A GUIDE FOR MLOPS TOOLS

Finding your way trough the MLOps lifecycle

From the TETRA project MLOps4ECM

Contents

1 Introduction

More and more companies are investing in machine learning (ML), trying to stay ahead of their competition. A prime example is condition monitoring applications, where a machine learning model is implemented to detect and predict faults in running machines, avoiding machine downtime and saving a lot of money. However, the investments of these companies do not always result in the machine learning model getting into production. Often times, the model gets stuck in the R&D environment without being able to transfer successfully to a real-world situation. This can be caused by several factors such as lack of versioning, unrealistic expectations, and the dynamic aspects of production environments. This is where Machine learning operations (MLOps) comes into play. MLOps, which is a porte-manteau of machine learning and DevOps, is a paradigm for creating and maintaining machine learning applications for deployment into production. It works on several principles, inherited by the world of DevOps, augmented with specific machine learning components. In DevOps, the goal is to close the bridge between developers (Dev) and Operations (Ops) people by using several principles, techniques and tools and using a closed-loop development pipeline (Figure 1). The core principles of DevOps are the following:

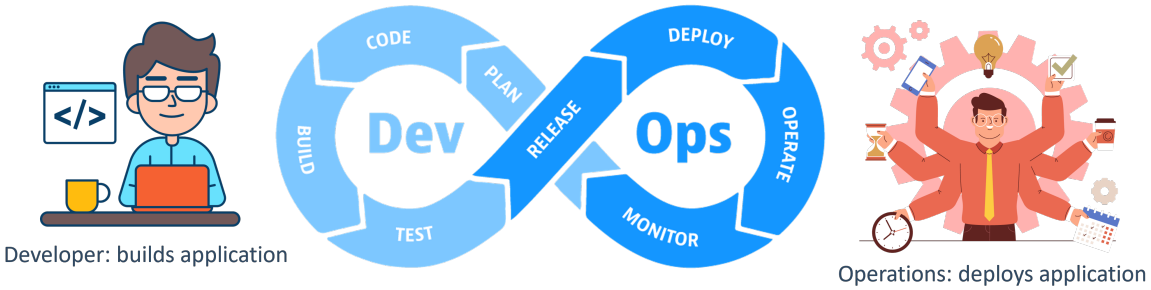

Figure 1: DevOps is the bridge between the developers and the operations

- Collaboration: Starts from the planning phase where everyone involved with the project (including developers, operation people and clients), should be in the room when the application is planned. It also involves constant communication between these three parties which is done by creating short feedback loops.
- Reproducibility: The ability to reproduce working code and faults making problem solving a lot faster. This is done through the use of code versioning, where you keep track of the different code changes over time.
- Continuous improvement: Instead of working on a new feature on a separate branch and only pushing it to the main branch once it is fully done, the developers should work in small tasks, continuously implementing small fixes. This avoids the problem of the code breaking once it is merged and not knowing which small change has caused this problem. Working with these smaller changes, creates short feedback loops which are a lot easier to debug. The Continuous creation, building, and testing of small fixes is called Continuous Integration (CI). This CI is often combined with Continuous Deployment (CD). The full feedback loop is referred to as CI/CD.
- Automation: During the MLOps pipeline there are a lot of steps to take and to make these steps go faster and reduce mistakes, they should be automated. For example, Once a piece of code is ready, the testing, building, and deployment steps should follow automatically. However, it is not always a good idea to blindly trust all new code, therefore there is often a manual approval step necessary before certain steps are activated.
- Monitoring: To be able to catch mistakes in the code or production environment, the entire process should be monitored. This includes creating an overview of the integration and testing, and the status of all aspect of the production environment. Such an overview is created with dashboards which can often alert you when something is wrong.

For machine learning applications, all these same principles still apply but are extended to include machine learning specific artifacts. In the world of DevOps we only have our code base as a changing artifact. In the

world of machine learning, this code is expanded with two other artifacts, the data and the machine learning model itself. These three artifacts all need to be managed in a clear and structured way for the machine learning application to be successful. We therefore extend our core DevOps principles to include the other two artifacts.

- Reproducibility: Next to the versioning of our code base, the changes of the used datasets and machine learning model should also be kept. This does not only include the data and the model itself but also other metrics associated with them, like their accuracy. These metrics are called metadata.
- Continuous improvement: The CI/CD loop stays a very important structure for our code base. There is even a similar improvement for our machine learning model. Like the code, the model should be continuously updated to learn its changing environment. Of course, the model also has to be evaluated each time it is retrained. This cycle of continuous model learning is called CL for short.
- Automation: Like CI/CD, CL should happen in an automatic way. The same is true for the collection and preparation of the data. Both these aspects include several steps which can take a lot of time if done manually. To automate this, pipelines are created for each artifact, data, model and code, which will start each step automatically when triggered by a certain event.
- Monitoring: For machine learning applications both the input data and the model performance should be monitored. This is done to detect when the machine learning model is degrading or when our production environment has changed.

To implement these principles in a clear way, an MLOps pipeline has been created which includes six steps for building a machine learning model ready for production environments. These steps are:

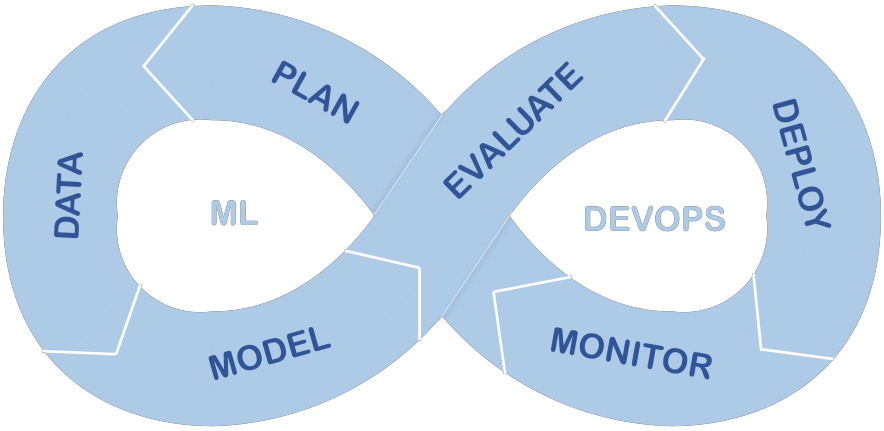

Figure 2: The MLOps pipeline containing the six steps

1. Plan: Together with all involved parties, plan what the machine learning application will look like and which steps need to be taken to get there. This is not only necessary before you start developing but also every time the machine learning application needs to be improved. This creates short feedback loops with the client, developers, and operations, which are needed for good collaboration.
2. Data: Before a machine learning model can be trained, data should be collected. This data needs to be cleaned and preprocessed before it can be used for model training. The changes in the datasets need to be tracked through data versioning.
3. Model: Using the data from the previous step, a machine learning model is trained. This step normally takes multiple iterations with changing hyperparameters to create a good performing model. The different model versions together with their metadata should also be tracked for reproducibility through model versioning or experiment tracking.
4. Evaluate: Once a machine learning model is created, it should be evaluated on realistic production data and in a similar environment as the one in production to know if the new model actually works and performs better than the previous solutions. This should be done automatically such that the created model has a reliable performance.

5. Deploy: This step includes the preparation for production. Here your machine learning model is compressed and changed into a general format to use in the production environment. Your code and machine learning models are also containerized if necessary to be able to use them easily in production. This step also involves actually moving the new code and machine learning model to the production environment and running it there.

6. Monitoring: This steps monitors the machine learning model and its production environment. Once this system has detected a drift in the data or model performance, data from this new drifted environment should be collected and the MLOps loop can be started all over again.

For all these steps there are different tools, both open source and paid, which can improve the MLOps experience. There are hundreds of tools available of which some try to solve a specific task, while others try to create a holistic MLOps solution. To help you through the landscape of MLOps and its tools, this guide will go over each step in the MLOps pipeline, and explain which types of tools there are and give an overview of the most commonly used tools. All tool names are links to either a tutorial of said tool, or their official website.

2 Plan

Before starting to build your application, a planning session with all involved parties should be held. This session includes the developers, operation people and the customer to make sure that everyone is on the same page. Here are some of the most important questions that need to be answered.

- What does the application need to do?
- Which data needs to be collected and what data is available?
- Which type of problem are you trying to solve? (classification, regression, generation, ...)
- Which environment will the model run on?
- What type of constraints are put on the application? (memory, fairness, ...)
- What type of machine learning model will you use?
- What tools do you want to use?

When answering these questions, try to be as clear and concrete as possible. This will help in further steps. As seen in the last question, one of the things you will do is decide on the tools you want to use. Some of these tools you can set up before you start with the next step. These mainly have to do with the automation of the pipeline. There are two main ways to implement automation using tools. The first one is through the use of workflow orchestration platforms. These tools make it possible to create and manage pipelines which implement certain tasks in a certain order. They show you what task ran when and if it was successful or not. Some tools also give you the possibility to schedule your tasks and alert you when something went wrong. Following is a short list of workflow orchestration tools:

- Apache airflow: Apache Airflow is one of the first open-source workflow orchestration platforms, and thus the most mature one. It was developed with a focus on engineering which makes it possible to implement more complex tasks but also makes it more complicated to work with. This makes it not very beginner friendly compared to other workflow orchestration tools.
- Prefect: A Python-based workflow orchestration tool which works with decorators to create workflows and tasks. It has data and code security and a free self-hosted orchestration option next to a paid Cloud solution. One of its main advantages is that it is easy to set up and to implement logging, notifications and retries.
- Dagster: Similar to Prefect, Dagster is a Python-based workflow orchestrator. It is more declaration based which a lot of people prefer. Compared to prefect, Dagster employs a more data-first approach for their pipelining whilst Prefect is a more generic workflow orchestration tool. They offer several subscription services on their website but also have an open-source version which can be used for free.

- Argo Workflows: An open-source container-native workflow orchestration tool, created to easily work with Kubernetes where each step in the workflow is a container. It uses Directed Acyclic Graph (DAG) to create workflows with dependency tracking between tasks. A related project "Argo CD" offers continuous delivery for Kubernetes.

- MLRun: This open-source ML orchestration tool tries to incorporate the whole ML lifecycle process providing data management, model training, evaluation, deployment and monitoring automation. There is an integration with a wide range of other ML tools and plugins next to workflow CI/CD aspects. This is one of the tools trying to provide a more holistic MLOps solution, however they are most well known for their workflow orchestration.

- Flyte: Another open-source workflow orchestration tool built upon Kubernetes to create durable, flexible, and scalable workflows. It has a declarative style resource provisioning and also includes versioning and branching.

Which platform to use depends on which programming languages you use and how you want to define your workflows. If all your machine learning code is written in Python, using Dagster or Prefect is a good idea since they are both python-first orchestration tools. Between the two the main difference is how the workflows are implemented. If you prefer a declarative approach, Dagster might be the better option, otherwise you can use Prefect. Apache Airflow is more mature and offers a lot more third-party integrations, but can be more difficult to work with compared to other tools. Lastly, When working with Kubernetes, using Argo or Flyte might be a good idea.

The second way to implement automation in your workflow is through CI/CD tools. These tools have a similar goal as workflow orchestration tools where they are used to automatically start tasks, schedule them and alert you when a fault has occurred. The biggest difference is that CI/CD tools are often linked with code versioning platforms and focus more on testing and deployment tasks, while workflow orchestration tools have a more general intake, including data and model pipelines. In practice, the difference between these two types of tools can get a bit hazy, and sometimes the use of a single tool can be enough to automate everything. What's next is a list of common CI/CD platforms.

- Gitlab CI/CD: Gitlab is a web-based platform for code versioning. It has added a CI/CD space which creates the possibility to add CI/CD capabilities to your Git repository. Making it possible to automatically build, deploy and test your code during software development. They have an open-source edition of Gitlab CI/CD which makes it possible to self-host on your own servers.

- Github Actions: Like Gitlab CI/CD, Github Actions is an integrated tool in the Github code versioning platform. With Actions, building, testing and deploying your code can be automated at certain times in your developing phase. A common choice is to trigger CI/CD tasks at every merge request to the main branch of your repository.

- Jenkins: Jenkins is a popular open-source automation server designed for CI/CD aspects like automated testing and deployment of your code. It has a pipeline as code feature to make the implementation of automation easier for developers since it can be implemented programmatically. However it mainly works with Java making it a little less suitable for ML applications since they are often written in Python code.

- Travis CI: This web-based CI/CD tool in their own words focuses on a developer first approach. It uses a simple web interface to configure and maintain pipelines and provides a lot of features with support for most popular programming languages. However, Travic CI is not free compared to the other CI/CD tools on this list.

- Azure DevOps: A Microsoft CI/CD tool which is very relevant in industry right now.

The choice of CI/CD tool depends on the code versioning tool you use and how you prefer to implement your CI/CD tasks. If you already work with github or gitlab for your code versioning, it is a logical choice to use their CI/CD tools. If you prefer to work with a web-based configuration, and you do not mind spending some money, it might be a good option to use Travis CI or Azure DevOps.

When the plan for your ML application has been made, and all tools have been decided upon, you can move on to the next step in the MLOps cycle, which is the data step.

3 Data

In this step everything concerning the data used for training your ML model is discussed, starting with data versioning. Throughout the lifetime of the machine learning model, different datasets in different forms will be created. To be able to reproduce certain machine learning model results, it is necessary to keep track of the changes in these datasets. We therefore implement data versioning. When thinking about versioning, the first thing that comes to mind is often GIT. However, GIT is not made to keep track of large files which datasets often are. Therefore, there are separate data versioning tools which work with GIT or provide versioning in other ways:

- Data Version Control (DVC): DVC is an open-source Git extension which makes it possible to link large data and model files to a Git repository. Working with similar commands as GIT, it is easy to set up and provide a quick data versioning solution.
- git Large File Storage (LFS): An open-source GIT extension to help you version large files while the files themselves are stored on a remote server.
- Pachyderm: An open-source data science platform which provides branching and data workflow management for experimenting with data. It is based on Docker containers and does provides the scalability and flexibility which docker containers provide as well. Next to their data versioning functions, this tool also provides data pipelining functionalities.

These tools helps you version datasets which are then stored in another place. However, where do you store all this data? There are two main options.The first option is to store it locally, either on the same physical machine where you are processing your data or on the local network through a network file system (SMB, NFS, Ceph). This can work very well when working with smaller datasets. However, it can get very expensive to buy and manage all storage space for the datasets yourself. That is why storing datasets in the Cloud has become a popular option. Data in the cloud is often stored in what is called a data lake, a storage which scales nearly infinitely and provides the possibility to store different types of data next to each other. The most popular type of data lake is S3, from Amazon or otherwise, since it is widely supported and is often the cheapest option. Storing and managing your data raw in S3 is possible but often not the best option. There are therefore a lot of tools which build on S3 to provide versioning and create an ACID (Atomicity, Consistency, Isolation, and Durability) databases. If you need more advanced operations for your data lakes or want to manage it better, you can use an Online Analytical Processing tool (OLAP). The following list contains some popular tools used in the world of data lakes.

- Amazon S3 (official website): Amazon has a whole platform to get your application running in the cloud. One part of this is the S3 data storage which also includes data versioning capabilities.
- LakeFS: This is an open-source platform which provides a git-like method with branching and committing to data lake models. It also provides ACID transactions and supports S3 buckets and Google cloud storage.
- Delta Lake: This open-source platform provides a storage layer to help improve data lakes. It provides ACID transactions to these lakes and includes metadata management. With Delta lake, there is a focus on using Apache Spark, although it does provide connections with other tools.
- Databricks: A data and ML SaaS platform by the founders of Apache Spark. It provides a data lakehouse platform, which runs on the cloud storage, next to other data and ML functions and capabilities.
- Microsoft Fabric: Another lakehouse platform which runs on the cloud storage from Microsoft. It offers tools for data management but also for data analysis.
- Snowflake: A cloud, SaaS data warehouse manager with their own storage layers running on AWS, Azure, or GCP. They provide SQL, ACID, and time-travel (versioning) features, as well as several analytics and data engineering capabilities.
- BigQuery: A cloud data warehouse manager platform from Google. One of the main differences with Snowflake is that BigQuery stores everything in Google's Colossus.

When the data versioning tool has been decided, you can start collecting data. Depending on the type of data, this can take a while. Not only to collect the data itself but also to create data labels. Think for example about creating object detection labels for a machine vision task. To help with this, there are again a lot of tools available:

- Label Studio: An open-source data labeling tool which supports different types of labeling projects. It is built as a Python library but used as a web-GUI and provides templates for common vision, time series and text labeling tasks. It also provides the possibility to use machine learning models during the labeling process to make the learning process go faster.
- Labelbox: This tool started out as a data labeling tool providing labeling for various data types in a single platform. However, now it also provides other ML project management services. It provides a limited free version and a subscription-based paid version.
- Dataloop: This platform focusses more on an all-round platform for development and production of machine learning applications. In this it also provides a data labeling tool including image, video and text data. However, this tool is not free to use.
- Audino: An open-source audio annotation tool specifically made to make audi-related labeling tasks easier.
- Tagtog: A text annotation tool providing labeling functionality for several text related tasks.
- CVAT: A free, open-source image annotation tool for computer vision. you can self-host the data labelling process on premise, with enterprise options for regulated environments.
- Roboflow: Another data labelling tool for computer vision, which is often used when working with YOLO models. It is has fast auto-label and label assist options which can be integrated with the dataset management system.
- Amazon SageMaker Ground Truth: A data labeller which Supports your own annotators or AWS's workforce. It offers built-in workflows and auto-labeling.

The choice of labelling tool depends strongly on the type of task you want to label. Some tools only work for specific tasks like Tagtog with text data, CVAT and Roboflow for vision data, and Audino for audio data. Others are more general like Labelbox, Dataloop, Sagemaker Ground Truth, and Labelstudio. Labelstudio has the advantage that it is an open source tool, which means that it is free to use.

Once your data is labeled, the preprocessing can start. This includes data cleaning, data transformations, dimensionality reductions, data augmentation and data splitting. These types of tasks are often done using Python code in notebooks like Jupyter, although this can also be done in normal Python environments as well. Some popular data preprocessing tools include:

- Pandas: Pandas is a well-known open-source Python library that makes it easy to create and manipulate row-based dataframes. It is widely used by the data science community to analyze and manipulate data.
- Polars: Another open-source library for data manipulation which is column-based instead of row-based. Its advantage is that it works a lot faster than pandas and some other data manipulation tools.
- Numpy: A fundamental open-source Python package to manipulate N-dimensional data arrays. It also works fast, especially when working together with Numba, an open-source JIT compiler that translates Numpy code into fast machine code.
- Dask: An Open-source python library for parallel computing making it possible to preprocess data working with several machines or nodes.
- Apache Spark: Apache spark is a data engineering engine specifically designed for large-scale data analytics giving the opportunity to scale to more than 1 machine or node. This tool is very popular for "big data" processing.

When working with data it is also important to see what you are working with, before and during the preprocessing phase. We therefore use visualization tools to create an overview of the data. The most common Python visualization libraries are:

- Matplotlib: An open source comprehensive library for creating static visualizations in Python. It is one of the most well-known and widely-used Python packages which makes it well documented and easy to use.
- Seaborn: A statistical open-source data visualization tool which is able to create quick and informative graphics of Pandas dataframes.
- Plotly: Plotly is an open source library for the creation of interactive plots. Although it works well and has a large community, its documentation often lacks in comparison to Matplotlib, making some implementations more difficult.

Each of these libraries have their own strength and weaknesses. Matplotlib is the most common Python library for visualization and is well documented. However, it is not able to create interactive plots. For this, Plotly can be used, even though it is not as well documented. Seaborn is a great tool to use when creating quick graphs for Pandas dataframes, creating interesting graphs with almost no coding. All these libraries work well for single graphs in a notebook. However, If you would like to create interactive dashboards from these graphs you could use tools like Plotly Dash, Streamlit, and Gradio. These are open-source Python libraries which are able to create HTML dashboards. Dash uses Plotly images and is more customizable whilst Streamlit is easier to use and requires less code. With Gradio you can create GUIs in jupyter notebooks but also outside notebooks as web applications. It is even less flexible then Streamlit, but very easy to use. These type of dashboards can be used to create deeper insights into the data and how they interact with each other.

The pipeline of data collection, labelling and preprocessing is looped through multiple times during the MLOps lifecycle, which again is why it is managed by a workflow orchestration tool to automatically follow these steps when new data is collected. Once you have prepared enough data, you can move on to the next step in the MLOps pipeline, being the Model step.

4 Model

This section talks about the tools needed during model training. In this process we are going to create a machine learning model and train it using our input data. This is often done in Python. Three main Python libraries used for machine model training are Scikit-learn, Pytorch, and Tensorflow. Scikit-learn has the advantage that it fairly easy and quick to set up a machine learning model, train and run it. However, there is less flexibility with the machine learning models to create, especially with Neural Networks. It does provide several classic machine learning models next to neural networks. Keras also provides a high-level API but only for neural networks, using TensorFlow. This makes it very easy to start creating neural networks but is less flexible in its neural network possibilities. Pytorch on the other hand, requires a bit more work to create a full neural network training loop but you can control every aspect of the loop easily. It also supports CUDA, making it possible to efficiently use your resources when training your machine learning model on a CUDA device. When training a machine learning model, you will have to choose certain parameters before training. These parameters, called hyperparameters, can have a large influence on the performance of your model. Think for example about the learning rate in a neural network or the type of kernel used in a support vector machine. It is therefore necessary to find the optimal combination of hyperparmeters for your dataset to obtain the best performing model. This process can be tedious and complicated. To help with this process, there are auto ML tools available to help you with your hyperparmeter search.

- Optuna: An open-source Python library for hyperparameter optimization automation. It has several samplers and pruning algorithms implemented and creates nice overview visualization of the search for a clear and controlled hyperparameter search.
- RayTune: Similar to Optuna, Raytune is a python library for hyperparameter optimization. One of the main differences with Optuna is that Raytune is better for working in a distributed manner, making it easy to parallelize the hyperparameter tuning. However, it is more complex to set up and has only basic pruning and visualization capabilities compared to Optuna.
- Weights and Biases: An AI developer platform to train, fine-tune and manage model. It also has a Sweeps option to automate hyperparameter search and nice visualizations of all results. It does provide a limited free version for personal use but requires a price for businesses.
- Edge impulse: An edge ML development platform to create ML application for edge devices. It also launched an auto machine learning tool called EON Turner which not only finds a good ML model but

also the best models for a certain target device. When given a dataset, it will analyze it and show you a few possible results for the given target device.

- Databricks autoML: A data and ML company by the founders of Apache Spark. It provides many data and ML features to train an ML model and deploy it. One of those features is an AutoML tool to quickly generate baseline models and notebooks for ML.

The choice here depends on what other tools you use during your MLOps pipeline. Weights and Biases and Databricks both provide other functionalities which you might want to use in your pipeline. Edge Impulse might be a good idea when you want to deploy your machine learning model on an edge device. Optuna is easy to use and implement in your existing model training and is completely free. Raytune is also free and is better when you want to parallelize your hyperparameter optimization.

Whilst trying different hyperparameters, it is again crucial to keep track of the different models you tried and their results. Next to that, during the MLOps lifecycle your machine learning model will change over time by retraining them. These changes also need to be tracked. Not only the machine learning models themselves but their hyperparameters and metric results should be saved. This is done through the use of model versioning tools also called experiment tracking tools.

- Weights and Biases (Wandb): An AI developer platform to train, fine-tune and manage model. One of its capabilities is to track, compare and visualize your ML models with minimal code. It also contains other ML lifecycle capabilities for deployments and resource monitoring. This tool is not open source but does have a free option.

- MLflow: Open-source ML developer platform widely used for its experiment tracking. It easily keeps track of ML model experiments with a few lines of code. It also has a model registry to easily share models and contains model deployment and evaluation capabilities.

- Comet ML: This is a Cloud-based MLOps platform for ML model experiment tracking. It also includes a visualization UI and recently implemented an LLM evaluation tool called Opik. It is not open source but does provide a limited free option.

- Aim: Aim is an open source and self-hosted ML metadata tracking tool. Its two main applications are experiment tracking and prompt engineering. It works in a UI and requires almost no code to set up.

- ClearML: An all in one web platform for MLops. The free version includes data versioning, model training, experiment management, model repository, pipelines and CI/CD automation. Clear ML Experiment tracks not only ML models and their metrics, it also logs code, notebooks, configuration files and containers.

The choice here lies in the price you want to pay and how you want to implement your experiment tracking. Aim and MLflow are both free and open-source but Aim does not require code whilst MLflow is fully implemented in code. On the other hand, there is Wandb, Comet ML and ClearML, which only provide a limited (open-source) free version. Wandb is more code based whilst Comet ML and ClearML are more cloud based approaches. They all provide other capabilities then experiment tracking and try to be more holistic MLOps platforms.

These are the three main types of tools used whilst training a machine learning model. Once you have trained your model, it is crucial to evaluate it, which leads to the next step in our MLOps lifecycle.

5 Evaluate

This section introduces tools for machine learning evaluations. The created machine learning model needs to be evaluated thoroughly outside and in the production environment, before it can be used. Before the evaluation can be done, it is important to know what you are evaluating on. There are several types of metrics you can evaluate.

The first, and most obvious metric is model performance. Here metrics are included like model accuracy, precision, recall, F1-score, area under the ROC-curve and mean squared error. These metrics should be evaluated on new data which has not been used during the training and hyperparameter optimization. This is done to make sure your machine learning model is not overfitting on the training and validation data.

Another metric is the fairness of the machine learning model. This moves into the field of model explainability. An important question to ask here is if your machine learning model or dataset show any bias and

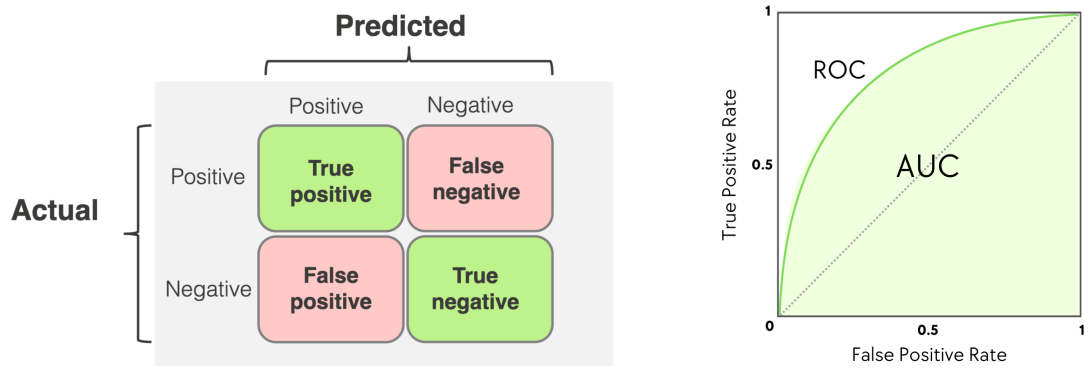

Figure 3: An example of a confusion matrix and the Area Under the Curve ROC curve. Two common metrics used for evaluating machine learning models.

if it makes its decisions in a fair and correct way. In some applications this is a crucial assessment to make. Think for example about a machine learning model which predicts wether or not a person should get a job. If it is trained on data from companies which denied women with the same skills as a man more quickly, the model will learn this too. In those cases it might be a good idea to look at techniques like SHAP and LIME to evaluate which features have a great influence in the result of your machine learning model. The next metric

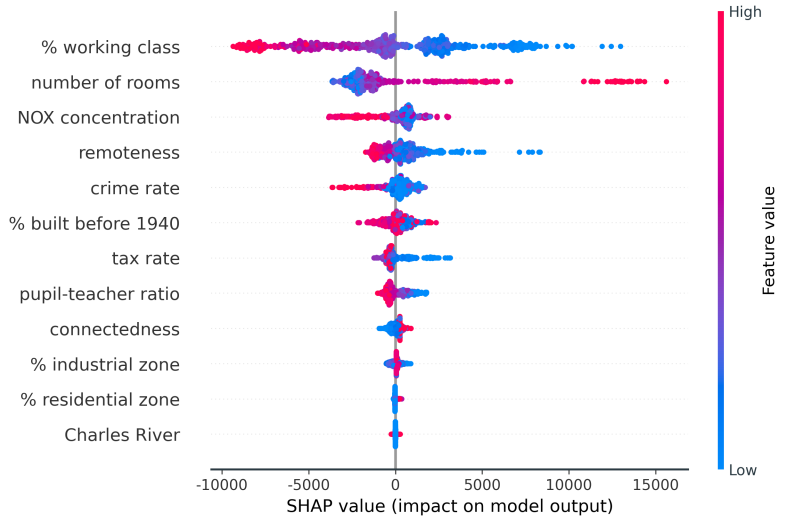

Figure 4: An example of the SHAP output where for each feature you can see if it has a lot of influence on the decision made by the ML model or not.

which might be important to asses is resource consumption. This includes metrics like memory usage and processing power. If the environment on which your machine learning model is running is an edge device, the size of your machine learning model might have a great influence on wether or not you can use it. In real-time industrial machinery, the speed of your machine learning model is of great importance for its usability. If your machine learning model seems too big or too slow, you could use techniques like quantization and pruning to make your machine learning model smaller and faster. However, this often comes at a small cost in accuracy. The last type of metric is business metrics.

Lastly, you might only want to use a machine learning model if it generates a certain amount of profit, or generates a certain amount of clicks on a website. These Key Performance Indicators (KPIs) can also be used for your model evaluation.

There are not a lot of specific model evaluation tools, they are often combined with model deployment, monitoring and experiment tracking tools. Two tools that are more model testing focused are Deepchecks ML and TruEra. Deepchecks focuses more on the testing of the ML model and has a CI and monitoring aspect. TruEra is a tool which has the ability to not only test the performance of a machine learning model but also has fairness & AI Bias and explainability features.

When there has already been a ML model deployed and you want to evaluate if a newly trained model actually performs better than the original model, there are also some testing techniques that can be used. The first one is champion testing. In this test, the original model is called the current champion. You then deploy the other (or several other) model(s) next to it. The data is put through all ML models but only the result of the champion model is used in the application. The results of the model are logged and evaluated. Once a challenger performs better than the original model it is selected as the new champion and used instead of the old model. This technique is used when you have the true labels of your model quickly available. When this is not the case, so when there is no ground truth available, A/B testing is more often used. Only a single new model is deployed next to the original one. Both models will process part of the real input data (e.g. 70% by the original model and 30% by the new model) and the results are evaluated by a statistical metric. Once the new model performs better, it is used instead of the old one.

Once your machine learning model is thoroughly evaluated, it is ready to be deployed in production. This brings us to the Deploy step.

6 Deploy

This section talks about the tools required for ML model deployment. One of the first steps here is to make your model production ready. This includes changing your machine learning model from its development form to a more general deployable state which can be understood by your production environment. There are a lot of tools available which try to make the switch from R&D environments to production environments easier.

- Amazon SageMaker: A platform providing quick model training and deployment. it can run with Jupyter Notebook creating a simple setup. It however only works with AWS ecosystem and does not provide a free version.
- TensorFlow serving: An open-source serving system for machine learning models designed for production environments. It provides out-of-the-box integration with TensorFlow models but can also be extended to serve other types of models.
- Seldon: A real-time machine learning deployment and monitoring tool to deploy models in Kubernetes, making the Kubernetes features of scalability and recourse definition available for the ML models. It also offers connection with CI/CD tools and has an alerting system when problems occur.
- KubeFlow: A ML system maintaining tool for Kubernetes. It packages ML models and organizes docker containers that help maintain an entire machine learning system, making ML deployment easier.
- BentoML: An inference platform to make ML deployment easier. It offers standard Python-based architecture for deploying and maintaining production grade APIs making it easy to package and serve trained models from any ML framework. It supports multiple platforms, not only Kubernetes
- NVIDIA Triton Inference Server:

Which tool you use here again depends on what other tools, and environments you use. If you work with Kubernetes, Kubeflow or Seldon might be the best option and hen working with Amazon, Sagemaker is the best choice. If you are using Tensorflow and want to work with an open-source tool, it might be a good idea to use Tensorflow serving. BentoML might be the best choice for you when you use other platforms than Kubernetes but still want a good deployment tool. These five tools are not the only deployment tools available, as you might have noticed, a lot of earlier mentioned MLOps tools also involve a deployment strategy including MLflow, Apache Airflow, and Weights and Biases.

Until now all tools focus on cloud or server deployments of the ML model. However, if you are working on an edge device, different priorities are in order like the size and speed of the machine learning model inference. For this you export your machine learning model to an export format which creates an optimized version of your model to run on the edge device. The main export format for neural networks is Open Neural Network Exchange (ONNX). Others options include TorchScript and TenserFlow Lite. To make your model even more efficient, you can optimize it through different techniques, for example through quantization. There are again some tools which can help you with deployment for the edge.

- ONNX Runtime: An open-source engine to accelerate your machine learning model for deployment. This tool works with multiple languages and accross multiple platforms with pretty good performance.

- Tensor RT: This tool from NVIDIA is an optimized inference library which is able to maximize performance of deep learning models specifically for NVIDIA hardware (GPUs).
- Tensorflow Lite: This tool improves your tensorflow machine learning model for embedded devices like mobile and even microcontrollers.
- PyTorch ExecuTorch: A promising tool, still in its alpha stage, for training and inference of Pytorch machine learning model on the edge.
- Apache TVM: This open-source machine learning compiler framework. It takes in pretrained models and compiles and generates deployable models for embedded devices. This tool is not very mature yet but still looks promising.
- Vendor specific frameworks like Apple Core ML, Microsoft DirectML, Qualcomm QNN SDK, 'Rockchip RKNN, Texas Instruments Edge AI Studio: These options can provide good results, however, they only work for their specific devices.

Next to the preparation of the ML model itself, we also need to deploy our code into production. A common problem with deploying code used to be the "it works on my machine" problem. This entails that a certain program works on the developers computer without any problem, but once another person tries to run it on their own computer, suddenly the code breaks. To solve this problem Virtual Machines (VM) were introduced. These virtual machines create a separate machine virtually on the computer of where you want to run the code. This machine contains the correct operating systems, bins and libraries to be able to run the program without any problems. However, the problem of these VMs is that they are quite large since they contain their own operating system. That is why containers were introduced. Instead of creating their own operating systems, containers include the application code, the code dependencies and minimal system utilities which should be enough to run the program without any faults. These containers are more lightweight, making them faster, smaller and easier to deploy and scale. The most common tool used for containerization in industry is Docker which packages your app and its dependencies into an image. This image can then run everywhere you want it to, producing the same behavior every time. It is normally common to create several docker images of separate parts of your program that then can interact with each other. For example, you create a database image, an app image, and a backend image. When working with small programs these can easily be set up and managed with the use of the Docker Compose tool. However, working, with large code bases which often run on multiple machines, things can get complicated very easily. That is when you can use Kubernetes, the widely used tool for container management.

These tools will make it possible to deploy your ML model in the production environment. However, this is not where the MLOps lifecycle ends. The next step in this cycle is the Monitor step.

7 Monitor

This sections talks about the monitoring tools needed for keeping track of the model and its environment during production. Your machine learning model should be running in production. However, the environments in which your machine learning model is running are not static, in fact they can change drastically over time. Different lighting conditions, different machines running in the background causing vibrations in the production environment, or the temperature in the environment changing due to seasonality or the influence can all have influence on the data received by the model. These changes are called drift and in a lot of situations drift can cause the performance of the machine learning model to degrade. It is therefore crucial to monitor the data and model performance, next to other production environment variables to catch these drifts and start adapting your model. There are three main ways to monitor your data and machine learning model. The first one is by monitoring the data quality. You monitor certain metrics of your incoming data and compare them to a reference dataset to compare the quality of the data. Some metrics to monitor are the number of Null values, the number of constant columns or rows, and the number of features and their input shapes. Monitoring these things can already give a first indication when something goes wrong like a sensor that breaks. The second aspect to monitor is the model performance. This is done through metrics like model accuracy, F1 score, MSE, and so on. These metrics clearly show when the model is degrading and you should start retraining. However, these metrics also assume the instant availability of data labels to compare the model predictions to. This is most of the time not the case. Therefore, there is a third aspect to monitoring which is the data drift in both the data inputs and model outputs. There are a lot of data drift detection methods available, but one of the most common ways to do this is by comparing the distribution of the current data to the

distribution of a reference dataset. This is done through the use of statistical tests and distance metrics like the Kolmogorov-Smirnov test and the Wasserstein distance. If the two distributions are deemed too different, or the distance metric is too large, drift is detected. There are again several ML monitoring tools which can help you implement these ML monitoring metrics. Often times they create dashboards or overview reports and can have alerting features when something goes wrong. A list of some more promenent ML monitoring tools can be seen below.

- Evidently: An open source ML observability platform allowing easy monitoring setup through the use of preset reports and test suites. It gives you the possibility to save these test suites and reports as HTML, JSON and dictionaries, not needing a connection to a separate server.
- Arize: A web UI based model monitoring platform which has monitoring aspects for drift and data integrity. It also has other features like explainability, model performance improvements and pre-launch validation. It does have a free option for a single developer.
- Fiddler: This is a web UI ML monitoring tool which lets you monitor data and predictions through with an in depth visualization. It also has outliers tracking, service metrics and alerts. This tool does not provide a free version.

Next to these more specific ML observability and monitoring frameworks, there are also other tools from the Evaluation and Deployment step which also provide ML monitoring features. For example TruEra, Amazon Sagemaker, and Kubeflow.

In most companies, the code and production environments is already being monitored using dashboarding. The tools which are often used to create these dashboards are Prometheus and Grafana, two tools which, when working together, can read and save incoming data and with the PromQL language and can be shown on a self-made dashboard. Some of the earlier mentioned ML monitoring tools can work together with Prometheus and Grafana to create clear dashboards for the results of their monitoring tests.

8 Closing the loop

The changes in the ML environment can cause the model to degrade. To do something about this, the original ML model needs to be adapted to its new environment. This is where the continuous learning comes in to play. There are two big questions to answer for adapting the model. The first question is "when do you adapt your model?". There are two main answers to this question. The first one is to start adapting your model in scheduled time intervals (daily, weekly, monthly, etc). This can be effective but you might be retraining a machine learning model when it is not necessary to do so, which uses unnecessary resources. Another, better strategy is therefore to first detect when the environment has changed using drift detection and only then adapt the machine learning model. The second question to answer is "How do you adapt your model?". Again, there are several strategies. The first one is to start training a new model from scratch, moving through all previous steps. This can be effective but can take a lot of time and energy for sometimes a small improvement. A second option is to implement what is called lazy learning where you finetune or retrain the original model on new data. This is a very fast option. However this method is susceptible to catastrophic forgetting, where the model overfits on the new data and fully forgets the old task it had learned. For example when you are working with an animal classification model, if your original model could differentiate cat and dogs and your new data contains only horses, it might only be able to classify horses afterwards and not be able to detect cats and dogs anymore. The next strategy is using incremental learning techniques. These are training method specifically created to avoid catastrophic forgetting. With this technique the difficulty lies in the tradeoff between remembering old data and learning new data. The last strategy is to work with ensemble methods where you work with multiple machine learning models, which can all be different types, and only use the one which is most relevant in that time. Having these different models available does make it easier to switch between the models efficiently avoiding model downtime for retraining. This method is very effective when you have seasonality in your data.

Next to closing the model loop, it is also possible to create a data flywheel. This is a self-improving continuous feedback loop where data collected from a product or service is used to enhance AI models, which in turn generate better outcomes, attract more users, and produce even more valuable data to continue the cycle.

9 Conclusion

It should be clear now that all these steps are crucial to obtain a functional and sustainable ML application. In these few chapters all the different steps in the MLOps lifecycle have been explained with relevant tools for each of these steps. One thing to reiterate is that whilst some tools have only been mentioned in one step, since they are most relevant there, a lot of these tools are also usable in multiple different steps in the MLOps lifecycle. For example MLflow works very well for experiment tracking but also has some great deployment features and tools like Amazon Sagemaker and MLRun try to build a all-round MLOps tool which has features from the Model to the Monitor step. The given lists of tools are also not exhaustive but do include some of the more prominent tools. For a more in depth list of tools, we recommend the awesome-MLOps list on github or the Neptune.ai article on the MLOps landscape. You can also use this map of the MLOps stack, provided from Henrik Skogstrom as a guide for deciding the MLOps tools you want to use. Lastly, for more exercises around these topics you can go to the MLOps4ECM website under the workshop events.

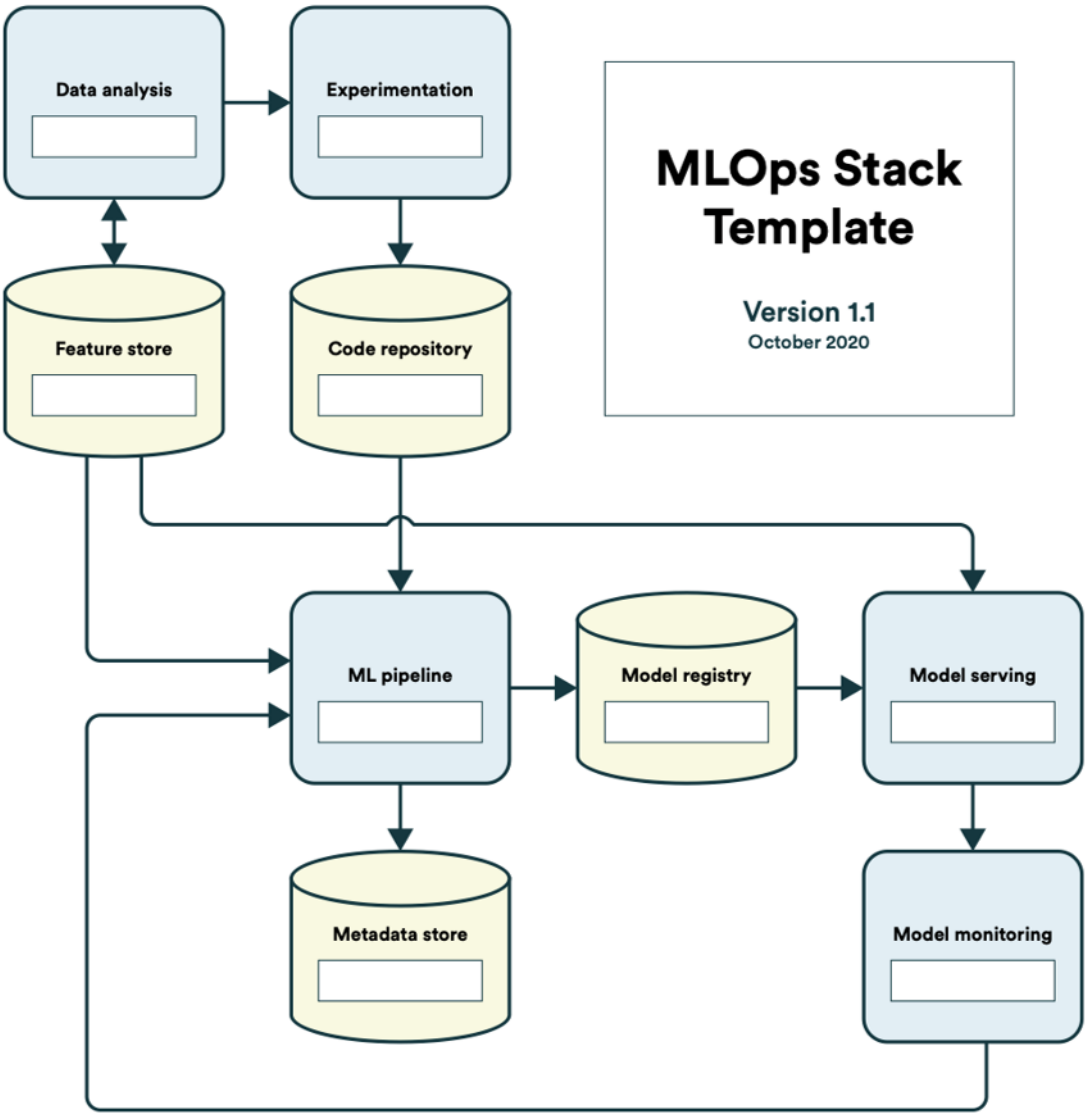

Figure 5: Overview of the MLOps stack which can be used as a template for tool choices during the plan phase.