitoring (steps 1–7). These stages capture key transition points in the adoption of MLOps in practice and, with respect to the last two levels, a different automation degree.

Automotive MLOps at the XC Division

Major Characteristics

At XC, we develop ML-enhanced systems to be integrated into automobiles. The trigger for our team to establish an MLOps framework was a project to develop a system to monitor impacts on the exterior of vehicles.¹ In this system, sensors are retrofitted to general-purpose passenger vehicles to introduce the perception capability of monitoring and detecting exterior impacts to the vehicle. The end goal is to alert whenever these impacts may be linked to a resulting structural damage to the vehicle. As an ML-enhanced system, the functionality was developed based on sensory data, namely inertial sensors and microphones. The intrinsic challenge of the feature itself is to cope with the structural differences of the vehicle and how impacts are manifested throughout the vehicle structure before reaching the sensors.

New data, related to events that occur while vehicles are being conducted, are regularly obtained. The acquired data are used to train the ML models by defining training and testing datasets. As the system scaled to many different brands and models of vehicles, the volume and variety of the data rapidly increased.

Additionally, driven by the need to find an ML model to serve an increasing vehicle landscape, the ability to iterate development, namely model training and validation, was stressed. Experimenting with multiple configurations of training and testing datasets, alongside with model parameterization, became challenging and cumbersome to track over time. These two aspects stressed the necessity to establish an MLOps framework to: 1) reduce effort and development time in iterating the deployment of ML models to production, and 2) enable efficient management of the raw data obtained from the vehicle sensors.

According to the characterization of the MLOps indicated in the “[MLOps in the Automotive Industry](#)” section, our framework was initially conceptualized to provide infrastructure and tool support to four MLOps steps: data extraction, data analysis, model training, and model validation. The driving objectives of the framework were defined as:

- To enable ML engineers to manage data effectively and efficiently when iterating ML model development.
- To automate the retraining of existing ML models based on the arrival of new data and, consequently, to generate new ML models.
- To automate the benchmarking of these new ML models against multiple testing sets that are maintained over time.

Overall, the anticipated benefits expected are threefold. First, we expected to decrease the development time, namely the cycle time for generating new ML models based on new data. Second, we intended to reduce development cost, mainly the engineering effort associated with

managing the influx of new data. Finally, the customer value increases since the ML models perform better. This performance improvement is possible because we can perform a higher number of iterations per unit of development time.

As depicted in Figure 1, the MLOps process that we have established is conceptualized according to three

phases (data ingestion and management, model development, and model benchmarking), which are described in the next three sections.

Data Ingestion and Management

The aim of the first phase of our MLOps framework (blue area in Figure 1) is to manage data. It follows three major steps.

Ingest Data. The framework enables the ingestion of data that originate either from data acquisition activities performed by the team or from the productive environment, i.e., directly from vehicles. Raw data are organized in binary files. Each of these binary files has a corresponding metadata file with the respective labeling information. These

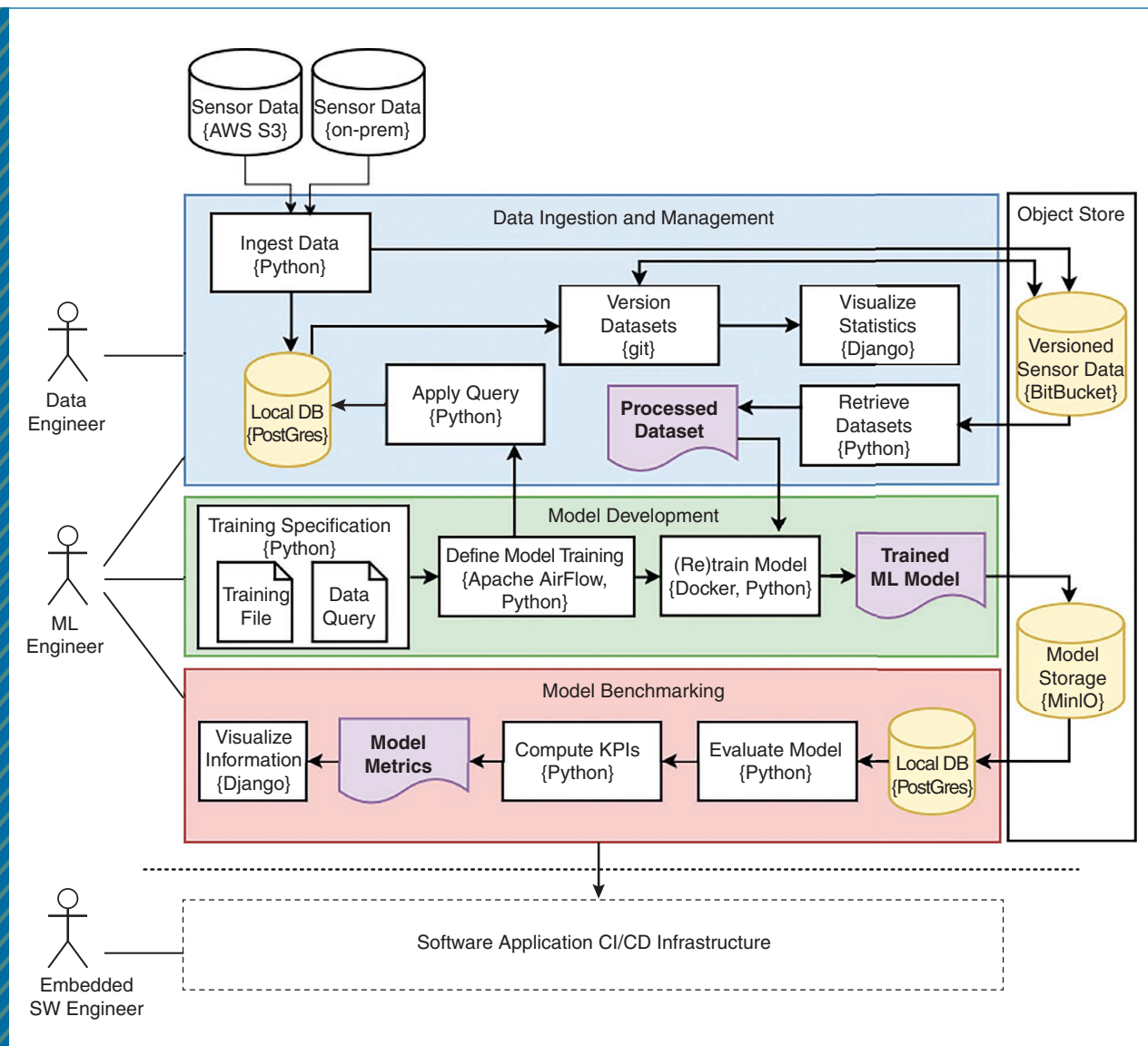


FIGURE 1. The MLOps process workflow. DB: database; KPI: key performance metric.

data have an initial staging phase in a cloud-based environment if they originate from the productive environment. The data are then pulled into an on-premise compute/storage infrastructure via an extract, transform, and load batch-based process.

Version Datasets. The MLOps framework enables the creation of dataset snapshots. For that, it generates a unique dataset ID, alongside a given tag name, for a dataset resulting from a search query using metadata information. A search query is translated into a list of raw data files that satisfies the metadata criteria in the query. This unique ID represents an immutable set of files allowing the ML engineer to retrieve this exact dataset, composed of a set of raw data files and associated metadata files, any time in the future.

Ingestion and versioning of datasets are possible by a framework component that also tracks changes to metadata information over time, e.g., correcting labels implies updating the metadata file information. Bitbucket, is used as version control system to keep track of these changes.

Retrieve Datasets. By having a unique dataset ID or tag name, ML engineers can retrieve any dataset from the repository (pair of raw data and associated metadata files), which ensures consistency among ML developers on dataset usage. This feature greatly improves how ML engineers perform iterations in ML development and ensures consistency in the use of different datasets in training.

A framework component for the retrieval of datasets was implemented in a web-based data service application programming interface (API) using Django. Having it accessible via an API greatly facilitates access to

data and enables the integration of applications to retrieve data from the data repository. From a technological point of view, the storage of raw data uses MinIO as an object storage solution that facilitates integration with other subsystems.

Model Development

For the second phase of development (green area in Figure 1), our framework aims to deliver automation capabilities to retrain ML models. The goal is to allow the ML engineer to define a specification for training an ML model with a specific dataset and have a continuous loop of retraining based on the arrival of new relevant data into the repository. This phase follows three major steps.

Define Model Training. Model training specifications serve as a base for reproducing ML models. A model training specification is defined by a configuration of how an ML algorithm is specified with associated hyperparameters alongside the search query of the training dataset. These configurations are stored and marked as active or inactive. Active configurations trigger retraining every time new data are added to the repository that satisfy the training set criteria. This step allows the ML engineers to define and manage several model training specifications over time. They can have different specifications of a given algorithm, making it easier to implement a more model-centric development approach. For developing the impact detector on the exterior of vehicles, a decision tree ML algorithm was selected, for which several configurations of hyperparameters are maintained that allow the team to explore solution spaces for better-performing models. The most relevant metric monitored is the false-positive

rate, which translates to triggering impact events that are generating vehicle structural damage. Fine-tuning the optimal tradeoff between false-positive and true-positive rates is the driving exercise of the training step.

(Re)train Models. Whenever new data are available in the system, the ML models can be retrained. We detect that new ingested data are available when they meet the search label criteria associated with a training specification. In this case, the newly updated training set is retrieved (via the retrieval component of the framework) and the associated ML model is retrained. The ML engineers can decide which configurations should be retrained.

A framework component for the second phase is now responsible for operating these model training specifications. In practice, a training specification is added by the ML engineer via a web app for that purpose. Training specifications are stored and accessed by a Python script that is responsible for generating a training procedure. A training procedure is based on a Docker image file containing the system runtime configuration (e.g., Python and associated imports) to perform the training. Additionally, information to retrieve the training and testing datasets is also present. The Docker image is created to execute any number of training runs for a given configuration. The actual training is managed by an instance of the Apache Airflow via a configuration script that links the Docker file to a directed acyclic graph configuration, specifying the necessary training steps (e.g., fetching datasets, triggering docker, and storage of training results).

Store New Models. Finally, it is necessary to manage the storage of new models



ANDRÉ L. FERREIRA is R&D AI solutions engineer at Bosch Car Multimedia Portugal S.A, 4705-820 Braga, Portugal and an invited assistant professor at the Departamento de Informática, Universidade do Minho, 4710-057 Braga, Portugal. His research interests include the development of sensor-based perception systems using AI and software development methodologies. Ferreira received his Ph.D. in software engineering from Universidade do Minho. Contact him at andre.ferreira2@pt.bosch.com.



JOÃO M. FERNANDES is a full professor at the Departamento de Informática, Universidade do Minho, 4710-057 Braga, Portugal. His research interests include software modeling, requirements engineering, software business, and bibliometrics. Fernandes received his Ph.D. in computer engineering from Universidade do Minho. Contact him at jmf@di.uminho.pt.

obtained after retraining. The generated models are stored alongside the test dataset and context information in a predefined storage file location. A model information management solution was developed using a relational database and object store for storing the models. This centralized information is used in the next phase of development.

Model Benchmarking

For the third phase of development (red area in [Figure 1](#)), the framework aims to deliver a model benchmark component that is also conceptualized according to three major steps.

Evaluate Models. After retraining, the teams need to evaluate the performance of the model by running it against different testing sets, each representing possible representations of reality.

The ML engineer defines testing sets and performance metrics

applicable to each model coming from the retraining phase. This is made possible by following the first stage of our framework, which allows the creation of data snapshots. The use of dataset IDs facilitates the definition and use of these test sets. Running the model inference for benchmarking is possible by runtime specification standardization, meaning that there is limited flexibility in which model benchmarking can be automated.

Compute Key Performance Indicators. Performance metrics are defined for each ML model being tested. After tests are performed, performance metrics are calculated automatically. In this step, the ML engineers need to define and provide code snippets of performance metrics associated with each of the ML models under consideration. The result is computed automatically and is stored in a database for later evaluation and comparisons with previous

models. Alongside performance metrics, Shapley additive explanation values are computed to provide additional information.

Visualize Information. Based on the performance metrics available, the ML teams need to check how those metrics compare to previous versions of the ML model. This information should be visually available to allow teams to easily decide whether performance has improved or not and how much has changed.

Our framework provides an application with a web interface developed in Django that displays relevant information on the performance of each ML model in the continuous loop of development. This tool facilitates analysis and decision-making as performance reporting is readily available right after each iteration of development.

Model Deployment and Monitoring

The decision to release a model into production is then taken based on the performance information. The model output is materialized into a file of model weights pushed into the software build process. This step is not yet in the scope of the automation mechanisms provided by the MLOps framework. The file is delivered via a commit into a continuous integration/continuous delivery pipeline for the final software build.

Monitoring model performance in the field is currently performed via the support of specific tools that are not directly linked to the pipeline. There is a partially manual step in analyzing and deciding which data events are arriving from the field. Selected data are made available to be added to training or validation sets for further model training.

We summarize the following key takeaways from our experience in automating MLOps at XC. Achieving a certain level of automation is highly desirable in the process of searching for the best-performing ML algorithms/models. It enables fast iteration with less effort and increases the chances of achieving a higher-performing ML model. Nevertheless, automation requires upfront investment and several factors may play a decisive role depending on the circumstances.

In this context, the determining factor was how fast a new model is desirable to be released to production. New data coming from the field are contributing highly to improving model performance. Evidences of true positives and false positives greatly influence the performance of the ML model, promoting the need to automate training when new data arrive. This motivated automation at the model training and benchmarking steps.

Automation of data management is desired and necessary whenever the speed and the volume of data are high. In this case, data inflow daily in large volumes. Having in place data management automation is key to coping with this inflow, otherwise development efforts would be prohibitively high. The desire to evolve the pipeline to include automation in model deployment and monitoring is also high at XC. The possibility to decide and issue an update over the air to vehicles in the fleet is the next step of the framework development roadmap. This feature will decrease the cycle time for delivering a new ML model into the field. 📄

Acknowledgment

This work has been supported by Fundação para a Ciência e Tecnologia

within the R&D Units Project Scope UIDB/00319/2020.

References

1. G. Coelho, L. M. Matos, P. J. Pereira, A. Ferreira, A. Pilastrri, and P. Cortez, "Deep autoencoders for acoustic anomaly detection: Experiments with working machine and in-vehicle audio," *Neural Comput. Appl.*, vol. 34, no. 22, pp. 19,485–19,499, 2022, doi: [10.1007/s00521-022-07375-2](https://doi.org/10.1007/s00521-022-07375-2).
2. D. Nova, A. Ferreira, and P. Cortez, "A machine learning approach to detect violent behaviour from video," in *Proc. 10th Int. Conf. Intell. Technol. Interactive Entertainment (INTETAIN)*, Cham, Switzerland: Springer-Verlag, 2019, pp. 85–94, doi: [10.1007/978-3-030-16447-8_9](https://doi.org/10.1007/978-3-030-16447-8_9).
3. J. Bosch, H. Holmström Olsson, B. Brinne, and I. Crnkovic, "AI engineering: Realizing the potential of AI," *IEEE Softw.*, vol. 39, no. 6, pp. 23–27, Nov./Dec. 2022, doi: [10.1109/MS.2022.3199621](https://doi.org/10.1109/MS.2022.3199621).
4. B. M. A. Matsui and D. H. Goya, "MLOps: Five steps to guide its effective implementation," in *Proc. 1st IEEE/ACM Int. Conf. AI Eng. Softw. Eng. AI (CAIN)*, 2022, pp. 33–34, doi: [10.1145/3522664.3528611](https://doi.org/10.1145/3522664.3528611).
5. A. Sagodi, J. Schniertshauer, and B. van Giffen, "Engineering AI-enabled computer vision systems: Lessons from manufacturing," *IEEE Softw.*, vol. 39, no. 6, pp. 51–57, Nov./Dec. 2022, doi: [10.1109/MS.2022.3189904](https://doi.org/10.1109/MS.2022.3189904).
6. I. Ozkaya, "What is really different in engineering AI-enabled systems?" *IEEE Softw.*, vol. 37, no. 4, pp. 3–6, Jul./Aug. 2020, doi: [10.1109/MS.2020.2993662](https://doi.org/10.1109/MS.2020.2993662).
7. K. Vaidhyanathan, A. Chandran, H. Muccini, and R. Roy, "Agile4MLS—Leveraging agile practices for developing machine learning-enabled systems: An industrial experience," *IEEE Softw.*, vol. 39, no. 6, pp. 43–50, Nov./Dec. 2022, doi: [10.1109/MS.2022.3195432](https://doi.org/10.1109/MS.2022.3195432).
8. Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does machine learning change software development practices?" *IEEE Trans. Softw. Eng.*, vol. 47, no. 9, pp. 1857–1871, Sep. 2021, doi: [10.1109/TSE.2019.2937083](https://doi.org/10.1109/TSE.2019.2937083).
9. M. Borg, "Pipeline infrastructure required to meet the requirements on AI," *IEEE Softw.*, vol. 40, no. 1, pp. 18–22, Jan./Feb. 2023, doi: [10.1109/MS.2022.3211687](https://doi.org/10.1109/MS.2022.3211687).
10. G. Recupito et al., "A multivocal literature review of MLOps tools and features," in *Proc. 48th Euro-micro Conf. Softw. Eng. Adv. Appl. (SEAA)*, 2022, pp. 84–91, doi: [10.1109/SEAA56994.2022.00021](https://doi.org/10.1109/SEAA56994.2022.00021).
11. D. Kreuzberger, N. Köhl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," *IEEE Access*, vol. 11, pp. 31,866–31,879, 2023, doi: [10.1109/ACCESS.2023.3262138](https://doi.org/10.1109/ACCESS.2023.3262138).
12. D. E. Rzig, F. Hassan, and M. Kessentini, "An empirical study on ML DevOps adoption trends, efforts, and benefits analysis," *Inf. Softw. Technol.*, vol. 152, Dec. 2022, Art. no. 107037, doi: [10.1016/j.infsof.2022.107037](https://doi.org/10.1016/j.infsof.2022.107037).
13. S. J. Warnett and U. Zdun, "On the understandability of MLOps system architectures," *IEEE Trans. Softw. Eng.*, vol. 50, no. 5, pp. 1015–1039, May 2024, doi: [10.1109/TSE.2024.3367488](https://doi.org/10.1109/TSE.2024.3367488).
14. M. M. John, H. Holmström Olsson, and J. Bosch, "Towards MLOps: A framework and maturity model," in *Proc. 47th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, 2021, pp. 334–341, doi: [10.1109/SEAA53835.2021.00050](https://doi.org/10.1109/SEAA53835.2021.00050).