INNOQ

MLOps

Sustainable Development and Operation of Machine Learning Applications

Anja Kammer • Alexander Kniesz • Dr. Larysa Visengeriyeva

MLOps

Sustainable Development and Operation of Machine Learning Applications

Alexander Kniesz
Anja Kammer
Dr. Larysa Visengeriyeva

innoQ Deutschland GmbH
Krischerstraße 100 · 40789 Monheim am Rhein · Germany
Phone +49 2173 33660 · www.INNOQ.com

Layout: Tammo van Lessen with XeLaTeX
Design: Murat Akgöz
Typesetting: André Deuerling

MLOps – Sustainable Development and Operation of Machine Learning Applications
Published by innoQ Deutschland GmbH
1st Edition · October 2024

Contents

1 Why you Might Want to use Machine Learning

According to Statista Digital Economy Compass 2019[1], two major trends will disrupt the economy and our lives:

- **Data-driven** world, which is linked to the exponentially-growing amount of digitally-collected data.
- The increasing importance of **Artificial Intelligence** / **Machine Learning** / **Data Science**, which derives insights from this tremendous amount of data.

For the sake of consistency, we will use the term *machine learning* (*ML*), however, the concepts apply to both *artificial intelligence* and *data science* fields.

Every machine learning pipeline is a set of operations, which are executed to produce a model. An ML model is roughly defined as a mathematical representation of a real-world process. We might think of the ML model as a function that takes some input data and produces an output (classification, sentiment, recommendation, or clusters). The performance of each model is evaluated by using evaluation metrics, such as *precision & recall*, or *accuracy*.

Being a powerful tool, machine learning can solve many practical problems. Similar to any other software tools, we would need to identify the "right" nail (use-case or problem) to use this "hammer" (machine learning algorithms).

We are interested in including machine learning into software systems because ML might solve some problems, which can be too complex to be solved traditionally. For such problems, a probabilistic (stochastic) solution that is implemented by using machine learning, might be the right way to pursue. For example, *Perceptive problems* in conversational UIs can be solved with techniques such as *speech recognition* or *sentiment analysis*. Machine learning (deep learning) appears to be the most appropriate one because such problems have a large number of elements with different representations. Another type of problems that suitable for ML, are multi-parameters problems. For example, we apply machine learning

[1] https://cdn.statcdn.com/download/pdf/DigitalEconomyCompass2019.pdf

approaches to generate a *stock prices* prediction, which is a foundation for a *stock trading* decisions.

Placing models into production means making your models available to the software systems. Practically, by deploying the ML model, we can provide the following functionality:

- *Recommendation*, which identifies the relevant product in a large collection based on the product description or user's previous interactions.
- *Top-K Items Selection*, which organizes a set of items in a particular order that is suitable for user (e.g. search result).
- *Classification*, which assigns the input examples to one of the previously defined classes (e.g *"spam"*/*"not spam"*).
- *Prediction*, which assigns some most probable value to an entity of interest, such as stock value.
- *Content Generation*, to produce new content by learning from existing examples, such as finishing a Bach chorale cantata by learning from his former compositions.
- *Question Answering*, which answers an explicit question for example: "Does this text describe this image?"
- *Automation*, which can be a set of user steps performed automatically, such as stock trading
- *Fraud and Anomaly Detection*, to identify an action or transaction being a fraud or suspicious
- *Information Extraction and Annotation*, to identify important information in a text, such as people's names, job descriptions, companies, and locations.

In the following Table, we summarize the ML/AI capabilities:

1.1 Deployment Gap

More and more enterprises are experimenting with ML. Getting a model into the real world involves more than just building it. In order to take full advantage of the built ML model by making it available to our core software system, we would need to incorporate the trained ML model into the core codebase. That means,

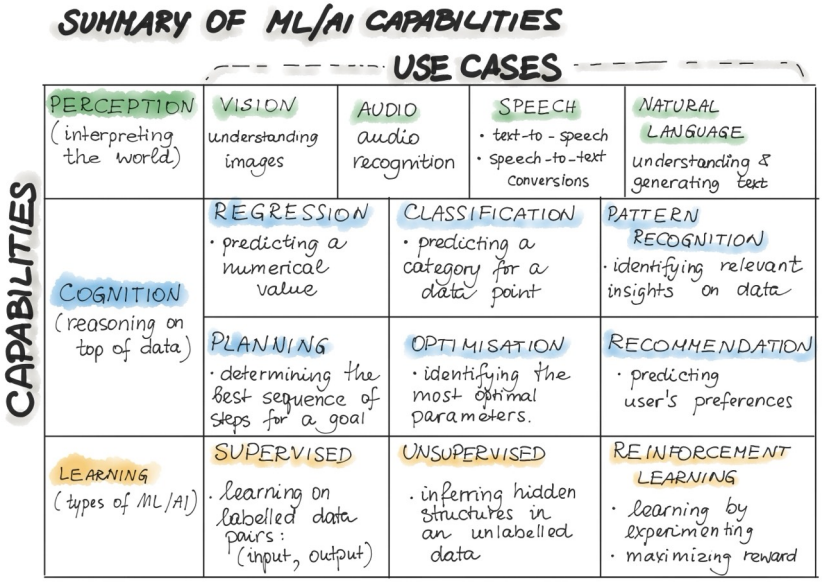

Figure 1.1: Summary of ML/AI Capabilities (source: "The AI Organization" by David Carmona)

we need to deploy the ML model into production. By deploying models, other software systems can supply data to these and get predictions, which are in turn populated back into the software systems. Therefore, the full advantage of ML models is only possible through the ML model deployment.

However, according to a report by Algorithmia "2020 State of Enterprise Machine Learning"[2], many companies haven't figured out how to achieve their ML/AI goals. Because bridging the gap between ML model building and practical deployments is still a challenging task. There's a fundamental difference between building a ML model in the Jupyter notebook model and deploying an ML model into a

[2] https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia_2020_State_of_Enterprise_ML.pdf?utm_campaign=The%20Batch&utm_source=hs_email&utm_medium=email&utm_content=80984419&_hsenc=p2ANqtz--sz-e2gfqUeDvVSmjsXfvwOnLHB2ZkSdQsO1IRRAdnBIboemf-JTh8NnwFxB-FeZberIw7_rI9ERTy8zFW8jvoTzjOfA&_hsmi=80984419

production system that generates business value. Although AI budgets are on the rise, only 22 percent of companies[3] that use machine learning have successfully deployed an ML model into production.

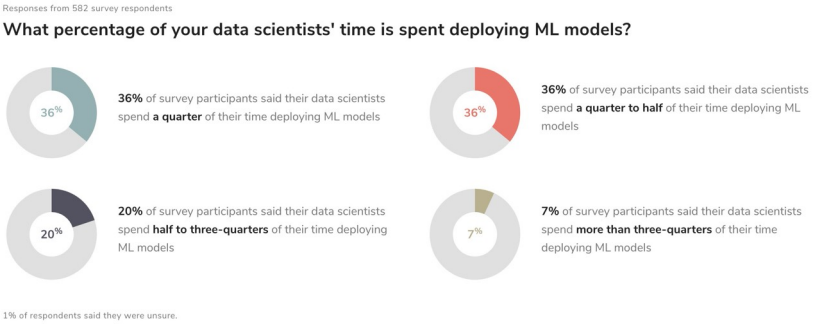

Figure 1.2: ML deployment difficulties (source: Algorithmia)

The "2020 State of Enterprise Machine Learning"[4] report is based on a survey of nearly 750 people including machine learning practitioners, managers for machine learning projects, and executives at tech firms. Half of the respondents answered that it takes their company between a week and three months to deploy an ML model. About 18 percent stated that it takes from three months to a year. According to the report *"The main challenges people face when developing ML capabilities are scale, version control, model reproducibility, and aligning stakeholders"*.

[3] https://designingforanalytics.com/resources/failure-rates-for-analytics-bi-iot-and-big-data-projects-85-yikes/

[4] https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia_2020_State_of_Enterprise_ML.pdf?utm_campaign=The%20Batch&utm_source=hs_email&utm_medium=email&utm_content=80984419&_hsenc=p2ANqtz--sz-e2gfqUeDvVSmjsXfvwOnLHB2ZkSdQsO1IRRAdnBIb0emf-JTh8NnwFxB-FeZberIw7_rI9ERTy8zFW8jvoTzjOfA&_hsmi=80984419

1.2 Scenarios of Change That Need to be Managed

The reason for the previously described deployment gap is that the development of the machine learning-based applications is fundamentally different from the development of the traditional software. The complete development pipeline includes three levels of change: **Data**, **ML Model**, and **Code**. This means that in machine learning-based systems, the trigger for a build might be the combination of a code change, data change, or model change. This is also known as "*Changing Anything Changes Everything*" principle[5].

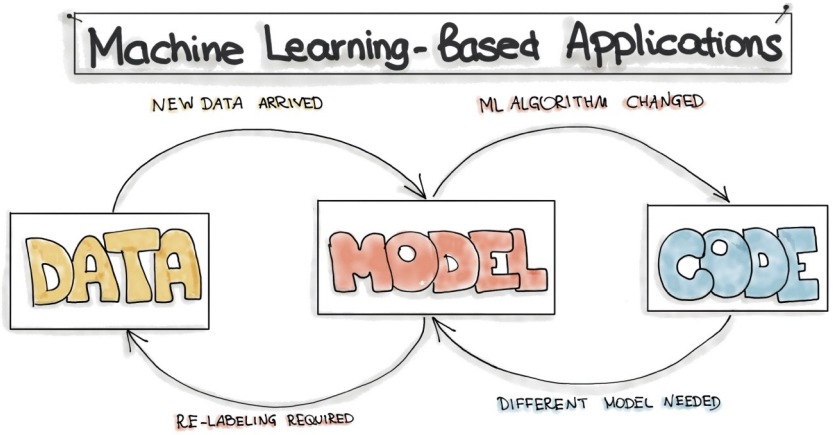

Figure 1.3: Three layers of change

In the following, we list some scenarios of possible changes in machine learning applications:

- After deploying the ML model into a software system, we might recognize that as time goes by, the model starts to decay and to behave abnormally, so we would need new data to re-train our ML model.

[5] https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

- After examining the available data, we might recognize that it's difficult to get the data needed to solve the problem we previously defined, so we would need to re-formulate the problem.
- In the ML project at some stages, we might go back in the process and either collect more data, or collect different data and re-label training data. This should trigger the re-training of the ML Model.
- After serving the model to the end-users, we might recognize that the assumptions we made for training the model are wrong, so we have to change our model.
- Sometimes the business objective might change while project development and we decide to change the machine learning algorithm to train the model.

Additionally, three common issues influence the value of ML models once they're in production.

The first is ***data quality***: since ML models are built on data, they are sensitive to the semantics, amount and completeness of incoming data.

The second is ***model decay***: the performance of ML models in production degenerate over time because of changes in the real-life data that has not been seen during the model training.

The third is ***locality***: when transferring ML models to new business customers, these models, which have been pre-trained on different user demographics, might not work correctly according to quality metrics.

Since ML/AI is expanding into new applications and shaping new industries, building successful ML projects remains a challenging task. As shown, there is a need to establish effective practices and processes around designing, building, and deploying ML models into production - MLOps.

Further reading: Why is DevOps for Machine Learning so Different?[6]

[6] https://hackernoon.com/why-is-devops-for-machine-learning-so-different-384z32f1

1.3 MLOps Definition

We saw what real-world problems might be solved by applying machine learning. We established the challenges of getting the ML models into production.

Finally, we are set up to define the term **MLOps**:

The term MLOps is defined as *"the extension of the DevOps methodology to include Machine Learning and Data Science assets as first-class citizens within the DevOps ecology"* Source: MLOps SIG[7].

Alternatively, we can use the definition of **Machine Learning Engineering** (**MLE**), where *MLE is the use of scientific principles, tools, and techniques of machine learning and traditional software engineering to design and build complex computing systems. MLE encompasses all stages from data collection, to model building, to make the model available for use by the product or the consumers."* (by A.Burkov).

MLOps, like DevOps, emerges from the understanding that separating the ML model development from the process that delivers it — ML operations — lowers quality, transparency, and agility of the whole intelligent software.

1.4 The Evolution of the MLOps

In the early 2000s, when businesses needed to implement machine learning solutions, they used the vendors' licensed software such as SAS, SPSS, and FICO. With the rise of open-source software and the availability of data, more software practitioners started using Python or R libraries for training ML models. However, the usage of the models in production was still a problem. As the containerization technology was emerging, the deployment of the model in a scalable way was solved by using Docker containers and Kubernetes. Recently, we see the evolution of those solutions into ML deployment platforms that cover the whole iteration of model experimentation, training, deployment, and monitoring. The following Figure visualizes the evolution of the MLOps.

[7] https://github.com/cdfoundation/sig-mlops/blob/master/roadmap/2020/MLOpsRoadmap2020.md

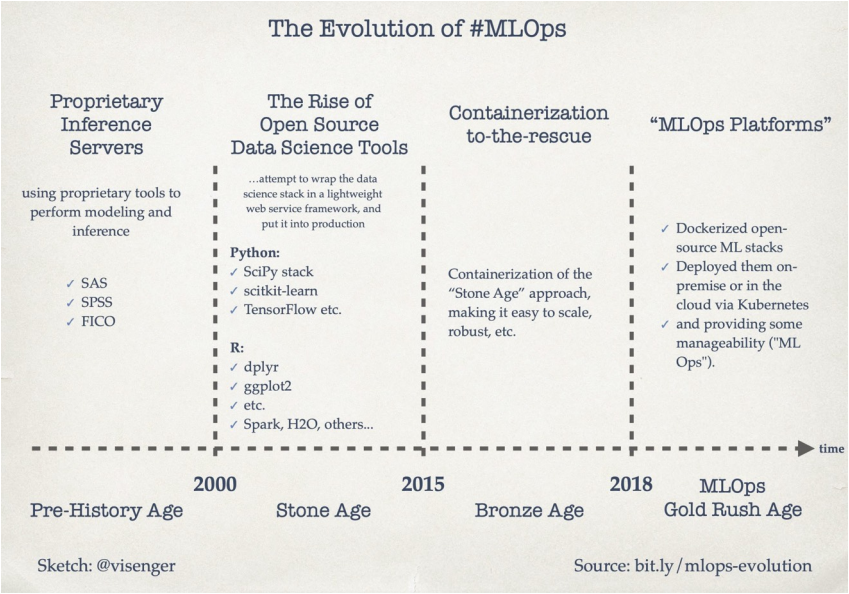

Figure 1.4: The evolution of MLOps (source: bit.ly/mlops-evolution)

2 "What is the business problem that we are trying to solve here?"

The most important phase in any software project is to understand the business problem and create requirements. ML-based software is no different here. The initial step includes a thorough study of business problems and requirements. These requirements are translated into the model objectives and the model outputs. Possible errors and minimum success for launching need to be specified. The most useful question to continue working on the AI/ML solution is ***"how costly are wrong predictions?"*** Answering that question will define the feasibility of the ML project.

2.1 Work Flow Decomposition

Each task of the entire business process needs to be decomposed into its constituent elements in order to see where prediction (ML model) can be introduced.

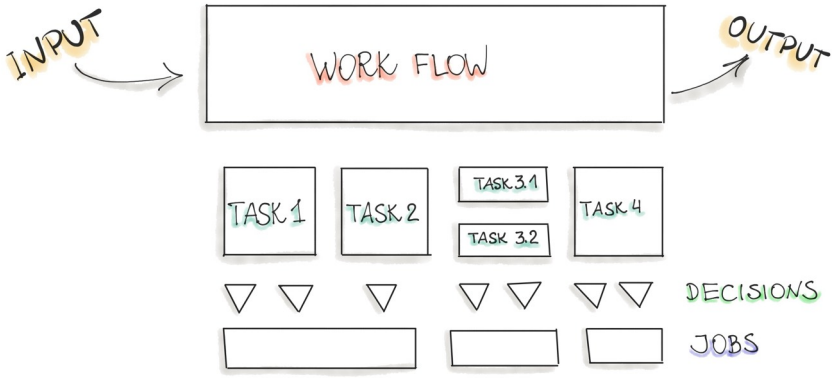

Figure 2.1: Work Flow Decomposition (source: ml-ops.org)

To anwer the question *"how to implement AI/ML"*, we follow the next steps:

1. Identify the concrete ***process*** that might be powered by AI/ML (see the Figure above).
2. Decompose that process into a directed graph of ***tasks***.
3. Identify where humans can be removed from the task, meaning, what task can be replaced by a prediction element such as ML model?
4. Estime the ROI for implementing an AI/ML tool to perform each task.
5. Rank-order the AI/ML implementation for each ***task*** in terms of ROI.
6. Start from the top of the list and structure the AI/ML implementation by completing either the *AI Canvas* or the *Machine Learning Canvas*.

The *AI Canvas* or its alternative, the *Machine Learning Canvas*, assist and help to structure the breakdown process. They also help to articulate exactly what is needed to predict and how we react on errors made by the prediction algorithm.

2.2 AI Canvas

The *AI Canvas* was proposed by A. Agrawal et. al in their book *"Prediction Machines. The Simple Economics of Artificial Intelligence." 2018*, and "is an aid for contemplating, building, and assessing AI tools". The example of such canvas and the description of each component is provided in the Figure below:

2.3 Machine Learning Canvas

While the above AI canvas represents a high-level structure of the ML/AI implementation, at some point we would like to specify both the vision for the ML system and the specifics of the system. To achieve those goals there is another tool, the *Machine Learning Canvas*, as suggested by Louis Dorard[1]. This canvas structures the ML project and helps to specify the core requirements to realise the project. Initially, we identify the objective by answering a question *what do we want to achieve for the end-users of the predictive system?* Next, we connect the business goal to the ML task.

[1] https://www.louisdorard.com/

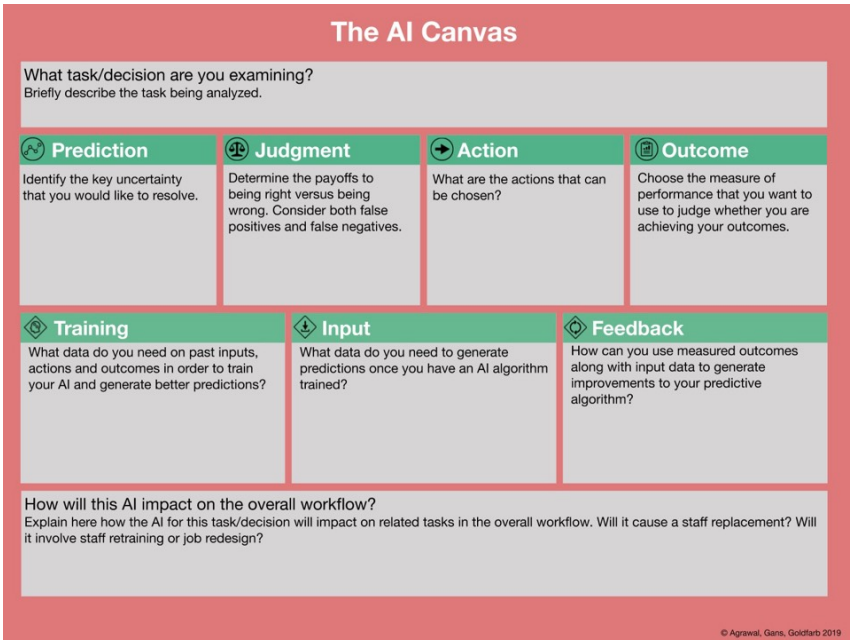

Figure 2.2: AI Canvas (source: https://hbr.org/2018/04/a-simple-tool-to-start-making-decisions-with-the-help-of-ai)

The central part of the canvas is the *Value Proposition* building block, which describes products or services that create some value for customers. Typically, we answer the following questions: *What* problems are we trying to solve? *Why* is it important? *Who* is the end-user of our system? What value does the ML project deliver to the end-user? How will they use your outputs/predictions?

The remaining canvas is divided into three broad categories: *Learning*, *Prediction*, and *Evaluation*. The Learning category is responsible to specify how the ML model will be learned. The Prediction part describes how the prediction is performed. Finally, the Evaluation category contains methods and metrics for the ML model and the system evaluation. The following machine learning canvas is an example provided by Louis Dorard[2]:

[2] https://www.louisdorard.com/

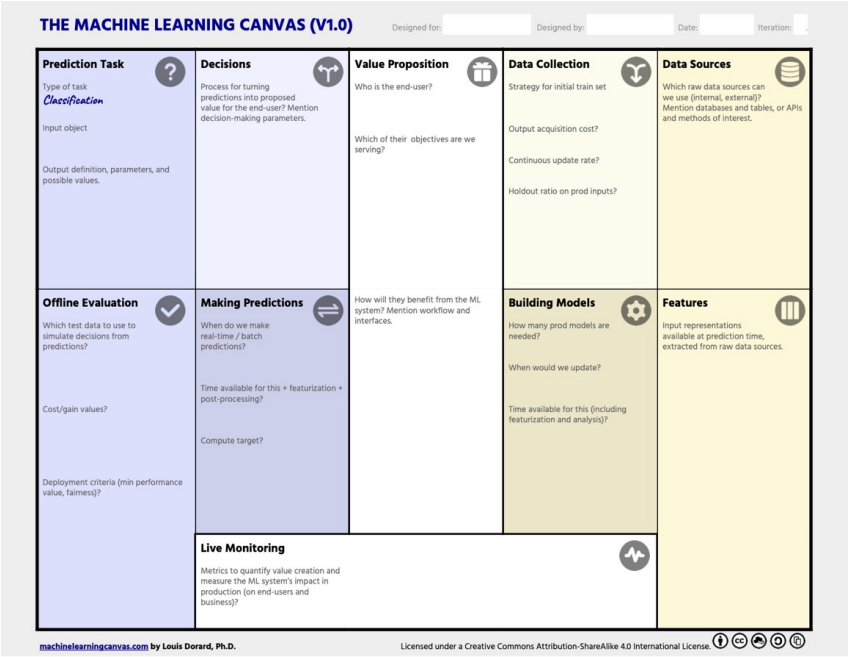

Figure 2.3: Machine Learning Canvas (source: machinelearningcanvas.com)

In total, the Machine Learning Canvas is structured as ten compound blocks, such as *Value Proposition, Data Sources, Prediction Task, Features (Engineering), Offline Evaluation, Decisions, Making Predictions, Collecting Data, Building Models,* and *Live Evaluation and Monitoring*. Each of those blocks is focused on one aspect of the future ML application:

2.3.1 Value Proposition

This is the crucial blocks in the whole canvas. Here we should answer three important questions:

1. *What* is the problem? What objective are we serving? What are we trying to do for the end-user?
2. *Why* is it important?

3. *Who* is the end-user? Can we specify the *persona*?

To create an effective *Value Proposition* statement, we could use the Geoffrey Moore's value positioning statement template:[3]

****For (target customer) who (need or opportunity), our (product/service name) is (product category) that (benefit).****

Narrowing the domain[4] of the problem could be useful for the next question regarding the required data. For example, instead of creating a universal chat-bot, build a bot that helps with scheduling conference-calls.

2.3.2 Data Sources

Data is essential for training ML models. In this block, we clarify all available and possible data sources to be used for the ML task. As an example, we might consider using:

- Internal/external databases.
- Data marts, OLAP cubes, data warehouses, OLTP systems.
- Hadoop clusters,
- REST APIs to gather data.
- Static files, spreadsheets.
- Web scraping.
- The output of other (ML) systems.
- Open-source data sets.
    - Useful publicly available datasets: Kaggle Datasets[5], Google's Dataset Search[6], UCI Repository[7], or Wikipedia's list of datasets for machine-learning research[8]

[3] https://the.gt/geoffrey-moore-positioning-statement/
[4] https://cdixon.org/2015/02/01/the-ai-startup-idea-maze
[5] https://www.kaggle.com/datasets
[6] https://datasetsearch.research.google.com/
[7] https://archive.ics.uci.edu/ml/datasets.php
[8] https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research

Furthermore, we should clarify the *hidden costs* of a machine learning application.

- How expensive could get the data storage?
- Should we purchase external data?
- What data assess tools and processes are available to make data accessible from other systems?

2.3.3 Prediction Task

After clarifying what data is available, we brainstorm what type of ML should be used. Here are some examples of questions that might clarify the ML Task:

- Supervised or unsupervised learning?
- Is this anomaly detection?
- Is the problem about which option should be taken? (recommendation)
- Do we need to predict a continuous value? (regression)
- Which category need to be predicted? (classification)
- Do we need to group our data? (clustering)
- If supervised, what type of ML task should be taken: classification, regression, or ranking?
- If classification, will it be binary- or multiclass classification task?
- What is the input for a prediction task?
    - e.g. E-mail text.
- What is the output of the prediction task?
    - e.g. “spam” and “regular”
- What is the degree of complexity our ML Model could assume?
    - e.g. is our model a combination of other ML models? Do we employ ensemble learning? How many hidden layers included in the deep learning model?
- What are the complexity costs, such as training and inference time, for the above models?

2.3.4 Features (Engineering)

As every ML algorithm requires input data in the form of features, we should clarify how should the input data be represented.

- How do we extract features from raw sources?
- Consider to include domain experts to specify what data aspects are most important for the particular ML task.

2.3.5 Offline Evaluation

Before any implementation of the ML model training, we would need to specify and set up the methods and metrics to evaluate the system before deployment. Here we would need to specify:

- Domain specific metrics that justify the deployment of the ML model. For example, simulated with the training and testing data, would the prediction of the model generate more revenue than the revenue created in the "traditional" way.
- What technical evaluation metrics should be used?
  - Precision, Recall, F-1 measure.
  - Accuracy.
- What is the meaning of model prediction errors such as *false positives* and *false negatives*?
- What is our test data?
- How much test data do we need to be confident that the ML model performs well?

2.3.6 Decisions

After completing the ML task, Feature engineering, and the evaluation details, the next is to specify:

- How are prediction used to make decisions?
- How does the end-user or the system interacts with the model predictions?

    - e.g. What happens if the user gets a list of product recommendations? What happens if the incoming e-mail is classified as "spam"?
- Are there *hidden costs* in decision making, such as *human in the loop*?

Such information is required to later decide on how to deploy the ML model.

2.3.7 Making Predictions

This block includes information about when we make a prediction on new inputs.

- When should predictions be available?
    - New predictions are made each time when the user opens the app, such as recommendations.
    - New predictions are made on request.
    - New predictions are made on schedule.
- Are predictions made *on the fly* for each data point or for a *batch* of the input data?
- How computationally complex could the *model inference* get in the application?
- Is there a *human in the loop* to support in making predictions?

2.3.8 Collecting Data

Related to the *Making Predictions*, the *Collecting Data* block gathers information about ***new data*** that should be collected in order to re-train the ML model. In this way, we specify how we prevent the *ML model decay* phenomenon. Further questions to answer in this block are:

- How do we label the new data?
- How expensive is it to collect new data?
- How expensive is it to process rich media like images, sound, or video?
- Is there *human in the loop* for the manual cleaning and labelling of the incoming data?

2.3.9 Building Models

Tightly related to the previous block, the *Building Models* answers questions regarding updating the ML models, because different ML tasks require different frequencies of model re-training:

- How often the model should be retrained?
  - e.g. hourly, weekly, or with every new data point.
- What are the *hidden costs* for model re-training?
  - e.g. do we use cloud resources to perform such tasks?
  - what is the price policy of the cloud vendor?
  - how should we perform hardware cost estimation?
  - common *Cloud Pricing Calculators* are Google Cloud Calculator[9], Amazon ML Pricing[10],Microsoft Azure Calculator[11]
- How long will it take to re-train the model?
- How do we deal with the scaling issues of cloud operations as they can be more complex and costly?
- Do we plan for change in the tech stack?
  - e.g. how can we deal with the tech stack evolution as new tools and development workflows are emerging in the modern AI?

2.3.10 Live Evaluation and Monitoring

After deployment, the ML model should be evaluated and here we would need to specify both *model* and *business* metrics, which should correlate. Generally, the metrics should follow the *S.M.A.R.T* methodology and be: *Specific, Measurable, Achievable, Relevant,* and *Time-bound*.

- How do we track the system performance?
  - e.g. A/B Testing

[9] https://cloud.google.com/products/calculator
[10] https://docs.aws.amazon.com/machine-learning/latest/dg/pricing.html
[11] https://azure.microsoft.com/en-in/pricing/calculator/

- How do we evaluate the value creation?
  - e.g users spent less time on the inbox.

The deliverable in this stage is the completed Machine Learning Canvas. The effort to fill out this canvas might initiate an existential discussion regarding the *real objective* and *hidden costs* for the ML-software. Such a discussion might result in a decision not to implement AI/ML at all. Possible reasons can be as follows:

- The solution to our problem does not tolerate wrong predictions.
- Implementing AI/ML would generate a low ROI.
- The maintenance of the ML/AI project is not guaranteed.

Another question would be *when to deploy ML/AI?* The following Figure shows the trade-off of early vs. late ML model deployment.

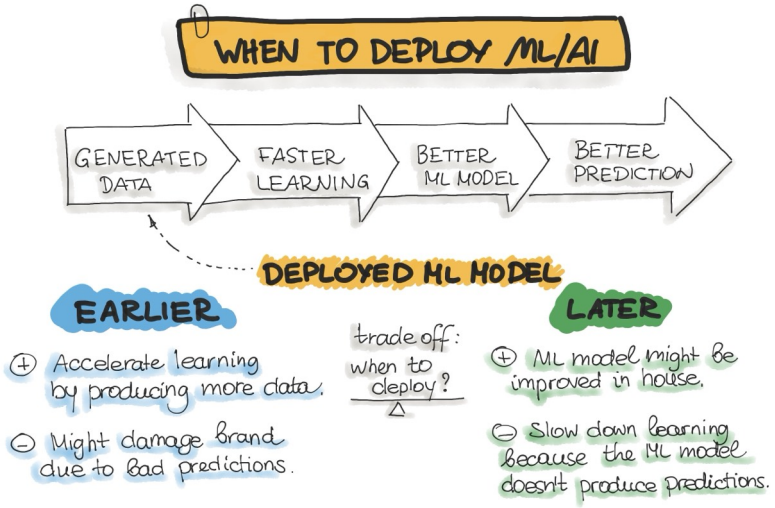

Figure 2.4: When to deploy ML/AI (source: ml-ops.org)

Training

- Domain-driven Design for Machine Learning products[12]

Further reading

- "What is THE main reason most ML projects fail?"[13]
- The New Business of AI (and How It's Different From Traditional Software)[14]

[12] https://www.socreatory.com/en/trainings/ddd4ml

[13] https://towardsdatascience.com/what-is-the-main-reason-most-ml-projects-fail-515d409a161f

[14] https://a16z.com/2020/02/16/the-new-business-of-ai-and-how-its-different-from-traditional-software/

3 Three Levels of ML Software

ML/AI is rapidly adopted by new applications and industries. As already been mentioned, the goal of a machine learning project is to build a statistical model by using collected data and applying machine learning algorithms. Yet building successful ML-based software projects is still difficult because every ML-based software needs to manage three main assets: **Data**, **Model**, and **Code**. Machine Learning Model Operationalization Management - **MLOps**, as a DevOps extension, establishes effective practices and processes around designing, building, and deploying ML models into production. We describe here essential technical methodologies, which are involved in the development of the Machine Learning-based software, namely *Data Engineering*, *ML Model Engineering*, and *Software Release Engineering*.

We recommend ***documenting*** everything you have learned in each step of the whole pipeline.

3.1 Data: Data Engineering Pipelines

We mentioned previously that the fundamental part of any machine learning workflow is **Data**[1]. Collecting good data sets has a huge impact on the quality and performance of the ML model. The famous citation

"Garbage In, Garbage Out",

in the machine learning context means that the ML model is only as good as your data. Therefore, the data, which has been used for training of the ML model, indirectly influence the overall performance of the production system. The amount and quality of the data set are usually problem-specific and can be empirically discovered.

Being an important step, data engineering is reported as heavily time-consuming. We might spend the majority of time on a machine learning project constructing data sets, cleaning, and transforming data.

[1] https://www.datamesh-architecture.com/

The data engineering pipeline includes a sequence of operations on the available data. The final goal of these operations is to create training and testing datasets for the ML algorithms. In the following, we describe each stage of the data engineering pipeline such as *Data Ingestion*, *Exploration and Validation*, *Data Wrangling* (*Cleaning*), and *Data Splitting*.

3.1.1 Data Ingestion

Data Ingestion - Collecting data by using various systems, frameworks and formats, such as internal/external databases, data marts, OLAP cubes, data warehouses, OLTP systems, Spark, HDFS etc. This step might also include synthetic data generation or data enrichment The best practices for this step include the following actions that should be maximally automated:

- Data Sources Identification: Find the data and document its origin (data provenance).
- Space Estimation: Check how much storage space it will take.
- Space Location: Create a workspace with enough storage space.
- Obtaining Data: Get the data and convert them to a format that can be easily manipulated without changing the data itself.
- Back up Data: Always work on a copy of the data and keep the original dataset untouched.
- Privacy Compliance: Ensure sensitive information is deleted or protected (e.g., anonymized) to ensure GDPR compliance.
- Metadata Catalog: Start documenting the metadata of the dataset by recording the basic information about the size, format, aliases, last modified time, and access control lists. (Further reading[2])
- Test Data: Sample a test set, put it aside, and never look at it to avoid the "*data snooping*" bias. You fell for this if you are selecting a particular kind of ML model by using the test set. This will lead to an ML model selection that is too optimistic and will not perform well in production.

[2] https://dl.acm.org/doi/pdf/10.1145/2882903.2903730?download=true

3.1.2 Exploration and Validation

Exploration and Validation - Includes data profiling to obtain information about the content and structure of the data. The output of this step is a set of metadata, such as max, min, avg of values. Data validation operations are user-defined error detection functions, which scan the dataset to spot some errors. The validation is a process of assessing the quality of the data by running dataset validation routines (error detection methods). For example, for "address"-attributes, are the address components consistent? Is the correct postal code associated with the address? Are there missing values in the relevant attributes? The best practices for this step include the following actions:

- Use RAD tools: Using Jupyter notebooks is a good way to keep records of data exploration and experimentation.
- Attribute Profiling: Obtain and document the metadata about each attribute, such as
  - Name
  - Number of Records
  - Data Type (categorical, numerical, int/float, text, structured, etc.)
  - Numerical Measures (min, max, avg, median, etc. for numerical data)
  - Amount of missing values (or *"missing value ratio"* = Number of absent values/ Number of records)
  - Type of distribution (Gaussian, uniform, logarithmic, etc.)
- Label Attribute Identification: For supervised learning tasks, identify the target attribute(s).
- Data Visualization: Build a visual representation for value distribution.
- Attributes Correlation: Compute and analyze the correlations between attributes.
- Additional Data: Identify data that would be useful for building the model (go back to "Data Ingestion").

3.1.3 Data Wrangling (Cleaning)

Data Wrangling (Cleaning) - Data preparation step where we programmatically wrangle data, e.g., by re-formatting or re-structuring particular attributes that

might change the form of the data's schema. We recommend writing scripts or functions for all data transformations in the data pipeline to re-use all these functionalities on future data.

- Transformations: Identify the promising transformations you may want to apply.
- Outliers: Fix or remove outliers (optional).
- Missing Values: Fill in missing values (e.g., with zero, mean, median) or drop their rows or columns.
- Not relevant Data: Drop the attributes that provide no useful information for the task (relevant for feature engineering).
- Restructure Data: Might include the following operations (from the book "Principles of Data Wrangling"[3])
  - Reordering record fields by moving columns
  - Creating new record fields through extracting values
  - Combining multiple record fields into a single record field
  - Filtering datasets by removing sets of records
  - Shifting the granularity of the dataset and the fields associated with records through aggregations and pivots.

3.1.4 Data Splitting

Data Splitting - Splitting the data into training (80 %), validation, and test datasets to be used during the core machine learning stages to produce the ML model.

3.2 Model: Machine Learning Pipelines

The core of the ML workflow is the phase of writing and executing machine learning algorithms to obtain an ML model. The model engineering pipeline is usually utilized by a data science team and includes a number of operations that lead to a final model. These operations include *Model Training*, *Model Evaluation*, *Model Testing*, and *Model Packaging*. We recommend automating these steps as much as possible.

[3] https://learning.oreilly.com/library/view/principles-of-data/9781491938911/

3.2.1 Model Training

Model Training - The process of applying the machine learning algorithm on training data to train an ML model. It also includes feature engineering the hyperparameter tuning for the model training activity. The following list is adopted from "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron[4]

- *Feature engineering* might include:
  - Discretize continuous features
  - Decompose features (e.g., categorical, date/time, etc.)
  - Add transformations of features (e.g., log(x), sqrt(x), x2, etc.)
  - Aggregate features into promising new features
  - Feature scaling: Standardize or normalize features
  - New features should be added quickly to get fast from a feature idea to the feature running in production. Further reading "Feature Engineering for Machine Learning. Principles and Techniques for Data Scientists" by Alice Zheng, Amanda Casari[5]

- *Model Engineering* might be an iterative process and include the following workflow:
  - Every ML model specification (code that creates an ML model) should go through a code review and be versioned.
  - Train many ML models from different categories (e.g., linear regression, logistic regression, k-means, naive Bayes, SVM, Random Forest, etc.) using *standard parameters*.
  - Measure and compare their performance. For each model, use *N-fold* cross-validation and compute the mean and standard deviation of the performance measure on the N folds.
  - Error Analysis: analyze the types of errors the ML models make.
  - Consider further feature selection and engineering.

[4] https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/app02.html#project_checklist_appendix

[5] http://shop.oreilly.com/product/0636920049081.do

- Identify the top three to five most promising models, preferring models that make different types of errors.
- Hyperparameters tuning by using cross-validation. Please note that data transformation choices are also hyperparameters. Random search for hyperparameters is preferred over grid search.
- Consider Ensemble methods such as *majority vote*, *bagging*, *boosting*, or *stacking*. Combining ML models should produce better performance than running them individually. (Further reading "Ensemble Methods: Foundations and Algorithms" by Zhi-Hua Zhou[6])

3.2.2 Model Evaluation

Model Evaluation - Validate the trained model to ensure it meets original business objectives before serving the ML model in production to the end-user.

3.2.3 Model Testing

Model Testing - Once the final ML model is trained, its performance needs to be measured by using the hold-back test dataset to estimate the generalization error by performing the final "Model Acceptance Test".

3.2.4 Model Packaging

Model Packaging - The process of exporting the final ML model into a specific format (e.g. PMML, PFA, or ONNX), which describes the model to be consumed by the business application. We cover the ML model packaging in the part 'ML Model serialization formats' below.

3.2.5 Different forms of ML workflows

Operating an ML model might assume several architectural styles. In the following, we discuss four architectural patterns which are classified along two dimensions:

[6] https://www.amazon.com/exec/obidos/ASIN/1439830037/acmorg-20

1. ML Model Training and

2. ML Model Prediction

Please note that for the sake of simplicity, we disregard the third dimension ***3. ML Model Type***, which denotes the type of machine learning algorithm such as supervised, unsupervised, semisupervised, and Reinforcement Learning.

There are two ways how we perform ***ML Model Training***:

1. Offline learning (aka *batch* or *static learning*): The model is trained on a set of already collected data. After deploying to the production environment, the ML model remains constant until it re-trained because the model will see a lot of real-live data and becomes *stale*. This phenomenon is called *'model decay'* and should be carefully monitored.
2. Online learning (aka *dynamic learning*): The model is regularly being re-trained as new data arrives, e.g. as data streams. This is usually the case for ML systems that use time-series data, such as sensor, or stock trading data to accommodate the temporal effects in the ML model.

The second dimension is ***ML Model Prediction***, which denotes the mechanics of the ML model to makes predictions. Here we also distinguish two modes:

1. Batch predictions: The deployed ML model makes a set of predictions based on historical input data. This is often sufficient for data that is not time-dependent, or when it is not critical to obtain real-time predictions as output.
2. Real-time predictions (aka on-demand predictions): Predictions are generated in real-time using the input data that is available at the time of the request.

After identifying these two dimensions, we can classify the operationalization of machine learning models into four ML architecture patterns:

In the following, we present a description of the model architectural patterns such as *Forecast*, *Web-Service*, *Online Learning*, and *AutoML*.

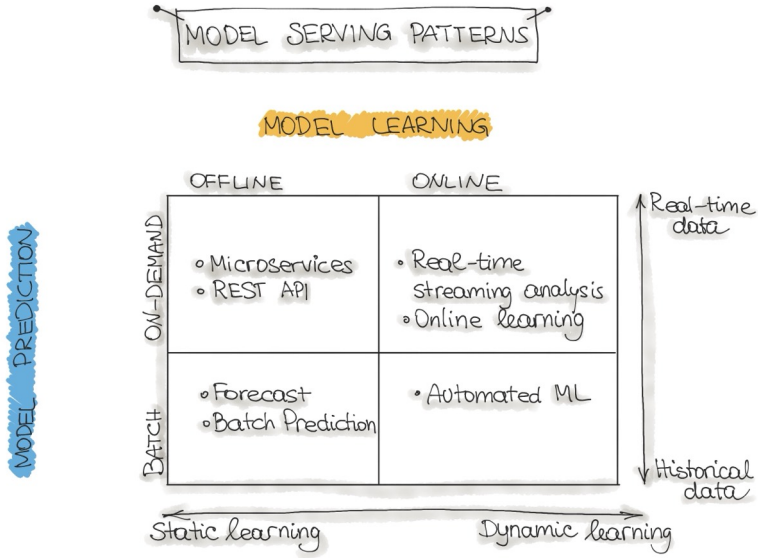

Figure 3.1: Model Serving Patterns (source: https://www.quora.com/How-do-you-take-a-machine-learning-model-to-production)

Forecast

This type of machine learning workflow is widely spread in academic research or data science education (e.g., Kaggle or DataCamp). This form is used to experiment with ML algorithms and data as it is the easiest way to create a machine learning system. Usually, we take an available dataset, train the ML model, then run this model on another (mostly historical) data, and the ML model makes predictions. This way, we output a forecast. This ML workflow is not very useful and, therefore, not common in an industry setting for production systems (e.g. mobile applications).

Web-Service

The most commonly described deployment architecture for ML models is a web service (microservise). The web service takes input data and outputs a prediction for the input data points. The model is trained offline on historical data, but it uses real-live data to make predictions. The difference from a forecast (batch predictions) is that the ML model runs near real-time and handles a single record at a time instead of processing all the data at once. The web service uses real-time data to make predictions, but the model remains constant until it is re-trained and re-deployed into the production system.

The figure below illustrates the architecture for wrapping trained models as deployable services. Please note, we discuss methods for wrapping trained ML models as deployable services in the Deployment Strategies Section.

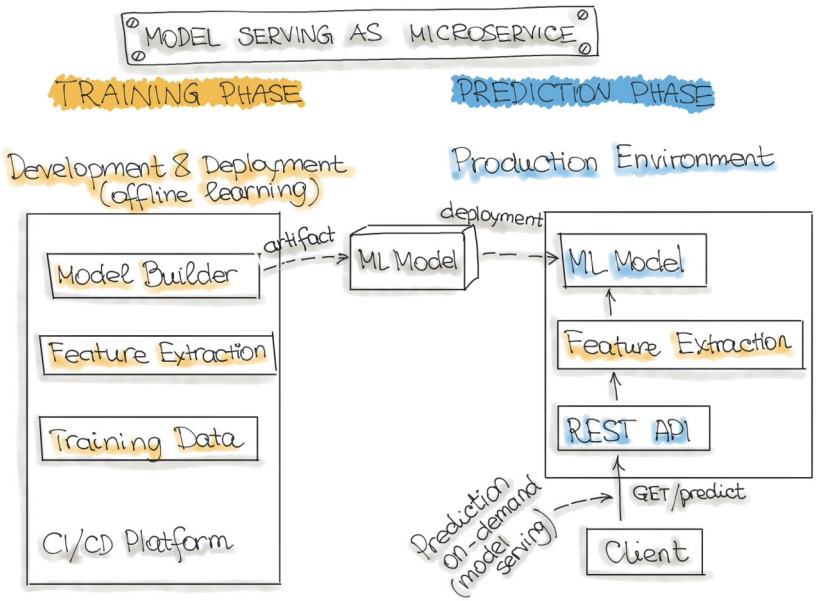

Figure 3.2: Model Serving as Micro Service (source: ml-ops.org)

Online Learning

The most dynamic way to embed machine learning into a production system is to implement *online learning*, which is also known as *real-time streaming analytics*. Please note that online learning can be a confusing name because the core learning or ML model training is usually not performed on the live system. We should call it *incremental learning*; however, the term *online learning* is already established within the ML community.

In this type of ML workflow, the ML learning algorithm is continuously receiving a data stream, either as single data points or in small groups called mini-batches. The system learns about new data on the fly as it arrives, so the ML model is incrementally being re-trained with new data. This continually re-trained model is instantly available as a web service.

Technically, this type of ML system works well with the *lambda architecture* in big data systems. Usually, the input data is a stream of events, and the ML model takes the data as it enters the system, provides predictions and re-learns on these new data. The model would typically run as a service on a Kubernetes cluster or similar.

A big difficulty with the online learning system in production is that if bad data is entering the system, the ML model, as well as the whole system performance, will increasingly decline.

AutoML

An even more sophisticated version of online learning is *automated machine learning* or *AutoML*.

AutoML is getting a lot of attention and is considered the next advance for enterprise ML. AutoML promises training ML models with minimal effort and without machine learning expertise. The user needs to provide data, and the AutoML system automatically selects an ML algorithm, such as neural network architecture, and configures the selected algorithm.

Instead of updating the model, we execute an entire ML model training pipeline in production that results in new models on the fly. For now, this is a very

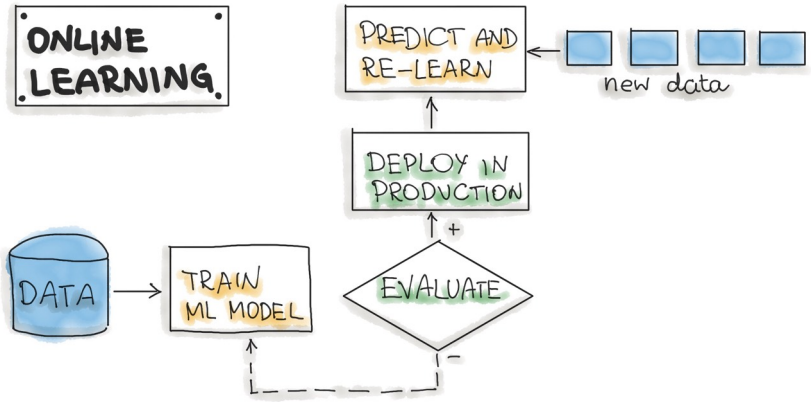

Figure 3.3: Online Learning (source: ml-ops.org)

experimental way to implement ML workflows. AutoML is usually provided by big cloud providers, such as Google[7] or MS Azure[8]. However, models build with AutoML need to reach the level of accuracy required for real-world success.

Further reading

- AutoML: Overview and Tools[9]
- AutoML Benchmark[10]

3.2.6 ML Model serialization formats

There are various formats to distribute ML models. In order to achieve a distributable format, the ML model should be present and should be executable as an independent asset. For example, we might want to use a Scikit-learn model in

[7] https://cloud.google.com/automl/

[8] https://docs.microsoft.com/en-us/azure/machine-learning/concept-automated-ml

[9] https://www.automl.org/automl/

[10] https://www.researchgate.net/profile/Marc_Andre_Zoeller/publication/332750780_Benchmark_and_Survey_of_Automated_Machine_Learning_Frameworks/links/5e15bd1792851c8364baa47a/Benchmark-and-Survey-of-Automated-Machine-Learning-Frameworks.pdf

a Spark job. This means that the ML models should work outside of the model-training environment. In the following, we describe *Language-agnostic* and *Vendor-specific exchange formats* for ML models.

Language-agnostic exchange formats

- Amalgamation is the simplest way to export an ML model. The model and all necessary code to run are bundled as one package. Usually, it is a single source code file that can be compiled on nearly any platform as a standalone program. For example, we can create a standalone version of an ML model by using SKompiler[11]. This python package provides a tool for transforming trained Scikit-learn models into other forms, such as SQL queries, Excel formulas, Portable Format for Analytics (PFA) files, or SymPy expressions. The last can be translated to code in a variety of languages, such as C, Javascript, Rust, Julia, etc. Amalgamation is a straightforward concept, and the exported ML models are portable. With some easy ML algorithms, such as logistic regression or decision tree, this format is compact and might have good performance, which is useful for constrained embedded environments. However, the ML model code and parameters need to be managed together.
- PMML is a format for model serving based on XML with the file extension .pmml. PMML has been standardized by the Data Mining Group (DMG)[12]. Basically, .ppml describes a model and pipeline in XML[13]. The PMML supports not all of the ML algorithms, and its usage in open source-driven tools is limited due to licensing issues.
- PFA (Portable Format for Analytics[14]) is designed as a replacement for PMML. From DMG: *"A PFA document is a string of JSON-formatted text that describes an executable called a scoring engine. Each engine has a well-defined input, a well-defined output, and functions for combining inputs to construct the output in an expression-centric syntax tree"*. PFA capabilities include (1) control structures, such as conditionals, loops, and user-defined functions, (2) expressed within

[11] https://pypi.org/project/SKompiler/
[12] http://dmg.org/dmg-members.html
[13] http://dmg.org/pmml/pmml_examples/
[14] http://dmg.org/pfa/docs/motivation/

JSON, and can, therefore, be easily generated and manipulated by other programs, (3) fine-grained function library supporting extensibility callbacks. To run ML models as PFA files, we will need a PFA-enabled environment.

- ONNX (Open Neural Network eXchange) is an ML framework independent file format. ONNX was created to allow any ML tool to share a single model format. This format is supported by many big tech companies such as Microsoft, Facebook, and Amazon. Once the ML model is serialized in the ONNX format, it can be consumed by onnx-enabled runtime libraries (also called inference engines) and then make predictions. Here[15] you will find the list of tools that can use ONNX format. Notably that most deep learning tools have ONNX support.

Source: Open Standard Models[16]

Vendor-specific exchange formats

- Scikit-Learn saves models as pickled python objects, with a .pkl file extension.
- H2O allows you to convert the models you have built to either POJO (Plain Old Java Object) or MOJO (Model Object, Optimized).
- SparkML models that can be saved in the MLeap file format and served in real-time using an MLeap model server. The MLeap runtime is a JAR that can run in any Java application.MLeap supports Spark, Scikit-learn, and Tensorflow for training pipelines and exporting them to an MLeap Bundle.
- TensorFlow saves models as .pb files; which is the protocol buffer files extension.
- PyTorch serves models by using their proprietary Torch Script as a .pt file. Their model format can be served from a C– application.
- Keras saves a model as a .h5 file, which is known in the scientific community as a data file saved in the Hierarchical Data Format (HDF). This type of file contains multidimensional arrays of data.
- Apple has its proprietary file format with the extension .mlmodel to store models embedded in iOS applications. The Core ML framework has native support for Objective-C and Swift programming languages. Applications trained

[15] https://github.com/onnx/tutorials#scoring-onnx-models
[16] https://github.com/adbreind/open-standard-models-2019

in other ML frameworks, such as TensorFlow, Scikit-Learn, and other frameworks need to use tools like such as coremltools and Tensorflow converter to translate their ML model files to the .mlmodel format for use on iOS.

The following Table summarizes all ML model serialization formats:

	Open-Format	Vendor	File Extension	License	ML Tools & Platforms Support	Human-readable	Compression
"almagination"	–	–	–	–	–	–	✓
PMML	✓	DMG	.pmml	AGPL	R, Python, Spark	✓ (XML)	✗
PFA	✓	DMG	JSON		PFA-enabled runtime	✓ (JSON)	✗
ONNX	✓	SIG, LFAI	.onnx		TF, CNTK, Core ML, MXNet, ML.NET	–	✓
TF Serving Format	✓	Google	.pf		Tensor Flow	✗	g-zip
Pickle Format	✓		.pkl		scikit-learn	✗	g-zip
JAR/ POJO	✓		.jar		H2O	✗	✓
HDF	✓		.h5		Keras	✗	✓
MLEAP	✓		.jar/ .zip		Spark, TF, scikit-learn	✗	g-zip
Torch Script	✗		.pt		PyTorch	✗	✓
Apple .mlmodel	✗	Apple	.mlmodel		TensorFlow, scikit-learn, Core ML	–	✓

Further reading:

- ML Models training file formats[17]

[17] https://towardsdatascience.com/guide-to-file-formats-for-machine-learning-columnar-training-inferencing-and-the-feature-store-2e0c3d18d4f9

- Open Standard Models[18]

3.3 Code: Deployment Pipelines

The final stage of delivering an ML project includes the following three steps:

1. *Model Serving* - The process of deploying the ML model in a production environment.
2. *Model Performance Monitoring* - The process of observing the ML model performance based on live and previously unseen data, such as prediction or recommendation. In particular, we are interested in ML-specific signals, such as prediction deviation from previous model performance. These signals might be used as triggers for model re-training.
3. *Model Performance Logging* - Every inference request results in a log-record.

In the following, we discuss *Model Serving Patterns* and *Model Deployment Strategies*.

3.3.1 Model Serving Patterns

Three components should be considered when we serve an ML model in a production environment. The *inference* is the process of getting data to be ingested by a model to compute *predictions*. This process requires a *model*, an *interpreter* for the execution, and *input data*.

Deploying an ML system to a production environment includes two aspects, first deploying the pipeline for automated retraining and ML model deployment. Second, providing the API for prediction on unseen data.

[18] https://github.com/adbreind/open-standard-models-2019

Model serving is a way to integrate the ML model in a software system. We distinguish between five patterns to put the ML model in production: ***Model-as-Service***, ***Model-as-Dependency***, ***Precompute***, ***Model-on-Demand***, and ***Hybrid-Serving***. Please note that the above-described model serialization formats might be used for any of the model serving patterns.

The following taxonomy shows these approaches:

<table>
<tr><th colspan="4">Machine Learning Model Serving Taxonomy</th></tr>
<tr><td></td><td colspan="3">ML Model</td></tr>
<tr><td>Service & Versioning</td><td>Together with the consuming application</td><td colspan="2">Independent from the consuming application</td></tr>
<tr><td>Compile/ Runtime Availabilty</td><td>Build & runtime available</td><td>Available remotely through REST API/RPC</td><td>Available at the runtime scope</td></tr>
<tr><td rowspan="2">Serving Patterns</td><td>"Model-as-Dependency"</td><td>"Model-as-Service"</td><td>"Precompute" and "Model on Demand"</td></tr>
<tr><td colspan="2">Hybrid Model Serving (Federated Learning)</td><td></td></tr>
</table>

Now, we present the serving patterns to productionize the ML model such as ***Model-as-Service***, ***Model-as-Dependency***, ***Precompute***, ***Model-on-Demand***, and ***Hybrid-Serving***.

Model-as-Service

Model-as-Service is a common pattern for wrapping an ML model as an independent service. We can wrap the ML model and the interpreter within a dedicated web service that applications can request through a REST API or consume as a gRPC service.

This pattern can be used for various ML workflows, such as Forecast, Web Service, Online Learning.

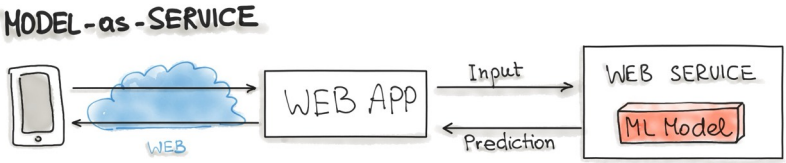

Figure 3.4: Model as Service (source: https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ch02.html#project_chapter)

Model-as-Dependency

Model-as-Dependency is probably the most straightforward way to package an ML model. A packaged ML model is considered as a dependency within the software application. For example, the application consumes the ML model like a conventional *jar* dependency by invoking the prediction method and passing the values. The return value of such method execution is some prediction that is performed by the previously trained ML model. The Model-as-Dependency approach is mostly used for implementing the Forecast pattern.

Precompute Serving Pattern

This type of ML model serving is tightly related to the *Forecast* ML workflow. With the *Precompute* serving pattern, we use an already trained ML model and precom-

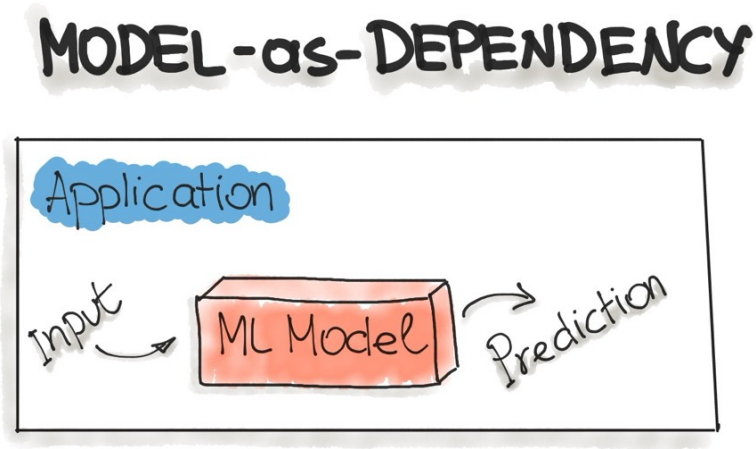

Figure 3.5: Model as Dependency (source: ml-ops.org)

pute the predictions for the incoming batch of data. The resulting predictions are persisted in the database. Therefore, for any input request, we query the database to get the prediction result.

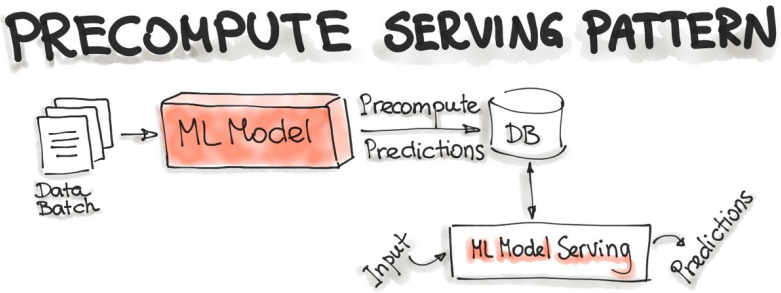

Figure 3.6: Precompute Serving Pattern (source: ml-ops.org)

For further reading we recommend: Bringing ML to Production (Slides)[19]

Model-on-Demand

The Model-on-Demand pattern also treats the ML model as a dependency that is available at runtime. This ML model, contrary to the Model-as-Dependency pattern, has its own release cycle and is published independently.

The *message-broker* architecture is typically used for such on-demand model serving. The *message-broker* topology architecture pattern contains two main types of architecture components: a *broker* component and an *event processor* component. The broker component is the central part that contains the event channels that are utilised within the event flow. The event channels, which are enclosed in the broker component, are message queues. We can imagine such architecture containing *input-* and *output-queues*. A message broker allows one process to write *prediction-requests* in an input queue. The *event processor* contains the model serving runtime and the ML model. This process connects to the broker, reads these requests in batch from the queue and sends them to the model to make the predictions. The model serving process runs the prediction generation on the input data and writes the resulted predictions to the output queue. Afterwards, the queued prediction results are pushed to the prediction service that initiated the prediction request.

Further reading:

- Event-driven architecture[20]
- Web services vs. streaming for real-time machine learning endpoints[21]

[19] https://www.slideshare.net/mikiobraun/bringing-ml-to-production-what-is-missing-amld-2020

[20] https://learning.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

[21] https://towardsdatascience.com/web-services-vs-streaming-for-real-time-machine-learning-endpoints-c08054e2b18e

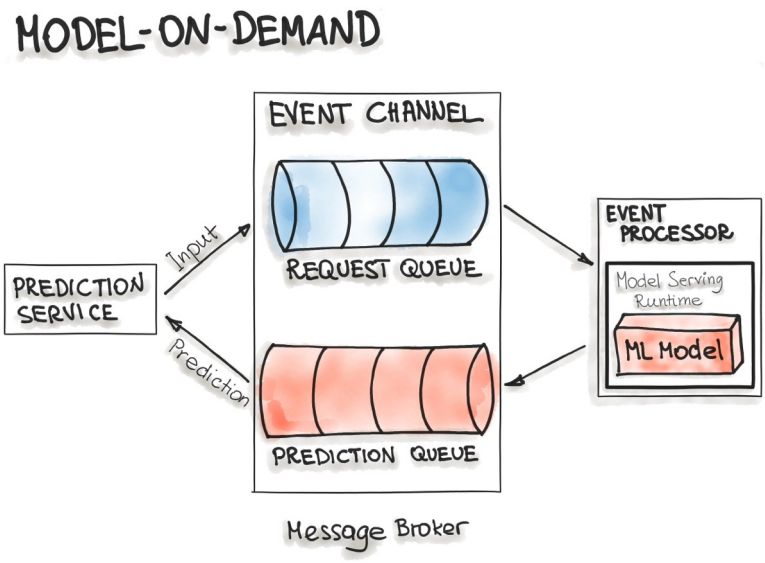

Figure 3.7: Model on Demand (source: ml-ops.org)

Hybrid-Serving (Federated Learning)

Federated Learning, also known as *hybrid-serving*, is another way of serving a model to the users. It is unique in the way it does, there is not only one model that predicts the outcome, but there are also lots of it. Exactly spoken there are as many models as users exist, in addition to the one that's held on a server. Let us start with the *unique* model, the one on the server. The model on the server-side is trained only once with the real-world data. It sets the initial model for each user. Also, it is a relatively general trained model so it fits for the majority of users. On the other side, there are the user-side models, which are the real unique models. Due to the raising hardware standards on mobile devices, it is possible for the devices to train their own models. Like that the devices will train their own highly specialized model for their own user. Once in a while, the devices send their already trained model data (not the personal data) to the server. There the server model will be adjusted, so the actual trends of the whole user community

will be covered by the model. This model is set to be the new initial model that all devices are using. For not having any downsides for the users, while the server model gets updated, this happens only when the device is idle, connected to WiFi and charging. Also, the testing is done on the devices, therefore the newly adopted model from the server is sent to the devices and tested for functionality.

The big benefit of this is that the data used for training and testing, which is highly personal, never leaves the devices while still capturing all data that is available. This way it is possible to train highly accurate models while not having to store tons of (probably personal) data in the cloud. But there is no such thing as a free lunch, normal machine learning algorithms are built with homogeneously and large datasets on powerful hardware which is always available for training. With Federated Learning there are other circumstances, the mobile devices are less powerful, the training data is distributed across millions of devices and these are not always available for training. Exactly for this TensorFlow Federated (TFF[22]) has been created. TFF is a lightweight form of TensorFlow created for Federated Learning.

3.3.2 Deployment Strategies

In the following, we discuss common ways for wrapping trained models as deployable services, namely deploying ML models as *Docker Containers to Cloud Instances* and as *Serverless Functions*.

Deploying ML Models as Docker Containers

As of now, there is no standard, open solution to ML model deployment. As ML model inference being considered *stateless*, *lightweight*, and *idempotent*, containerization becomes the de-facto standard for delivery. This means we deploy a container that wraps an ML model inference code. For on-premise, cloud, or hybrid deployments, Docker is considered to be de-facto standard containerization technology.

[22] https://medium.com/tensorflow/introducing-tensorflow-federated-a4147aa20041

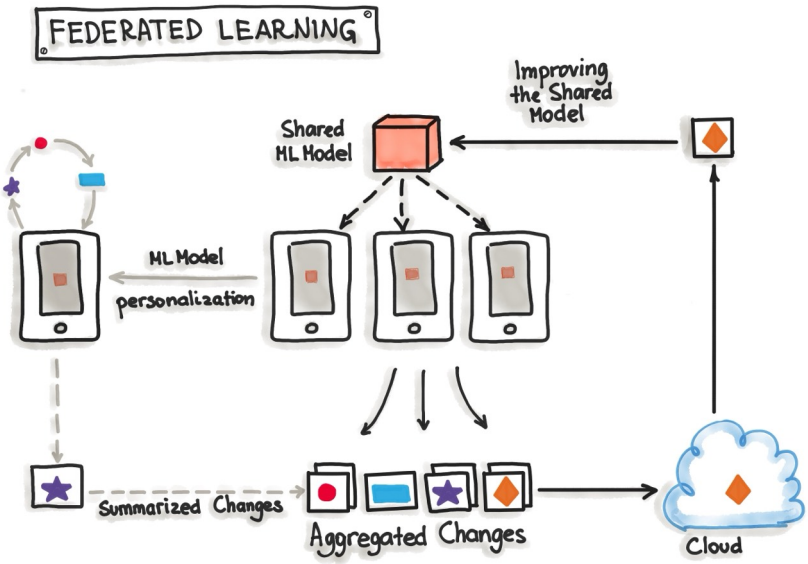

Figure 3.8: Federated Learning (source: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html)

One ubiquitous way is to package the whole ML tech stack (dependencies) and the code for ML model prediction into a Docker container. Then Kubernetes or an alternative (e.g. AWS Fargate) does the orchestration. The ML model functionality, such as prediction, is then available through a REST API (e.g. implemented as Flask application[23])

Deploying ML Models as Serverless Functions

Various cloud vendors already provide machine-learning platforms, and you can deploy your model with their services. Examples are Amazon AWS Sagemaker, Google Cloud AI Platform, Azure Machine Learning Studio, and IBM Watson Machine Learning, to name a few. Commercial cloud services also provide con-

[23] https://flask.palletsprojects.com/en/1.1.x/

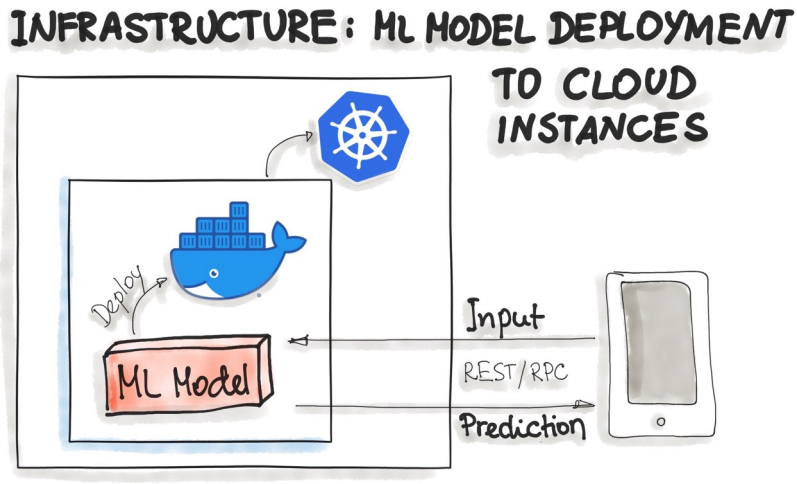

Figure 3.9: Infrastructure: ML Model Deployment to Cloud Instances (source: ml-ops.org)

tainerization of ML models such as AWS Lambda and Google App Engine servlet host.

In order to deploy an ML model as a serverless function, the application code and dependencies are packaged into .zip files, with a single entry point function. This function then could be managed by major cloud providers such as Azure Functions, AWS Lambda, or Google Cloud Functions. However, attention should be paid to possible constraints of the deployed artifacts such as the size of the artifact.

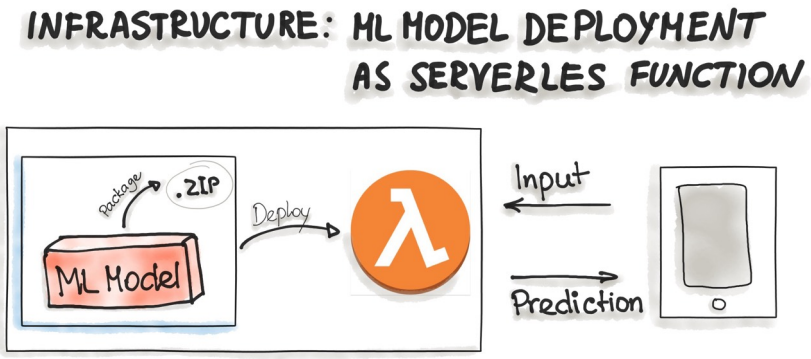

Figure 3.10: Infrastructure: ML Model Deployment as Serverless Function (source: ml-ops.org)

4 MLOps Principles

As machine learning and AI propagate in software products and services, we need to establish best practices and tools to test, deploy, manage, and monitor ML models in real-world production. In short, with MLOps, we strive to avoid "*technical debt*" in machine learning applications.

SIG MLOps defines "*an optimal MLOps experience [as] one where Machine Learning assets are treated consistently with all other software assets within a CI/CD environment. Machine Learning models can be deployed alongside the services that wrap them and the services that consume them as part of a unified release process.*" By codifying these practices, we hope to accelerate the adoption of ML/AI in software systems and fast delivery of intelligent software.

In the following, we describe a set of important concepts in MLOps such as *Iterative-Incremental Development, Automation, Continuous Deployment, Versioning, Testing, Reproducibility, and Monitoring*.

4.1 Iterative-Incremental Process in MLOps

The complete MLOps process includes three broad phases of "*Designing the ML-powered application*", "*ML Experimentation and Development*", *and* "*ML Operations*".

The first phase is devoted to *business understanding, data understanding* and *designing the ML-powered software*. In this stage, we identify our potential user, design the machine learning solution to solve its problem, and assess the further development of the project. Mostly, we would act within two categories of problems - either increasing the productivity of the user or increasing the interactivity of our application.

Initially, we define ML use-cases and prioritize them. The best practice for ML projects is to work on one ML use case at a time. Furthermore, the *design* phase aims to inspect the available data that will be needed to train our model and to specify the functional and non-functional requirements of our ML model. We

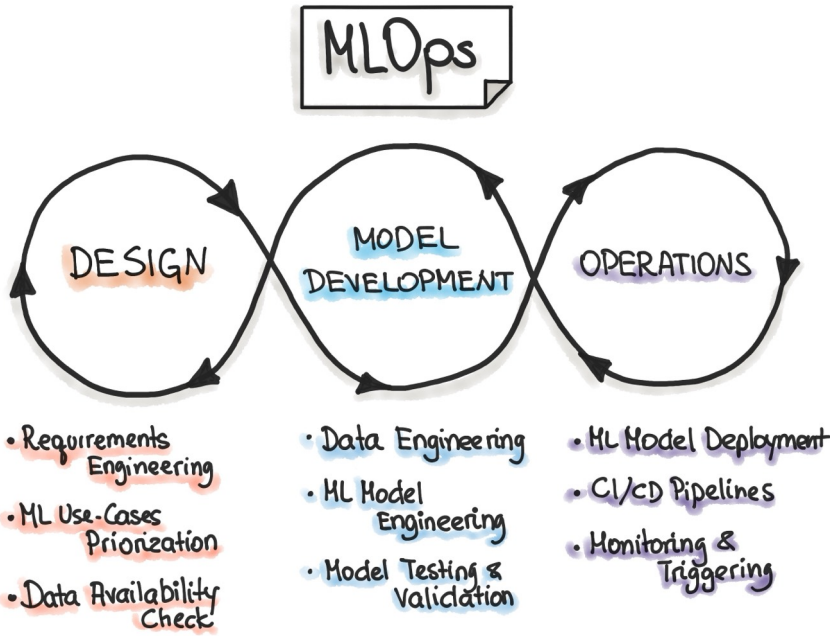

Figure 4.1: Three broad phases of the Iterative-Incremental Process in MLOps

should use these requirements to design the architecture of the ML-application, establish the serving strategy, and create a test suite for the future ML model.

The follow-up phase "*ML Experimentation and Development*" is devoted to verifying the applicability of ML for our problem by implementing *Proof-of-Concept for ML Model*. Here, we run iteratively different steps, such as *identifying or polishing the suitable ML algorithm for our problem, data engineering*, and *model engineering*. The primary goal in this phase is to deliver a stable quality ML model that we will run in production.

The main focus of the "*ML Operations*" phase is to deliver the previously developed ML model in production by using established DevOps practices such as testing, versioning, continuous delivery, and monitoring.

All three phases are interconnected and influence each other. For example, the design decision during the design stage will propagate into the experimentation

phase and finally influence the deployment options during the final operations phase.

4.2 Automation

The level of automation of the Data, ML Model, and Code pipelines determines the maturity of the ML process. With increased maturity, the velocity for the training of new models is also increased. The objective of an MLOps team is to automate the deployment of ML models into the core software system or as a service component. This means, to automate the complete ML-workflow steps without any manual intervention. Triggers for automated model training and deployment can be calendar events, messaging, monitoring events, as well as changes on data, model training code, and application code.

Automated testing helps discover problems quickly and in early stages. This enables fast fixing of errors and learning from mistakes.

To adopt MLOps, we see *three levels of automation*, starting from the initial level with manual model training and deployment, up to running both ML and CI/CD pipelines automatically.

1. **Manual process.**
   This is a typical data science process, which is performed at the beginning of implementing ML. This level has an experimental and iterative nature. Every step in each pipeline, such as data preparation and validation, model training and testing, are executed manually. The common way to process is to use Rapid Application Development (RAD) tools, such as Jupyter Notebooks.

2. **ML pipeline automation.**
   The next level includes the execution of model training automatically. We introduce here the continuous training of the model. Whenever new data is available, the process of model retraining is triggered. This level of automation also includes data and model validation steps.

3. **CI/CD pipeline automation.**
   In the final stage, we introduce a CI/CD system to perform fast and reliable ML model deployments in production. The core difference from the previous step is that we now automatically build, test, and deploy the Data, ML Model, and the ML training pipeline components.

The following picture shows the automated ML pipeline with CI/CD routines:

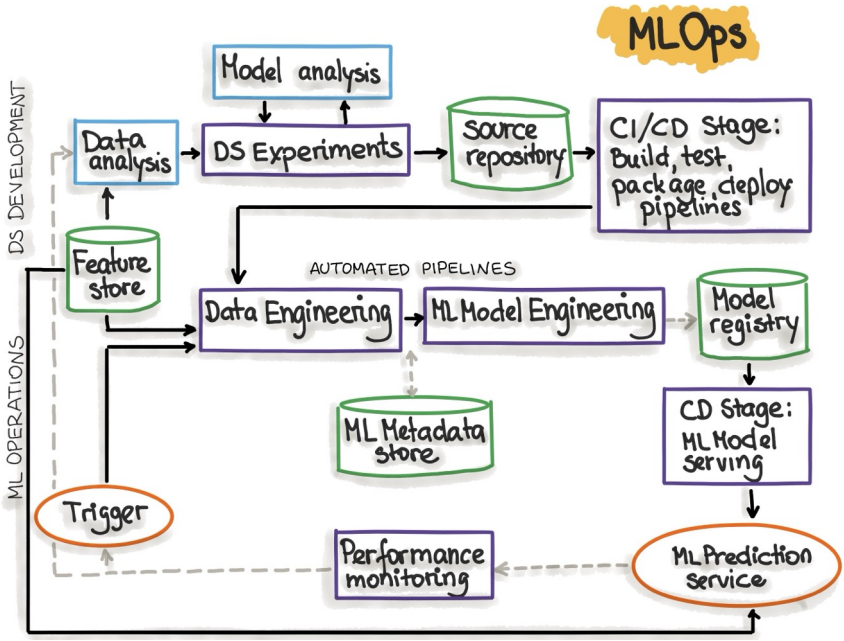

Figure 4.2: Automated ML pipeline with CI/CD (source: MLOps: Continuous delivery and automation pipelines in machine learning)

The MLOps stages that reflect the process of ML pipeline automation are explained in the following table:

MLOps Stage	Output of the Stage Execution
Development & Experimentation (ML algorithms, new ML models)	Source code for pipelines: Data extraction, validation, preparation, model training, model evaluation, model testing
Pipeline Continuous Integration (Build source code and run tests)	Pipeline components to be deployed: packages and executables.
Pipeline Continuous Delivery (Deploy pipelines to the target environment)	Deployed pipeline with new implementation of the model.
Automated Triggering (Pipeline is automatically executed in production. Schedule or trigger are used)	Trained model that is stored in the model registry.
Model Continuous Delivery (Model serving for prediction)	Deployed model prediction service (e.g. model exposed as REST API)
Monitoring (Collecting data about the model performance on live data)	Trigger to execute the pipeline or to start a new experiment cycle.

After analyzing the MLOps Stages, we might notice that the MLOps setup requires several components to be installed or prepared. The following table lists those components:

MLOps Setup Components	Description
Source Control	Versioning the Code, Data, and ML Model artifacts.
Test & Build Services	Using CI tools for (1) Quality assurance for all ML artifacts, and (2) Building packages and executables for pipelines.
Deployment Services	Using CD tools for deploying pipelines to the target environment.
Model Registry	A registry for storing already trained ML models.
Feature Store	Preprocessing input data as features to be consumed in the model training pipeline and during the model serving.
ML Metadata Store	Tracking metadata of model training, for example model name, parameters, training data, test data, and metric results.
ML Pipeline Orchestrator	Automating the steps of the ML experiments.

Further reading: "MLOps: Continuous delivery and automation pipelines in machine learning"[1]

[1] https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#top_of_page

4.3 Continuous X

To understand ***Model deployment***, we first specify the "ML assets" as ML model, its parameters and hyperparameters, training scripts, training and testing data. We are interested in the identity, components, versioning, and dependencies of these ML artifacts. The target destination for an ML artifact may be a (micro-) service or some infrastructure components. A deployment service provides orchestration, logging, monitoring, and notification to ensure that the ML models, code and data artifacts are stable.

MLOps is an ML engineering culture that includes the following practices: - **Continuous Integration** (**CI**) extends the testing and validating code and components by adding testing and validating data and models. - **Continuous Delivery** (**CD**) concerns with the delivery of an ML training pipeline that automatically deploys another ML model prediction service. - **Continuous Training** (**CT**) is unique to ML systems property, which automatically retrains ML models for re-deployment. - **Continuous Monitoring** (**CM**) concerns with monitoring production data and model performance metrics, which are bound to business metrics.

4.4 Versioning

The goal of *versioning* is to treat ML training scripts, ML models, and data sets for model training as first-class citizens in DevOps processes by tracking ML models and data sets with version control systems. The common reasons when ML model and data changes (according to SIG MLOps[2]) are the following:

- ML models can be retrained based on new training data.
- Models may be retrained based on new training approaches.
- Models may be self-learning.
- Models may degrade over time.
- Models may be deployed in new applications.
- Models may be subject to attack and require revision.
- Models can be quickly rolled back to a previous serving version.

[2] https://lists.cd.foundation/g/sig-mlops

- Corporate or government compliance may require audit or investigation on both ML model or data, hence we need access to all versions of the productionized ML model.
- Data may reside across multiple systems.
- Data may only be able to reside in restricted jurisdictions.
- Data storage may not be immutable.
- Data ownership may be a factor.

Analogous to the best practices for developing reliable software systems, every ML model specification (ML training code that creates an ML model) should go through a code review phase. Furthermore,

every ML model specification should be versioned in a VCS to make the training of ML models auditable and reproducible.

Further reading: How do we manage ML models? Model Management Frameworks[3]

4.5 Experiments Tracking

Machine Learning development is a highly iterative and research-centric process. In contrast to the traditional software development process, in ML development, multiple experiments on model training can be executed in parallel before making the decision what model will be promoted to production.

The experimentation during ML development might have the following scenario: One way to track multiple experiments is to use different (Git-) branches, each dedicated to a separate experiment. The output of each branch is a trained model. Depending on the selected metric, the trained ML models are compared with each other, and the appropriate model is selected. Such low friction branching is fully supported by the tool DVC[4], which is an extension of Git and an open-source version control system for machine learning projects. Another popular tool

[3] https://www.inovex.de/blog/machine-learning-model-management/
[4] https://dvc.org/

for ML experiments tracking is the Weights and Biases (wandb)[5] library, which automatically tracks the hyperparameters and metrics of the experiments.

4.6 Testing

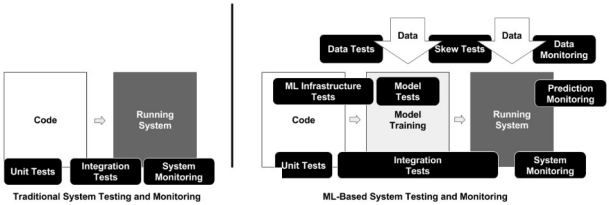

Figure 1. **ML Systems Require Extensive Testing and Monitoring.** The key consideration is that unlike a manually coded system (left), ML-based system behavior is not easily specified in advance. This behavior depends on dynamic qualities of the data, and on various model configuration choices.

Figure 4.3: ML Systems require extensive Testing and Monitoring (source: "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction" by E.Breck et al. 2017)

6

The complete development pipeline includes three essential components, **data pipeline**, **ML model pipeline**, and **application pipeline**. In accordance with this separation, we distinguish three scopes for testing in ML systems: **tests for features and data**, **tests for model development**, and **tests for ML infrastructure**.

4.6.1 Features and Data Tests

- Data validation: Automatic check for data and features schema/domain.
  - Action: In order to build a schema (domain values), calculate statistics from the training data. This schema can be used as *expectation definition* or *semantic role* for input data during training and serving stages.

[5] https://www.wandb.com/

[6] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/aad9f93b86b7addfea4c419b9100c6cdd26cacea.pdf

- Features importance test to understand whether new features add a predictive power.
  - Action: Compute correlation coefficient on features columns.
  - Action: Train model with one or two features.
  - Action: Use the subset of features "One of *k* left out and train a set of different models.
  - Measure data dependencies, inference latency, and RAM usage for each new feature. Compare it with the predictive power of the newly added features.
  - Drop out unused/deprecated features from your infrastructure and document it.
- Features and data pipelines should be policy-compliant (e.g. GDPR). These requirements should be programmatically checked in both development and production environments.
- Feature creation code should be tested by unit tests (to capture bugs in features).

4.6.2 Tests for Reliable Model Development

We need to provide specific testing support for detecting ML-specific errors.

- Testing ML training should include routines, which verify that algorithms make decisions aligned to business objective.
  This means that ML algorithm loss metrics (MSE, log-loss, etc.) should correlate with business impact metrics (revenue, user engagement, etc.)
  - Action: The loss metrics - impact metrics relationship can be measured in small scale A/B testing using an intentionally degraded model.
  - Further reading: Selecting the Right Metric for evaluating Machine Learning Models. here 1[7], here 2[8]

[7] https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4

[8] https://medium.com/usf-msds/choosing-the-right-metric-for-evaluating-machine-learning-models-part-2-86d5649a5428

- Model staleness test.
  The model is defined as *stale* if the trained model does not include up-to-date data and/or does not satisfy the business impact requirements.
  Stale models can affect the quality of prediction in intelligent software.
  - Action: A/B experiment with older models. Including the range of ages to produce an *Age vs. Prediction Quality* curve to facilitate the understanding of how often the ML model should be trained.
- Assessing the cost of more sophisticated ML models.
  - Action: ML model performance should be compared to the simple baseline ML model (e.g. linear model vs neural network).
- Validating performance of a model.
  - It is recommended to separate the teams and procedures collecting the training and test data to remove the dependencies and avoid false methodology propagating from the training set to the test set (source[9]).
  - Action: Use an additional test set, which is disjoint from the training and validation sets. Use this *test set* only for a final evaluation.
- Fairness/Bias/Inclusion testing for the ML model performance.
  - Action: Collect more data that includes potentially under-represented categories.
  - Action: Examine input features if they correlate with protected user categories.
  - Further reading: "Tour of Data Sampling Methods for Imbalanced Classification"[10]
- Conventional unit testing for any feature creation, ML model specification code (training), and testing.
- Model governance testing (coming soon)

[9] https://arxiv.org/pdf/2003.05155.pdf
[10] https://machinelearningmastery.com/data-sampling-methods-for-imbalanced-classification/

4.6.3 ML infrastructure test

- Training the ML models should be reproducible, which means that training the ML model on the same data should produce identical ML models.
  - Diff-testing of ML models relies on deterministic training, which is hard to achieve due to non-convexity of the ML algorithms, random seed generation, or distributed ML model training.
  - Action: determine the non-deterministic parts in the model training code base and try to minimize non-determinism.
- Test ML API usage. Stress testing.
  - Action: Unit tests to randomly generate input data and train the model for a single optimization step (e.g., gradient descent).
  - Action: Crash tests for model training. The ML model should restore from a checkpoint after a mid-training crash.
- Test the algorithmic correctness.
  - Action: Unit test that is not intended to complete the ML model training but to train for a few iterations and ensure that loss decreases while training.
  - Avoid: Diff-testing with previously built ML models because such tests are hard to maintain.
- Integration testing: The full ML pipeline should be integration tested.
  - Action: Create a fully automated test that regularly triggers the entire ML pipeline.
    The test should validate that the data and code successfully finish each stage of training and the resulting ML model performs as expected.
  - All integration tests should be run before the ML model reaches the production environment.
- Validating the ML model before serving it.
  - Action: Setting a threshold and testing for slow degradation in model quality over many versions on a validation set.
  - Action: Setting a threshold and testing for sudden performance drops in a new version of the ML model.

- ML models are *canaried* before serving.
  - Action: Testing that an ML model successfully loads into production serving and the prediction on real-life data is generated as expected.
- Testing that the model in the training environment gives the same score as the model in the serving environment.
  - Action: The difference between the performance on the holdout data and the "nextday" data.
    Some difference will always exist.
    Pay attention to large differences in performance between holdout and "nextday" data because it may indicate that some time-sensitive features cause ML model degradation.
  - Action: Avoid result differences between training and serving environments.
    Applying a model to an example in the training data and the same example at serving should result in the same prediction.
    A difference here indicates an engineering error.

4.7 Monitoring

Once the ML model has been deployed, it needs to be monitored to ensure that the ML model performs as expected. The following checklist for model monitoring activities in production is adopted from "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction" by E.Breck et al. 2017:[11]

- Monitor dependency changes throughout the complete pipeline result in notification.
  - Data version change.
  - Changes in source system.
  - Dependencies upgrade.
- Monitor data invariants in training and serving inputs: Alert if data does not match the schema, which has been specified in the training step.

[11] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/aad9f93b86b7addfea4c419b9100c6cdd26cacea.pdf

- Action: tuning of alerting threshold to ensure that alerts remain useful and not misleading.

- Monitor whether training and serving features compute the same value.
  - Since the generation of training and serving features might take place in physically separated locations, we must carefully test that these different code paths are logically identical.
  - Action: (1) Log a sample of the serving traffic.
    (2) Compute distribution statistics (min, max, avg, values, % of missing values, etc.) on the training features and the sampled serving features and ensure that they match.

- Monitor the numerical stability of the ML model.
  - Action: trigger alerts for the occurrence of any NaNs or infinities.

- Monitor computational performance of an ML system.
  Both dramatic and slow-leak regression in computational performance should be notified.
  - Action: measure the performance of versions and components of code, data, and model by pre-setting the alerting threshold.
  - Action: collect system usage metrics like GPU memory allocation,

network traffic, and disk usage. These metrics are useful for *cloud costs* estimations. - Monitor how *stale* the system in production is. - Measure the *age* of the model. Older ML models tend to decay in performance. - Action: Model monitoring is a continuous process; therefore, it is important to identify the elements for monitoring and create a strategy for model monitoring before reaching production. - Monitor the processes of feature generation as they impact the model. - Action: rerun feature generation on a frequent basis. - Monitor degradation of the predictive quality of the ML model on served data.
Both dramatic and slow-leak regression in prediction quality should be notified. - Degradation might happen due to changes in data or differing code paths, etc. - Action: Measure statistical bias in predictions (avg in predictions in a slice of data).
Models should have nearly zero bias. - Action: If a label is available immediately

after the prediction is made, we can measure the quality of prediction in real-time and identify problems.

The picture below shows that the model monitoring can be implemented by tracking the precision, recall, and F1-score of the model prediction along with the time. The decrease in precision, recall, and F1-score triggers the model retraining, which leads to model recovery.

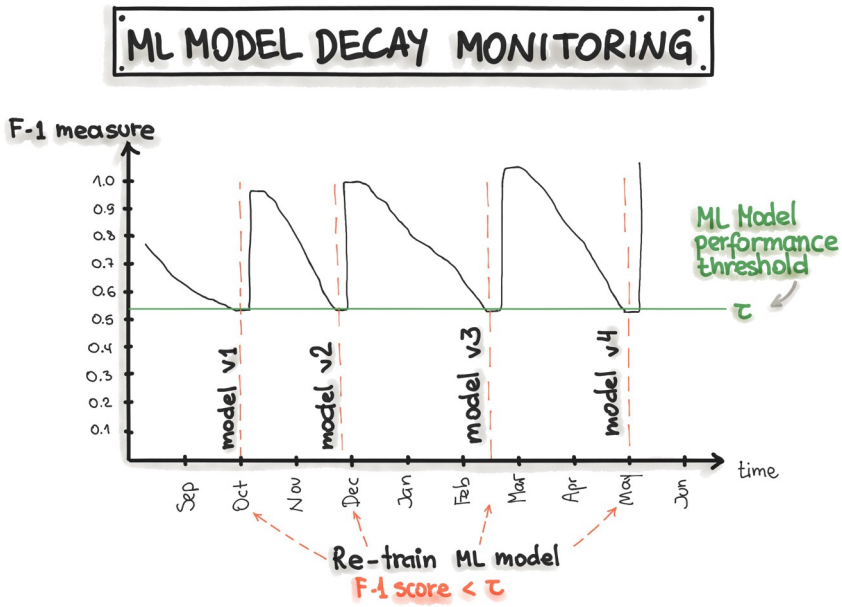

Figure 4.4: ML Model Decay Monitoring (source: ml-ops.org)

4.8 "ML Test Score" System

The "ML Test Score" measures the overall *readiness* of the ML system for production. The final **ML Test Score** is computed as follows:

- For each test, **half a point** is awarded for executing the test manually, with the results documented and distributed.
- A **full point** is awarded if there is a system in place to run that test automatically on a repeated basis.
- Sum the score of each of the four sections individually: Data Tests, Model Tests, ML Infrastructure Tests, and Monitoring.
- The final **ML Test Score** is computed by taking the minimum of the scores aggregated for each of the sections: Data Tests, Model Tests, ML Infrastructure Tests, and Monitoring.

After computing the **ML Test Score**, we can reason about the *readiness* of the ML system for production. The following table provides the interpretation ranges:

Points	Description
0	More of the research project than a productionized system.
(0,1]	Not totally untested, but it is worth considering the possibility of serious holes in reliability.
(1,2]	There has been a first pass at basic productionization, but additional investment may be needed.
(2,3]	Reasonably tested, but it is possible that more of those tests and procedures may be automated.
(3,5]	Strong level of automated testing and monitoring.
>5	Exceptional level of automated testing and monitoring.

Source: "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction" by E.Breck et al. 2017[12]

[12] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/aad9f93b86b7addfea4c419b9100c6cdd26cacea.pdf

4.9 Reproducibility

Reproducibility in a machine learning workflow means that every phase of either data processing, ML model training, and ML model deployment should produce identical results given the same input.

Phase	Challenges	How to Ensure Reproducibility
Collecting Data	Generation of the training data can't be reproduced (e.g due to constant database changes or data loading is random)	1) Always backup your data. 2) Saving a snapshot of the data set (e.g. on the cloud storage). 3) Data sources should be designed with timestamps so that a view of the data at any point can be retrieved. 4) Data versioning.
Feature Engineering	Scenarios: 1) Missing values are imputed with random or mean values. 2) Removing labels based on the percentage of observation. 3) Non-deterministic feature extraction methods.	1) Feature generation code should be taken under version control. 2) Require reproducibility of the previous step "Collecting Data".

Model Training / Model Build	Non-determinism	1) Ensure the order of features is always the same. 2) Document and automate feature transformation, such as normalization. 3) Document and automate hyperparameter selection. 4) For ensemble learning: document and automate the combination of ML models.
Model Deployment	1) Training the ML model has been performed with a software version that is different from the production environment. 2) The input data, which is required by the ML model is missing in the production environment.	1) Software versions and dependencies should match the production environment. 2) Use a container (Docker) and document its specification, such as image version. 3) Ideally, the same programming language is used for training and deployment.

4.10 Loosely Coupled Architecture (Modularity)

According to Gene Kim et al., in their book "Accelerate", *"high performance [in software delivery] is possible with all kinds of systems, provided that systems—and the teams that build and maintain them — are loosely coupled. This key architectural property enables teams to easily test and deploy individual components or services even as the organization and the number of systems it operates grow—that is, it allows organizations to increase their productivity as they scale."*

Additionally, Gene Kim et al., recommend to *"use a loosely coupled architecture. This affects the extent to which a team can test and deploy their applications on demand, without requiring orchestration with other services. Having a loosely coupled architecture allows your teams to work independently, without relying on other teams for support and services, which in turn enables them to work quickly and deliver value to the organization."*

Regarding ML-based software systems, it can be more difficult to achieve *loose coupling* between machine learning components than for traditional software components. ML systems have weak component boundaries in several ways. For example, the outputs of ML models can be used as the inputs to another ML model and such interleaved dependencies might affect one another during training and testing.

Basic modularity can be achieved by structuring the machine learning project. To set up a standard project structure, we recommend using dedicated templates such as

- Cookiecutter Data Science Project Template[13]
- The Data Science Lifecycle Process Template[14]
- PyScaffold[15]

[13] https://drivendata.github.io/cookiecutter-data-science/
[14] https://github.com/dslp/dslp-repo-template
[15] https://github.com/pyscaffold/pyscaffold

4.11 ML-based Software Delivery Metrics (4 metrics from "Accelerate")

In the most recent study on the state of DevOps[16], the authors emphasized four key metrics that capture the effectiveness of the software development and delivery of elite/high performing organizations: *Deployment Frequency, Lead Time for Changes, Mean Time To Restore*, and *Change Fail Percentage*. These metrics have been found useful[17] to measure and improve one's ML-based software delivery. In the following list, we give the definition of each of the metrics and make the connection to MLOps.

Deployment Frequency

- **DevOps**:
  - Question: "How often does your organization deploy code to production or release it to end-users?"
- **MLOps**:
  - Question: "ML Model Deployment Frequency depends on:"
    1. Model retraining requirements (ranging from less frequent to online training). Two aspects are crucial for model retraining: 1.1. Model decay metric. 1.2. New data availability.
    2. The level of automation of the deployment process, which might range between manual deployment and fully automated CI/CD pipeline.

Lead Time for Changes

- **DevOps**:
  - Question: "How long does it take to go from code committed to code successfully running in production?"

[16] https://services.google.com/fh/files/misc/state-of-devops-2019.pdf
[17] https://www.thoughtworks.com/radar/techniques/four-key-metrics

- **MLOps**:
  - Question: "ML Model Lead Time for Changes depends on:"
    1. Duration of the explorative phase in Data Science to finalize the ML model for deployment/serving.
    2. Duration of the ML model training.
    3. The number and duration of manual steps during the deployment process.

Mean Time To Restore (MTTR)

- **DevOps**:
  - Question: "How long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?"
- **MLOps**:
  - Question: "ML Model MTTR depends on:"
    - The number and duration of manually performed model debugging and model deployment steps.
    - In cases where the ML model should be retrained, MTTR also depends on the duration of the ML model training.
    - Alternatively, MTTR refers to the duration of the rollback of the ML model to the previous version.

Change Failure Rate

- **DevOps**:
  - Question: "What percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?"

- **MLOps**:
  - Question: "ML Model Change Failure Rate can be expressed in the difference between the currently deployed ML model performance metrics and the previous model's metrics, such as Precision, Recall, F1, accuracy, AUC, ROC, false positives, etc. ML Model Change Failure Rate is also related to A/B testing."

To improve the effectiveness of the ML development and delivery process, one should measure the above four key metrics. A practical way to achieve such effectiveness is to implement the CI/CD pipeline first and adopt test-driven development for Data, ML Model, and Software Code pipelines.

4.12 Summary of MLOps Principles and Best Practices

The complete ML development pipeline includes three levels where changes can occur: ***Data***, ***ML Model***, and ***Code***. This means that in machine learning-based systems, the trigger for a build might be the combination of a code change, data change, or model change. The following list summarizes the MLOps principles for building ML-based software:

Versioning

- **Data**:
  1. Data preparation pipelines
  2. Features store
  3. Datasets
  4. Metadata

- **ML Model**:
    1. ML model training pipeline
    2. ML model (object)
    3. Hyperparameters
    4. Experiment tracking
- **Code**:
    1. Application code
    2. Configurations

Testing

- **Data**:
    1. Data Validation (error detection)
    2. Feature creation unit testing
- **ML Model**:
    1. Model specification is unit tested
    2. ML model training pipeline is integration tested
    3. ML model is validated before being operationalized
    4. ML model staleness test (in production)
    5. Testing ML model relevance and correctness
    6. Testing non-functional requirements (security, fairness, interpretability)
- **Code**:
    1. Unit testing
    2. Integration testing for the end-to-end pipeline

Automation

- **Data**:
    1. Data transformation
    2. Feature creation and manipulation

- **ML Model**:
    1. Data engineering pipeline
    2. ML model training pipeline
    3. Hyperparameter/Parameter selection
- **Code**:
    1. ML model deployment with CI/CD
    2. Application build

Reproducibility

- **Data**:
    1. Backup data
    2. Data versioning
    3. Extract metadata
    4. Versioning of feature engineering
- **ML Model**:
    1. Hyperparameter tuning is identical between dev and prod
    2. The order of features is the same
    3. Ensemble learning: the combination of ML models is the same
    4. The model pseudo-code is documented
- **Code**:
    1. Versions of all dependencies in dev and prod are identical
    2. Same technical stack for dev and production environments
    3. Reproducing results by providing container images or virtual machines

Deployment

- **Data**:
    1. Feature store is used in dev and prod environments

- **ML Model**:
    1. Containerization of the ML stack
    2. REST API
    3. On-premise, cloud, or edge
- **Code**:
    1. On-premise, cloud, or edge

Monitoring

- **Data**:
    1. Data distribution changes (training vs. serving data)
    2. Training vs. serving features
- **ML Model**:
    1. ML model decay
    2. Numerical stability
    3. Computational performance of the ML model
- **Code**:
    1. Predictive quality of the application on serving data

Along with the MLOps principles, following best practices should help reduce the "technical debt" of the ML project.

The table on the following page summarizes different best practices.

MLOps Best Practices	Data	ML Model	Code
Documentation	1) Data sources 2) Decisions, how/where to get data 3) Labeling methods	1) Model selection criteria 2) Design of experiments 3) Model pseudo-code	1) Deployment process 2) How to run locally
Project Structure	1) Data folder for raw and processed data 2) A folder for data engineering pipeline 3) Test folder for data engineering methods	1) A folder that contains the trained model 2) A folder for notebooks 3) A folder for feature engineering 4) A folder for ML model engineering	1) A folder for bash/shell scripts 2) A folder for tests 3) A folder for deployment files (e.g Docker files)

5 CRISP-ML(Q). The ML Lifecycle Process.

The machine learning community is still trying to establish a standard process model for machine learning development. As a result, many machine learning and data science projects are still not well organized. Results are not reproducible. In general, such projects are conducted in an ad-hoc manner. To guide ML practitioners through the development life cycle, the Cross-Industry Standard Process for the development of Machine Learning applications with Quality assurance methodology (CRISP-ML(Q))[1] was recently proposed. We review the core phases of the ML development process model (see Figure 1). There is a particular order of the individual stages. Still, machine learning workflows are fundamentally iterative and exploratory, so that depending on the results from the later phases, we might re-examine earlier steps.

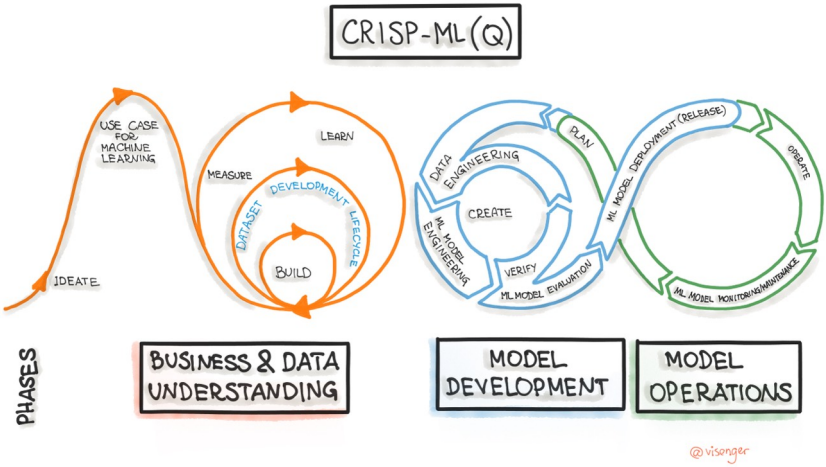

Figure 5.1: Machine Learning Development Life Cycle Process (source: ml-ops.org)

Overall, the CRISP-ML(Q) process model describes six phases:

[1] https://arxiv.org/pdf/2003.05155.pdf

1. Business and Data Understanding
2. Data Engineering (Data Preparation)
3. Machine Learning Model Engineering
4. Quality Assurance for Machine Learning Applications
5. Deployment
6. Monitoring and Maintenance.

For each phase of the process model (see Figure 2), the quality assurance approach in CRISP-ML(Q) requires the definition of requirements and constraints (e.g., performance, data quality requirements, model robustness, etc.), instantiation of the specific tasks (e.g., ML algorithm selection, model training, etc.), specification of risks that might negatively impact the efficiency and success of the ML application (e.g., bias, overfitting, lack of reproducibility, etc.), quality assurance methods to mitigate risks when these risks need to be diminished (e.g., cross-validation, documenting process and results, etc.).

In the following, we describe each of the six CRISP-ML(Q) stages with respect to the quality assurance structure.

5.1 Business and Data Understanding

Developing machine learning applications starts with identifying the scope of the ML application, the success criteria, and a data quality verification. The goal of this first phase is to ensure the feasibility of the project.

We gather success criteria along with business, machine learning, and economic success criteria during this phase. These criteria are required to be measurable. Therefore, defining clear and measurable Key Performance Indicators (KPI) such as *"time savings per user and session"* is required. A helpful approach is to define a non-ML heuristic benchmark[2] to communicate the impact of machine learning tasks with the business stakeholders.

Confirming the feasibility before setting up the ML project is a best practice in an industrial setting. Applying the Machine Learning Canvas[3] framework would be a

[2] https://learning.oreilly.com/library/view/machine-learning-design/9781098115777/ch08.html
[3] https://www.louisdorard.com/machine-learning-canvas

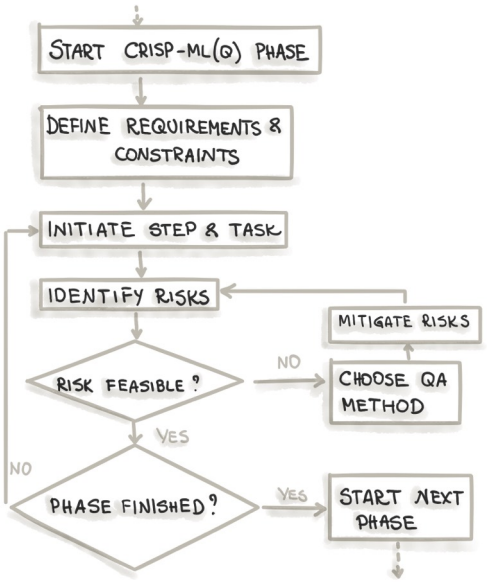

Figure 5.2: CRISP-ML(Q) approach for quality assurance for each of the six phases (source: ml-ops.org)

structured way to perform this task. The ML Canvas guides through the prediction and learning phases of the ML application. In addition, it enables all stakeholders to specify data availability, regulatory constraints, and application requirements such as robustness, scalability, explainability, and resource demand.

As data guides the process, data collection and data quality verification are essential to achieving business goals. Therefore, one crucial requirement is the documentation of the statistical properties of data and the data generating process. Similarly, data requirements should be stated and documented as well, as it becomes a foundation for data quality assurance during the operational phase of the ML project.

5.2 Data Engineering (Data Preparation)

The second phase of the CRISP-ML(Q) process model aims to prepare data for the following modeling phase. *Data selection, data cleaning, feature engineering,* and *data standardization* tasks are performed during this phase.

We identify valuable and necessary features for future model training by using either *filter methods, wrapper methods*, or *embedded methods* for data selection. Furthermore, we select data by discarding samples that do not satisfy data quality requirements. At this point, we also might tackle the problem of unbalanced classes by applying *over-sampling* or *under-sampling* strategies.

The data cleaning task implies that we perform error detection and error correction steps for the available data. Adding unit testing for data[4] will mitigate the risk of error propagation to the next phase. Depending on the machine learning task, we might need to perform feature engineering and data augmentation activities. For example, such methods include one-hot encoding, clustering, or discretization of continuous attributes.

The data standardization task denotes the process of unifying the ML tools' input data to avoid the risk of erroneous data. Finally, the normalization task will mitigate the risk of bias to features on larger scales. We build data and input data transformation pipelines for data pre-processing and feature creation to ensure the ML application's reproducibility during this phase.

5.3 Machine Learning Model Engineering

The modeling phase is the ML-specific part of the process. This phase aims to specify one or several machine learning models to be deployed in the production. The translation to the ML task depends on the business problem that we are trying to solve. Constraints and requirements from the *Business and Data Understanding* phase will shape this phase. For example, the application domain's model assessment metrics[5] might include *performance metrics, robustness, fairness,*

[4] https://ssc.io/pdf/p1993-schelter.pdf
[5] https://arxiv.org/pdf/1906.10742.pdf

scalability, *interpretability*, *model complexity degree*, and *model resource demand*. We should adjust the importance of each of these metrics according to the use case.

Generally, the modeling phase includes *model selection*, *model specialization*, and *model training* tasks. Additionally, depending on the application, we might use a pre-trained model, compress the model, or apply ensemble learning methods to get the final ML model.

One main complaint about machine learning projects is the lack of reproducibility. Therefore we should ensure that the method and the results of the modeling phase are reproducible by collecting the model training method's metadata. Typically we collect the following metadata: algorithm, training, validation and testing data set, hyper-parameters, and runtime environment description. The result reproducibility assumes the validation of the model's mean performance on different random seeds. Following best practices, documenting trained models increases the transparency and explainability in ML projects. A helpful framework here is the "Model Cards Toolkit"[6].

Many phases in ML development are iterative. Sometimes, we might need to review the business goals, KPIs, and available data from the previous steps to adjust the outcomes of the ML model results.

Finally, we package the ML workflow in a pipeline to create repeatable model training during the modeling phase.

5.4 Evaluating Machine Learning Models

Consequently, model training is followed by a model evaluation phase, also known as offline testing. During this phase, the performance of the trained model needs to be validated on a test set. Additionally, the model robustness should be assessed using noisy or wrong input data. Furthermore, it is best practice to develop an explainable ML model to provide trust, meet regulatory requirements, and govern humans in ML-assisted decisions.

[6] https://arxiv.org/pdf/1810.03993.pdf

Finally, the model deployment decision should be met automatically based on success criteria or manually by domain and ML experts. Similar to the modeling phase, all outcomes of the evaluation phase need to be documented.

5.5 Deployment

The ML model deployment denotes a process of the ML model integration into the existing software system. After succeeding in the evaluation step in the ML development life cycle, the ML model is graduated to be deployed in the (pre-) production environment. Deployment approaches are specified during the first phase of the ML development life cycle. These approaches will differ depending on the use case and the training and prediction modus, either batch or online. For example, deploying an ML model means exposing its predictive functionality as interactive dashboards, precomputed predictions, wrapping the ML model as a plug-in component in a microkernel software architecture[7], or web service endpoint in a distributed system.

The ML model deployment includes the following tasks: *inference hardware definition*, *model evaluation* in a production environment (online testing, e.g., A/B tests), providing *user acceptance and usability testing*, providing a *fall-back plan for model outages*, and setting up the *deployment strategy* to roll out the new model gradually (e.g. canary or green/blue deployment).

5.6 Monitoring and Maintenance

Once the ML model has been put into production, it is essential to monitor its performance and maintain it. When an ML model performs on real-world data, the main risk is the "model staleness" effect when the performance of the ML model drops as it starts operating on unseen data. Furthermore, model performance is affected by hardware performance and the existing software stack. Therefore, the best practice to prevent the model performance drop is to perform the *monitoring task* when the model performance is continuously evaluated to decide whether the model needs to be re-trained. This is known as the Continued

[7] https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch03.html

Model Evaluation[8] pattern. The decision from the monitoring task leads to the second task - updating the ML model. Additionally to monitoring and re-training, reflecting on the business use case and the ML task might be valuable for adjusting the ML process.

5.7 Conclusion

Getting ML models in production involves many interplaying components and processes. In this article, we reviewed the CRISP-ML(Q) development lifecycle model. Overall, CRISP-ML(Q) is a systematic process model for machine learning software development that creates an awareness of possible risks and emphasizes quality assurance to diminish these risks to ensure the ML project's success. The following list summarizes the CRISP-ML(Q) core phases and the corresponding tasks:

CRISP-ML(Q) Phases and Tasks

1. **Business and Data Understanding**
    - Define business objectives
    - Translate business objectives into ML objectives
    - Collect and verify data
    - Assess the project feasibility
    - Create POC
2. **Data Engineering**
    - Feature selection
    - Data selection
    - Class balancing
    - Cleaning data (noise reduction, data imputation)
    - Feature engineering (data construction)
    - Data augmentation
    - Data standardization

[8] https://www.oreilly.com/library/view/machine-learning-design/9781098115777/

3. **ML Model Engineering**

   - Define quality measure of the model
   - ML algorithm selection (baseline selection)
   - Adding domain knowledge to specialize the model
   - Model training
   - Optional: applying transfer learning (using pre-trained models)
   - Model compression
   - Ensemble learning
   - Documenting the ML model and experiments

4. **ML Model Evaluation**

   - Validate model's performance
   - Determine robustness
   - Increase model's explainability
   - Make a decision whether to deploy the model
   - Document the evaluation phase

5. **Model Deployment**

   - Evaluate model under production conditions
   - Assure user acceptance and usability
   - Model governance
   - Deploy according to the selected strategy (A/B testing, multi-armed bandits)

6. **Model Monitoring and Maintenance**

   - Monitor the efficiency and efficacy of the model prediction serving
   - Compare to the previously specified success criteria (thresholds)
   - Retrain model if required
   - Collect new data
   - Perform labeling of the new data points
   - Repeat tasks from the Model Engineering and Model Evaluation phases
   - Continuous integration, training, and deployment of the model

5.8 Acknowledgements

We would like to thank Alexey Grigoriev[9] for insightful discussions and his valuable feedback for this chapter.

Machine Learning Operations (MLOps) defines language-, framework-, platform-, and infrastructure-agnostic practices to design, develop, and maintain machine learning applications. However, getting ML in production implies many interplaying components. Moreover, with the currently exploding number of MLOps platforms and frameworks, it is challenging to keep up with the development pace. One of the main issues for ML-adopters is technology integration and compatibility.

According to the current surveys[10], 49% of organizations experience challenges with integrating their ML tooling, frameworks, and languages technology stack. The reason for this challenge is that ML technology is still evolving and is in its early stages. Additionally, the development of MLOps tooling is happening at a fast pace, making the adoption of such a fast-developing infrastructure difficult for getting ML-system into production sustainably.

To specify an architecture and infrastructure stack for Machine Learning Operations, we suggest a general MLOps Stack Canvas framework designed to be application- and industry-neutral. We align to the CRISP-ML(Q) model[11] and describe the eleven components of the MLOps stack and line them up along with the ML Lifecycle and the "AI Readiness"[12] level to select the right amount of MLOps processes and technlogy components.

[9] https://www.linkedin.com/in/agrigorev/
[10] https://info.algorithmia.com/tt-state-of-ml-2021
[11] https://ml-ops.org/content/crisp-ml
[12] https://services.google.com/fh/files/misc/ai_adoption_framework_whitepaper.pdf

6 MLOps Stack Canvas

The CRISP-ML(Q) provides a process - a set of steps that we will perform during the machine learning development life cycle. Another aspect of ML development is infrastructure components, such as tools, platforms, and frameworks needed for successful ML model operations (see Figure 1).

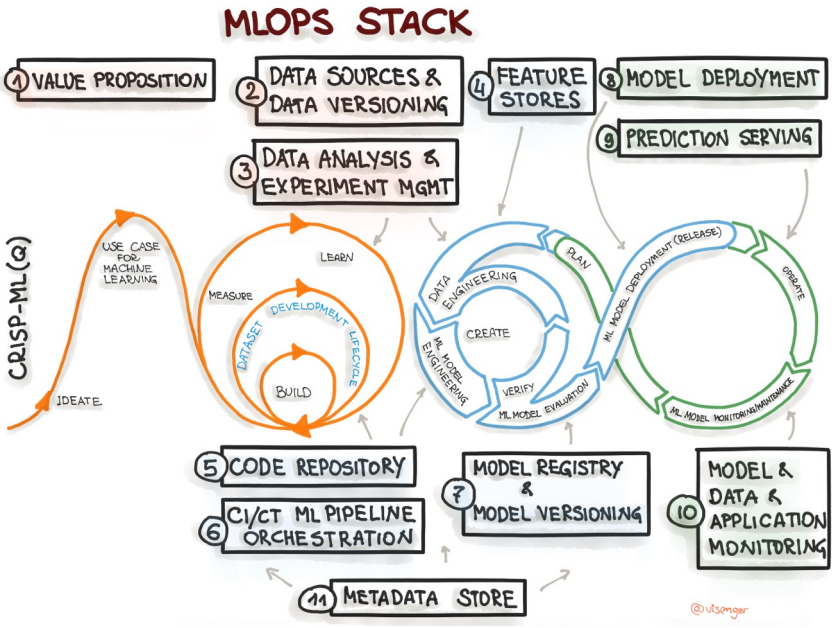

Figure 6.1: Mapping the CRISP-ML(Q) process model to the MLOps stack (source: ml-ops.org)

Hence, building the infrastructure stack for MLOps is the next important part of the ML project. However, given a lack of standardization in the ML development process model and operations and the ever-growing number of tools, creating an infrastructure stack is an overwhelming exercise for many ML aspiring teams. Given the hype around MLOps tools, small teams with a small amount of data are

getting the impression that MLOps solutions would require significant infrastructure investment.

According to the recent survey conducted by Algorithmia[1], *"the second greatest ML challenge is technology integration and compatibility,"* which indicates that many organizations are still at the beginning of the ML life cycle. To support the understanding of your requirements for the ML system and the audit process of the infrastructure components, we suggest an MLOps Stack Canvas framework (see Figure 2).

We envision the MLOps Stack Canvas as help to architect the ML system. Similar to the canonical Business Model Canvas, our canvas condenses the main elements of a whole technology stack for an ML application into a single page. This framework guides the development teams through the MLOps building blocks and lets them answer the MLOps infrastructure-related questions and identify the necessary tools chain.

The purpose of the MLOps Stack Canvas is to help you to structure workflows, architecture, and infrastructure components for the MLOps stack in the ML project. The scope of MLOps Stack Canvas is the following: + Ensuring that ML model solutions have a (business) impact. + Planning the cost of the infrastructure components for the MLOps stack by considering three main areas: + Data and code management + ML model management, and + Metadata management + Planning the cost of the orchestration of the ML system to manage its life-cycle and maintainability by considering + Continuous integration/training/delivery for ML assets + Monitoring to ensure the ML is achieving the business objectives + Alerting to deal with model failures. + Designing the ML system to fulfill: + Reproducibility: versioning, feature store, and pipelines, + Reliability: models should have few outages and safe failovers, and + Efficiency: model predictions are fast and as cost-effective as possible.

Generally, the MLOps Stack Canvas[2] consists of three main areas: *Data and Code Management*, *Model Management*, and *Metadata Management*. Each of these areas contains its own building blocks.

[1] https://info.algorithmia.com/tt-state-of-ml-2021
[2] https://miro.com/app/board/o9J_lfoc4Hg=/

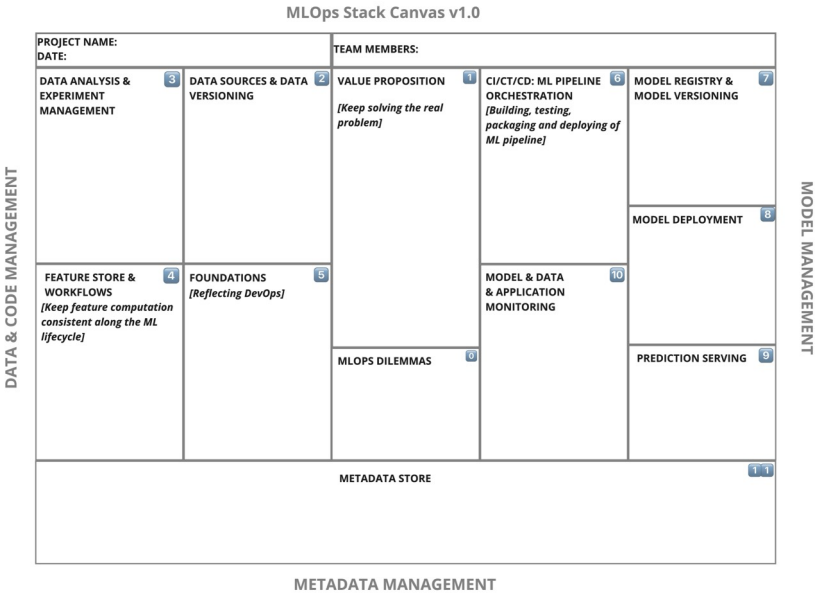

Figure 6.2: MLOps Stack Canvas to identify the infrastructure components (source: ml-ops.org)

Figure 2: MLOps Stack Canvas to identify the infrastructure components.

Please note that you might access the MLOps Stack Canvas[3] as a Miro board and it is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

In the following, we explain each of these blocks.

6.1 Blocks of the MLOps Stack Canvas

6.1.1 Value Proposition

The central component in the MLOps Stack Canvas is the Value Proposition. Generally, a value proposition is a statement that outlines why a customer would

[3] https://miro.com/miroverse/mlops-stack-canvas/

benefit from using our software or service. We create awareness about pain points, and the problem is being solved. An ML system generates predictions that are a foundation for a later decision to increase productivity or improve user experience. Therefore, the ML system should create value for an end-user. The value proposition section summarizes the key things that make up your software product and why end-users should use it. The goal is to focus on solving the real problem. To create an effective Value Proposition statement, we could use Geoffrey Moore's value positioning statement template:

"For (target customer) who (need or opportunity), our (product/service name) is (product category) that (benefit)."

We recommend applying the Machine Learning Canvas[4] through the initial phase of the CRISP-ML(Q) process to achieve both the high-level prototype of the ML system and to specify the value proposition. This value proposition could then be re-used in the MLOps Stack Canvas. Another practical tool to specify the value proposition is Value Proposition Canvas[5]. Generally, to formulate the value proposition for the ML project, we should answer the following questions:

1. What are we trying to do for the end-user(s)?
2. What is the problem?
3. Why is this an important problem?
4. Who is our persona? (ML Engineer, Data Scientist, Operation/Business user)
5. Who owns the models in production?

6.1.2 Data Sources and Data Versioning

Data is a fundamental part of machine learning. Hence, the next component, after articulating the business problem in Value Proposition, is "Data Sources and Data Versioning." The overall goal is to estimate the cost of data acquisition, storage, and processing, which are the main activities during the Business and Data Understanding and Data Preparation phases of the CRISP-ML(Q). Dataset development and its further processing for ML algorithms, such as assigning

[4] https://www.louisdorard.com/machine-learning-canvas
[5] https://www.peterjthomson.com/2013/11/value-proposition-canvas/

labels to unlabeled data points, might be costly. Since we consider ML models and data as "first-class citizens," data versioning might need to be implemented to analyze the model performance every time new data is available.

In addition, data versioning might be a regulatory requirement to explain predictions for every version of the learned model. Generally, we distinguish three levels of data versioning as described in "Machine Learning Engineering" by A.Burkov[6]: 1) Data is versioned as a snapshot at training time; 2) Versioning data and code as one asset; 3) Using specialized data versioning systems.

The following considerations in the "Data Sources and Data Versioning" component should be made:

1. Is this data versioning optional or mandatory? E.g., is data versioning a requirement for a system like a regulatory requirement?
2. What data sources are available? (e.g., owned, public, earned, paid data)
3. What is the storage for the above data? (e.g., data lake, DWH)
4. Is manual labeling required? Do we have human resources for it?
5. How to version data for each trained model?
6. What tooling is available for data pipelines/workflows?

Additionally to the current MLOps Stack Canvas, we recommend using the Data Landscape Canvas[7] to create an overview of the available, accessible, and required data sources for the ML project.

6.1.3 Data Analysis and Experiment Management

As described in CRISP-ML(Q), the initial phase of the project concerns tasks to translate business objectives to ML tasks, to collect and understand data, and to assess the feasibility of the project. To achieve these tasks, we run experiments and implement a proof of concept. The "Data Analysis and Experiment Management" block's focus is the applicability of the ML technology for the specified business goals and data preparation. Here, we should answer the following questions regarding tooling:

[6] http://www.mlebook.com/

[7] https://www.canvasgeneration.com/canvas/data-landscape/

1. What programming language to use for analysis? (R, Python, Scala, Julia. Or is SQL sufficient for analysis?)
2. Are there any infrastructure requirements for model training?
3. What ML-specific and business evaluation metrics need to be computed?
4. Reproducibility: What metadata about ML experiments is collected? (data sets, hyperparameters)
5. What ML Framework know-how is there?

6.1.4 Feature Store and Workflows

Being a crucial activity in machine learning, feature engineering is a process of transforming raw input data into feature vectors that are suitable input formats for machine learning algorithms before model training and prediction. The motivation for the "Feature Store,"[8] as a new component in the MLOps stack, is the need for management, reproducibility, discovery, and reuse of features across ML projects and various data science teams. A feature store is defined as an interface between data engineering and ML model engineering to separate the feature engineering from the CRISP-ML(Q) model development process. Feature stores promise the speed up in the development and operationalization of ML models. However, as an advanced component, feature stores might add complexity and its implementation need to be critically considered for every ML project: 1. Is this optional or mandatory? Do we have a data governance process such that feature engineering has to be reproducible? 1. How are features computed (workflows) during the training and prediction phases? 1. What are infrastructure requirements for feature engineering? 1. "Buy or make" for feature stores? 1. What databases are involved in feature storage? 1. Do we design APIs for feature engineering?

6.1.5 Foundations (Reflecting DevOps)

Related to almost every phase of the ML lifecycle, the next block in the MLOps Stack Canvas is about the inventory of the available DevOps infrastructure and raising awareness about the current DevOps principles in the ML project. The

[8] https://eugeneyan.com/writing/feature-stores/

intuition behind this activity is that we might extrapolate DevOps best practices to the MLOps activities. However, suppose there are gaps in the traditional DevOps practices. In that case, we should improve them first before starting with more complex activities such as model and data versioning, continuous model training, or feature store. We, therefore, follow the guidelines of the Accelerate State of DevOps Report[9] and execute a self-assessment for the software delivery performance by answering the following questions:

1. How do we maintain the code? What source version control system is used?
2. How do we monitor the system performance?
3. Do we need versioning for notebooks?
4. Is there a trunk-based development in place?
5. Deployment and testing automation: What is the CI/CD pipeline for the codebase? What tools are used for it?
6. Do we track deployment frequency, lead time for changes, mean time to restore, and change failure rate metrics[10]?

Following the DevOps principles has a direct impact on the software delivery performance. As MLOps build upon DevOps, establishing the stable DevOps culture for software projects is a prerequisite for successful ML projects.

6.1.6 Continuous Integration, Training, and Deployment: ML Pipeline Orchestration

In the previous block, we reviewed the existing CI/CD pipelines for the software delivery. Since the core software and ML model might have different release cycles, we examine the continuous integration routine for the ML model release in this block. Additionally, we introduce a new practice such as Continuous Training (CT).

Generally, continuous integration denotes the building, testing, and packaging of data and model pipelines. Continuous training is a new property, unique to

[9] https://cloud.google.com/blog/products/devops-sre/the-2019-accelerate-state-of-devops-elite-performance-productivity-and-scaling

[10] https://ml-ops.org/content/mlops-principles#ml-based-software-delivery-metrics-4-metrics-from-accelerate

ML systems concerned with automatically retraining ML models. We utilize the pipeline pattern[11], a software design pattern that implements the construction and execution of a sequence of operations. Typically, data and ML pipelines include operations such as data verification, feature and data selection, data cleaning, feature engineering, and model training. The set of acyclic pipelines construct a directed acyclic graph (DAG) and denote the overall workflow job. Depending on the maturity level, we might want to automate the data and model training pipeline workflows to operationalize the model. We trigger data preparation and model training pipelines whenever the new data is available, or the source code for the pipeline has changed. In this block of the MLOps Stack Canvas, we should clarify the processes and the toolchain for CI/CT in the CRISP-ML(Q) Model Engineering phase:

1. How often are models expected to be retrained? What is the trigger for it (scheduled, event-based, or ad hoc)?
2. Where does this happen (locally or on a cloud)?
3. What is the formalized workflow for an ML pipeline? (e.g., Data prep -> model training -> model eval & validation) What tech stack is used?
4. Is distributed model training required? Do we have an infrastructure for the distributed training?
5. What is the workflow for the CI pipeline? What tools are used?
6. What are the non-functional requirements for the ML model[12] (efficiency, fairness, robustness, interpretability, etc.)? How are they tested? Are these tests integrated into the CI/CT workflow?

6.1.7 Model Registry and Model Versioning

The next block of MLOps Stack Canvas concerns the ML model registry and versioning component, which is an essential infrastructural part for the Model Evaluation phase in the CRISP-ML(Q) process. The machine learning model is an essential asset along with the data and the software code. Similar to code versioning in traditional software engineering, establishing a model and data

[11] https://learning.oreilly.com/library/view/machine-learning-design/9781098115777/ch08.html
[12] https://arxiv.org/pdf/1906.10742.pdf

versioning practice is the foundation for reproducibility in machine learning. Depending on your use case, a change in code or data might trigger a model re-training.

The common reason for the machine learning model update is the "model decay," where the model performance declines with time as new data arrives. All ML models should be versioned and protocolled in regulated industries such as health, finance, or military. At the same time, there might be a need to ensure backward compatibility by rolling back previously built models. Also, by tracking several versions of the ML model, it is possible to implement different deployment strategies such as "canary"- or "shadow"-deployment by analyzing the latest trained model's performance improvement. Hence, in this category, we answer the following questions:

1. Is this optional or mandatory? The model registry might be mandatory if you have multiple models in production and need to track them all. The reproducibility requirement might be the reason that you need the model versioning.
2. Where should new ML models be stored and tracked?
3. What versioning standards are used? (e.g., semantic versioning)

Please note that as an advanced component, the ML model registry and versioning might be reasonable in the later stages of the ML projects.

6.1.8 Model Deployment

After training and evaluation, we transit to the next phase of the CRISP-ML(Q) and deploy the ML model. Deploying an ML model denotes making it available on the target environment for receiving prediction requests. Continuous Deployment (CD) is an automatic deployment of ML models into the target environment based on predetermined evaluation metrics. In this section of the MLOps Stack Canvas, we specify all model exposure strategies and infrastructure aspects of CD by answering the following questions:

1. What is the delivery format for the model?
2. What is the expected time for changes? (Time from commit to production)

3. What is the target environment to serve predictions?
4. What is your model release policy? Is A/B testing or multi-armed bandits testing required? (e.g., for measuring the effectiveness of the new model on business metrics and deciding what model should be promoted into the production environment)
5. What is your deployment strategy? (e.g. shadow/canary deployment required?)

6.1.9 Prediction Serving

This block of the MLOps Stack Canvas deals with the ML model serving as a process of applying a machine learning model to the new input data. Generally, we distinguish between online and batch modes of prediction serving. They can be implemented using five patterns[13] to put the ML model in production: *Model-as-Service, Model-as-Dependency, Precompute, Model-on-Demand,* and *Hybrid-Serving*. Each of these patterns would require different infrastructure settings. For example, Model-as-Service implements a model serving as a distributed service for input requests via REST API and implies the on-demand modus for prediction responses. At the same time, the Precompute pattern means that the prediction modus is batch and the model predictions are precomputed and stored in a relational database.

To identify the environment for model serving, we should answer the following questions in the Prediction Serving block of the MLOps Stack Canvas:

1. What is the serving mode? (batch or online)
2. Is distributed model serving required?
3. Is multi-model prediction serving[14] required?
4. Is pre-assertion for input data implemented?
5. What fallback method for an inadequate model output (post-assertion) is implemented? (Do we have a heuristic benchmark?)
6. Do you need ML inference accelerators (TPUs)?
7. What is the expected target volume of predictions per month or hours?

[13] https://ml-ops.org/content/three-levels-of-ml-software#code-deployment-pipelines
[14] https://www.oreilly.com/content/efficient-machine-learning-inference/

6.1.10 ML Model, Data, and System Monitoring

Once the ML Model is deployed, it must be constantly monitored to ensure the model quality, meaning that the model serving produces correct results. This block of the MLOps Stack Canvas clarifies the monitoring part of running the ML System in production and relates to the sixth phase of the CRISP-ML(Q) process:

1. Is this optional or mandatory? For instance, do you need to assess the effectiveness of your model during prediction serving? Do you need to monitor your model for performance degradation and trigger an alert if your model starts performing badly? Is the model retraining based on events such as data or concept drift?
2. What ML metrics are collected?
3. What domain-specific metrics are collected?
4. How is the model performance decay detected? (Data Monitoring)
5. How is the data skew detected? (Data Monitoring)
6. What operational aspects need to be monitored? (e.g., model prediction latency, CPU/RAM usage)
7. What is the alerting strategy? (thresholds)
8. What triggers the model re-training? (ad hoc, event-based, or scheduled)

6.1.11 Metadata Store

Finally, we should specify an overlapping area such as metadata management. Metadata management is the management of information about each execution of the experiments, data and model pipeline is recorded to provide data and artifacts lineage, reproducibility, and debug errors.

The Metadata Store is a cross-cutting component that spans all previous elements of the MLOps infrastructure stack. Depending on your organization and regulatory requirements, you might need to implement an ML model governance process. This process will mainly rely on ML metadata. Therefore, the requirement for ML governance is the ML metadata store component. In the last block of the MLOps Stack canvas, we answer the following questions:

1. What kind of metadata in code, data, and model management need to be collected? (e.g., the pipeline run ID, trigger, performed steps, start/end timestamps, train/test dataset split, hyperparameters, model object, various statistics/profiling, etc.)
2. Are any ML governance processes included in the MLOps lifecycle? What metadata will be required?
3. What is the documentation strategy: Do we treat documentation as a code? (examples: Datasheets for Datasets[15] and Model Card for Model Reporting[16])
4. What operational metrics need to be collected? E.g., time to restore, change fail percentage.

6.1.12 3 Dilemmas of MLOps

There are also organizational aspects of MLOps, which belong to the general discussion about building the ML projects' infrastructure. 1. **Tooling**: Should we buy, use existing open-source or build in-house tools for any of the MLOps components? What are the risks, trade-offs, and impacts of each of the decisions? 1. **Platforms**: Should we agree on one MLOps platform or create a hybrid solution? What are the risks, trade-offs, and impacts of each of the decisions? 1. **Skills**: How expensive is it to either acquire or educate our own machine learning engineering talents?

The MLOps Stack Canvas scope is to assist you while identifying the workflows, architecture, and infrastructure components for the MLOps stack in the ML project. Answering questions in this canvas should get you a good estimation of costs that accompany your ML project in every phase.

6.2 Documenting MLOps Architecture

One of the effective ways to document the MLOps architecture is by using Architecture Decision Records[17] (ARD) construct. For example, one ARD can manifest

[15] https://arxiv.org/abs/1803.09010
[16] https://arxiv.org/abs/1810.03993
[17] https://adr.github.io/

each building block from the MLOps Stack Canvas. The simplified ARD format consists of three essential components:

ARD: A brief description of the architectural decision.

Context: A short description of the problem.

Decision: Here, we describe the actual architectural decision with a detailed explanation.

Consequences: This section provides any implications of the architecture decision. This section is also a good place to discuss any architectural trade-offs.

The following is a simplified example of such ARD.

ARD: Dataset versioning.

Context: As requested by the regulatory requirements, every retraining of the ML model should be in sync with the changes in the dataset.

Decision: The ML model should be retrained whenever a new batch of data points is collected. Therefore, we decided to use DVC to track datasets and ML models. An alternative solution would be LakeFS.

Consequences: We need to move our data storage to the DVC-supported storage mechanisms. Additionally, upskilling our team members is required.

6.3 MLOps Maturity Level

Machine learning is still a new technology for many organizations. Launching ML-powered software solutions goes through three stages of "AI Readiness"[18], which denotes a company's maturity in incorporating AI into the business. The AI maturity can be distinguished between three phases: *Tactical, Strategic,* and *Transformational* (see Figure 3).

The initial phase, known as **Tactical**, denotes that the organizations explore the capabilities of ML/AI technologies. The reasonable way is to start with non-critical use cases and short-term projects. While building a proof of concepts, we create an

[18] https://services.google.com/fh/files/misc/ai_adoption_framework_whitepaper.pdf

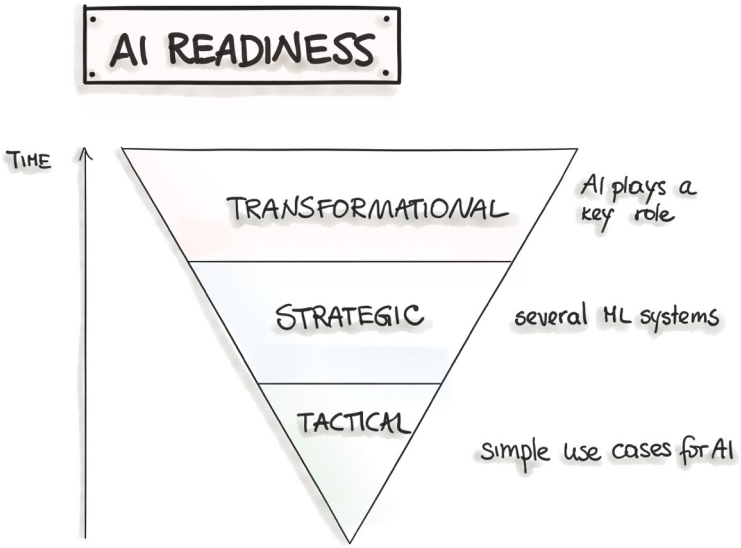

Figure 6.3: AI Readiness Phases (source: ml-ops.org)

opinion about the applicability of machine learning. This phase's focus is clearly on learning and understanding what value ML might generate. Furthermore, all processes are mainly manual because there is little to no MLOps infrastructure and ML skills are being developed.

The next phase, called **Strategic**, implies that business goals coordinate the amount of ML use cases and parts of the processes are automated. Typically, there are basic ML skills in the team and basic infrastructure to get ML models into production. The main distinction to the previous phase is utilizing pipelines for data preparation and model training. Additionally, the ML system provides end points, such as REST API, to expose ML models to the target application. Basic ML monitoring and alerting functionality are in place as well.

In the **Transformational** phase, organizations use ML to stimulate innovation. In this phase, ML is productionized as a fully automated process, which im-

plies a sophisticated data platform, widely adopted common patterns for ML development, and CI/CT/CD practices . Usually, the ML expertise is a part of a cross-functional team to maintain the productionized model. Furthermore, organizations utilize advanced MLOps components, such as feature store, ML model and data versioning, and model monitoring and alerting to trigger model re-training.

In general, **the amount of MLOps practices and infrastructure components will depend on the organization's ML-maturity described in the "AI Readiness" framework**. The suggested MLOps Stack Canvas might be aligned with the "AI Readiness" maturity level of an organization. It is reasonable to apply the MLOps Canvas starting from the second phase, "Strategic," where the AI use cases are aligned to the business core domain and teams start utilizing pipelines, experiment workflows and dedicated APIs for exposing the ML model.

6.4 Conclusion

Getting ML models in production involves many interplaying components. With the currently exploding number of MLOps platforms and frameworks, it is challenging to keep up with the industry's development pace and create the right technical environment for ML projects. To specify an architecture and infrastructure stack for Machine Learning Operations, we reviewed the CRISP-ML(Q) development lifecycle and suggested an application- and industry-neutral *MLOps Stack Canvas*. This Canvas helps identify the workflows, architecture, and infrastructure components for the MLOps stack in the ML project. Answering questions in the canvas should get a reasonable estimation of required components and its costs in the ML project.

6.5 Acknowledgements

We would like to thank Alexey Grigoriev[19] and Louis Dorard[20] for the insightful discussions and their valuable feedback for this chapter.

[19] https://www.linkedin.com/in/agrigorev/
[20] https://www.linkedin.com/in/louisdorard/

7 MLOps and Model Governance

MLOps and Model Governance are often perceived as separate processes. However, they are tightly interwoven. This chapter introduces the integration of these frameworks and explains essential principles and technical components of MLOps and ML model governance.

Recently, Machine Learning Operations (MLOPs)[1] has received a lot of attention as it promises to bring machine learning (ML) models into production quickly, effectively, and for the long term. MLOps is equivalent to DevOps in software engineering: it is an extension of DevOps for the design, development, and sustainable deployment of ML models in software systems. Model Governance encompasses a set of processes and frameworks that help in the deployment of ML. Setting up automatized and reproducible data and ML pipelines reduces the amount of time required to bring models into production (time-to-market). There are six interactive phases in the ML development process:

- Business and Data Understanding
- Data Engineering
- Model Engineering
- Quality Assurance for ML Systems
- Deployment
- Monitoring and Maintenance

This figure shows the most important phases of the ML life cycle according to CRISP-ML(Q)[2]:

However, the operationalization of ML models is not the only challenge many companies are facing today. The use of MLobliges companies responsibility and compliance with legal requirements. To fulfill these obligations, a company requires processes through which it is able to:

- control access to ML models
- put guidelines and legal requirements into practice

[1] https://www.innoq.com/de/articles/2020/10/mlops-operations-fuer-machine-learning/
[2] https://arxiv.org/pdf/2003.05155.pdf

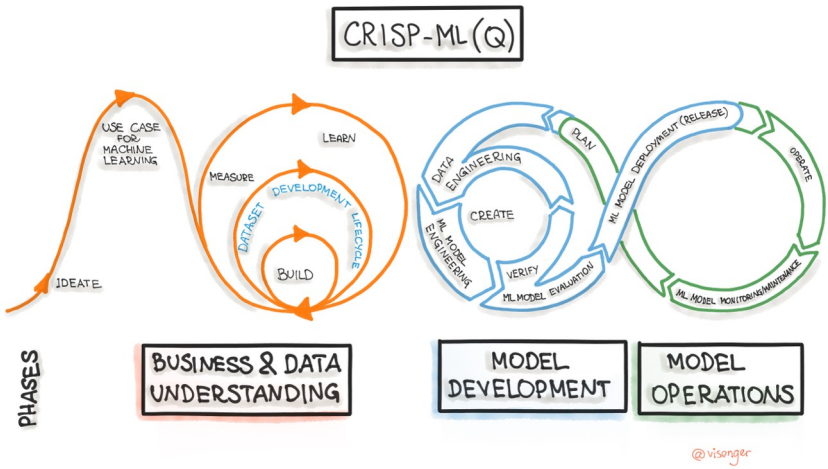

Figure 7.1: CRISP-ML(Q) process model (source: ml-ops.org)

- track interactions with the ML models and their results
- document the foundation of an ML model (stakeholders, business context, training data, feature selection, guidelines for model reproduction, choice of parameters, results of model evaluation and validation) Collectively, these processes are referred to as *Model Governance*[3].

7.1 Model Governance - A New Challenge

Organizations often don't recognize the importance of Model Governance until models are supposed to be deployed. Many companies have automatized ML pipelines but fail to bring models into compliance with legal requirements. According to an *Algorithmia*-Study[4] from 2021, 56 percent of respondents considered the implementation of model governance to be one of the biggest challenges for successfully bringing ML applications into production. In Germany, this *IDG*

[3] https://www.datarobot.com/blog/what-is-model-governance/
[4] https://www.artificialintelligence-news.com

Research Services ML 2021 study[5] found that 26.2 percent of companies believe that compliance risks pose the biggest challenge, while 35.8 percent consider legal considerations (such as transparency of algorithmic decision-making) to be a remaining difficulty.

7.2 Model Governance Will Not Be Optional

Companies already have to comply with several legal regulations[6]. In addition, these requirements are expected to be expanded by AI-specific regulations: the EU published a regulation draft in April 2021 and entered into force in August 2024[7] as the first legal framework for AI. The EU AI Act takes an approach that classifies different types of AI systems according to four distinct risk categories[8]. The risk category determines the scope of the regulations: the higher the determined risk, the more requirements have to be met:

7.2.1 Category 1 ("Unacceptable Risk")

AI software considered to be a significant risk for security, livelihoods, and human rights is forbidden (e.g., social scoring systems).

7.2.2 Category 2 ("High Risk")

AI software in the "high" risk category is subject to strict requirements. These include the following aspects[9]: Robustness, security, accuracy (precision), doc-

[5] https://www.lufthansa-industry-solutions.com/de-de/studien/idg-studie-machine-learning-2021

[6] https://www.lufthansa-industry-solutions.com/de-de/studien/idg-studie-machine-learning-2021?gclid=CjoKCQjw18WKBhCUARIsAFiW7JyBopj392wCTvucZ2vWlUqEFXl5pVuj1eVrgMs7YT3i5foYy-iMYuIaAgO5EALw_wcB#download

[7] https://artificialintelligenceact.eu/ai-act-explorer/

[8] https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/excellence-trust-artificial-intelligence_de

[9] https://germany.representation.ec.europa.eu/news/fur-vertrauenswurdige-kunstliche-intelligenz-eu-kommission-legt-weltweit-ersten-rechtsrahmen-vor-2021-04-21_de

If, we enable "Tag" as a separate tab in NNDM with its own entity model as below :

1. **tags**
   - `id SERIAL PRIMARY KEY`
   - `name VARCHAR(255) NOT NULL UNIQUE`
   - `description TEXT`
   - `group_id INTEGER REFERENCES groups(id)`
   - `path VARCHAR(255) NOT NULL` : The hierarchical path of the tag, e.g., `/ProcessCell1/ProcessUnit1/EquipmentModule1/T_101`.
2. **groups**
   - `id SERIAL PRIMARY KEY`
   - `name VARCHAR(255) NOT NULL UNIQUE`
   - `description TEXT`
3. **group_types**
   - `id SERIAL PRIMARY KEY`
   - `name VARCHAR(255) NOT NULL UNIQUE`
   - `mandatory_attributes JSONB` : A JSON array containing mandatory attribute names for this group type.
4. **groups_to_group_types**
   - `group_id INTEGER REFERENCES groups(id)`
   - `group_type_id INTEGER REFERENCES group_types(id)`
   - `PRIMARY KEY (group_id, group_type_id)`
5. **attributes** (for storing both mandatory and non-mandatory attributes)
   - `id SERIAL PRIMARY KEY`
   - `name VARCHAR(255) NOT NULL UNIQUE`
   - `data_type VARCHAR(50) NOT NULL` : The data type of the attribute, e.g., 'text', 'integer', 'boolean'.
6. **tags_to_attributes**
   - `tag_id INTEGER REFERENCES tags(id)`
   - `attribute_id INTEGER REFERENCES attributes(id)`
   - `value JSONB` : The value of the attribute for the given tag.
   - `PRIMARY KEY (tag_id, attribute_id)`
7. **group_attributes** (for storing mandatory attributes that are specific to each group)
   - `group_id INTEGER REFERENCES groups(id)`
   - `attribute_id INTEGER REFERENCES attributes(id)`
   - `value JSONB` : The value of the mandatory attribute for the given group.
   - `PRIMARY KEY (group_id, attribute_id)`

Here's an example of how these tables might relate to each other:

- A `tag` has a `name`, `description`, and belongs to a `group`. Each tag has a unique `path` that represents its hierarchical position.
- A `group` can have one or more associated `group_types`. Each group type defines mandatory attributes for the groups of that type.
- The `tags_to_attributes` table allows you to associate arbitrary attributes with tags, while the `group_attributes` table stores mandatory attribute values specific to each group.
- The `attributes` table stores the definitions and data types of attributes.

This solves all use cases for tags - without leveraging purview. And then we can have the logic app connect to NNDM (if this is exposed via API) and enable approvals from there

Figure 7.2: Different risk categories according to EU draft on AI regulation (source: ml-ops.org)

umentation and logging as well as appropriate risk assessment and mitigation. Further requirements[10] include high-quality training data, non-discrimination, traceability, transparency, human monitoring, and the need for conformity testing and proof of compliance through CE marking. Examples of ML systems in this category include private and public services (credit scoring) or systems used in education or vocational training to decide on a person's access to education and career path (e.g., exam scoring).

7.2.3 Category 3 ("Limited Risk")

This AI software is subject to a transparency obligation. For example, chatbot users must be informed that they are interacting with AI software.

7.2.4 Category 4 ("Minimal Risk")

AI software in this category is not subject to any regulation (e.g., spam filters).

As the regulation is supposed to apply not only to EU-based companies but also to any company offering AI services within the EU, the law would have a similar scope of application as the GDPR[11]. The regulation must be approved by the EU Parliament and pass through the legislative procedures of the individual member states[12]. If the law enters into force in 2024, high-risk systems will have to undergo conformity assessment[13] before deployment. After passing the conformity assessment, the AI system can be registered in an EU database and receive a declaration of conformity, which is required to obtain the necessary CE marking.

However, the EU draft still has room for improvement[14]. The definition of an AI system is probably the biggest challenge in this context. This definition is

[10] https://planit.legal/das-ki-gesetz-der-eu-entwurf-und-diskussionsstand/
[11] https://planit.legal/das-ki-gesetz-der-eu-entwurf-und-diskussionsstand/
[12] https://www.taylorwessing.com/de/insights-and-events/insights/2021/04/eine-neue-europaeische-regulierung-fuer-kuenstliche-intelligenz
[13] https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/excellence-trust-artificial-intelligence_en
[14] https://planit.legal/das-ki-gesetz-der-eu-entwurf-und-diskussionsstand/

blurry and might lead to different interpretations. Companies could be struggling to align their technical work with definitions in the regulation as the general character of the draft makes it unclear how the definitions, evaluation criteria and requirements will have to be put into practice. It is also important to note that EU regulations are not the only decisive argument for embracing model governance. Model governance is also relevant for low-risk ML not operating in a regulated domain, but which are related to high business risks. For example, if a company sells spam filters that regularly legitimate emails, its market position might be at risk due to this malfunction. As a result, model governance is not only needed to meet legal requirements, but also to ensure the quality of ML systems.

7.3 The Integration of Model Governance and MLOps

The integration of MLOps and model governance depends on two aspects:

Strength of the regulations – determined by the business domain, the risk category of an ML model, and the business risk

There are industries with a long tradition of strict regulations, such as the health or finance sectors. Upcoming EU regulations might add new requirements to these domains, but they could also apply to non-regulated domains if the system is considered to be of high risk. Finally, the influence of AI systems on business success is crucial: if commercial success is heavily dependent on AI systems, management requirements should be correspondingly strict.

Number of ML models that need to be integrated into the software systems

The number of ML models shows two aspects. First, it reveals how strongly a company integrates ML into its main business domain and/or how organizationally and technically mature the company is for implementing the planned ML projects. Thus, a low number of ML models can either mean that ML does not play an important role in a company's business concept or that companies are not yet sufficiently equipped to implement ML.

Fig. 3 shows the integration of model governance and MLOps along the degree of regulation and the number of models as Venn diagrams:

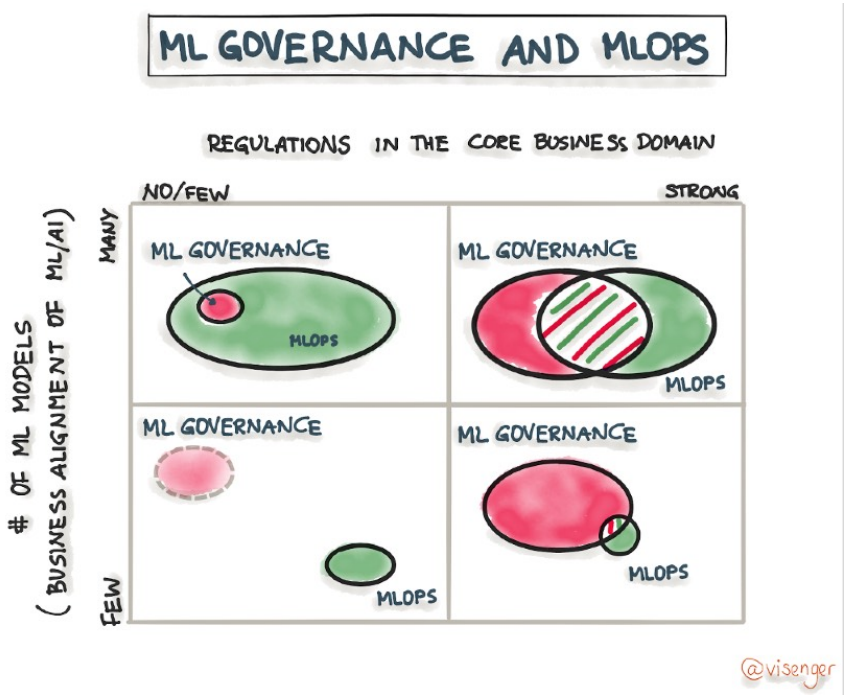

Figure 7.3: The integration of MLOps and Model Governance, displayed as Venn diagrams. The higher the degree of regulation, the closer the integration. The more models we have, the bigger the importance of MlOps becomes (source: ml-ops.org)

7.3.1 Variant 1: Many Models and Strict Regulation

This scenario, where we see many models operating in a highly regulated application domain, is the most complex case. Model governance and MLOps are equally important here and should be closely related to each other – model governance should be integrated into every step of the MLOps life cycle (development, deployment, and operations).

Examples: Which Domains Are Subject to Strict Regulation?

Models used in the healthcare and financial sectors are examples of models operating in strictly regulated domains. However, EU law might also apply to models of the high-risk category, even though they are not being applied in traditionally regulated fields. For example, high-risk systems could be ML systems that support process automation in critical infrastructure, but also models that automate human decision-making in order to determine access to vocational and school education.

Framework for Model Governance

It is important to integrate model governance processes into every step of the ML life cycle from the very beginning. The following Model Governance framework[15] guides through the whole life cycle and covers both legal and corporate requirements.

This list describes the main components of the framework for model governance which should be integrated into every stage of the ML life cycle:

ML Lifecycle, Model Governance Components und Tasks und Artifacts

7.3.2 1. Development (building the training pipeline)

- **Reproducibility, Validation**:
  - Model Metadata Management
  - Model Documentation
  - Model and Data Registry
  - Model Evaluation and Validation

[15] https://www.oreilly.com/library/view/the-framework-for/9781098100483/ch01.html

7.3.3 2. Deployment & Operations

- **Observation, Visibility, Control**:
  - Logging (Serving Logs)
  - Continuous Monitoring and Evaluation
  - ML Infrastructure Cost Transparency
  - Versioning of Models and Data Sets
  - Tracking ML-Metadata in ML Metadata and Artifact Registry
- **Monitoring and Alerting**:
  - Continuous Monitoring and Evaluation
  - Automated Alert Function (in response to performance loss or distribution shifts)
- **Model Service Catalog**:
  - Providing requested information on models for internal reusability
- **Security**:
  - Compliance with IT standards
  - Authentication, SSO, and RBAC
  - Management of Model Endpoints and API
  - Management of Keys and Secrets
  - System Testing
- **Conformity and Auditability**:
  - Collecting relevant information from Model Logging (ML metadata)
  - Documentation
  - Audit Results
  - Conformity Testing
  - Certificate of Conformity (CE Mark)

7.4 Reproducibility and Validation

In the first phase of the ML lifecycle, **reproducibility must be established and the model should be validated**.

Reproducibility is the ability to obtain the same result twice. Similar to scientists precisely specifying experimental procedures, ML reproducibility must provide **relevant metadata** and information to reproduce models. **Model metadata management** includes the type of algorithm, features and transformations, data snapshots, hyperparameters, performance metrics, verifiable code from source code management, and the training environment.

Documentation is a commonly required part in regulated domains. However, the project itself also benefits from good documentation within the company as transparency and traceability minimize the risks of *technical debt*[16]. Documentation includes the following aspects: an explanation of the business context, a high-level explanation of the algorithm, model parameters, selection and definition of features, instructions for reproducing the model, and examples for training the algorithms as well as examples for making predictions by the algorithm. Documentation can be supported by useful toolkits such as *Data Sheets*[17] and *Model Cards*[18]. Data Sheets record which kind of mechanisms or procedures were used for data collection or whether ethical review procedures took place. Model Cards complement Data Sheets and provide information about the development of a model, the assumptions that have been made, and expectations about model behavior across different cultural, demographic, or phenotypic groups.

The **validation** of ML models is a multi-stage process with different metrics, such as performance indicators like accuracy or F1 score, or business metrics like KPIs to check for statistically significant improvement compared to a control group in A/B testing. KPIs can also be used to test whether the ML problem itself is properly formulated. In addition, the development team should check whether the model can be reproduced. Another important component of validation is explainability: are developers able to explain how individual features affect a prediction? Often, explanations for models can only be approximated.

After model deployment, model governance processes should be integrated into the deployment and operations phases of the ML lifecycle:

[16] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/aad9f93b86b7addfea4c419b9100c6cdd26cacea.pdf

[17] https://arxiv.org/pdf/1803.09010.pdf

[18] https://arxiv.org/pdf/1810.03993.pdf

7.5 Observation, Security, Control

This component enables companies to provide transparency about the model which includes **logging, metrics and auditing**: model logging values are processed into metrics and visualized in dashboards for analysis and communication purposes. **Cost transparency** provides visibility into costs and facilitates billing of various teams for a specific model and resource usage. **Model Usage Reports** provide visibility into the success and adoption of individual models and can support access control. The last component includes *versioning of model and data*, to ensure that all models can be restored without data loss or modification and that model prediction can be traced back to the model version that has produced them.

7.6 Monitoring and Alerting

Key metrics should be continuously monitored to detect deviations early. If deviations, such as distribution shifts or performance loss are detected, alerts should be sent to inform stakeholders immediately (if MLOps-pipelines are fully automated, deviations will automatically trigger the CI/CD training pipeline). The monitoring engine requires appropriate platform and infrastructure integration with dashboard and monitoring tools, continuous monitoring of uptime SLA as a metric for application stability and availability, and alerting functionalities when problems occur.

7.7 Model Service Catalog

The Model Service Catalog is an internal marketplace of all ML models in the enterprise. The catalog should have a good UX, it should be connected to the location of model storage, and it should display relevant metadata for a model such as its latest version, inputs, and outputs. Employees with appropriate permissions can access the catalog, search for models, and retrieve requested information about the models.

7.8 Security

ML security[19] is an important aspect of the model governance agenda. This study[20] by Gartner estimates that by 2022 thirty percent of cyber security attacks will have an ML-specific character. To be protected against these attacks, measures must be taken to meet required security standards. For example, making models accessible through HTTP comes with the concurrent risk of misuse. Therefore, adherence to IT standards (DNS, proxies, and load balancing for data traffic) is very important. However, the complexity of these standards may require the services of third-party providers.

ML security management needs to secure and manage endpoints to make sure that only authorized users can create, change, or delete endpoints. Authentication, SSO, and role-based access control (RBAC) are further important aspects of ML security. ML models must be integrated into token-based authentication solutions to make sure that only eligible users are able to query the model. In addition, key and secret management should provide solutions for creating, saving, and managing keys. Finally, models are supposed to pass security audits. Therefore, it is very recommendable to involve IT and company experts from the beginning of a project to fully consider all security requirements.

Protection against ML-specific cybersecurity attacks is also very important. With models often being trained on sensitive data, **data and information security** play an important role. Using this Adversarial ML Threat Matrix[21] might be helpful: it is comparable to the classic Attack Chain Modeling and builds on the well-established MITRE Att&CK[22], a globally accessible knowledge base of attacks and techniques. MITRE Att&CK is used as a guideline for the development of specific threat models and methods in the private sector, in governments, and in the field of cyber security products and services. The Adversarial ML Threat Matrix is a similar concept for ML security – it contains a collection of known weaknesses and the associated attacks.

[19] https://www.innoq.com/de/articles/2021/08/machine-learning-security-teil-2/
[20] https://www.gartner.com/en/documents/3939991
[21] https://github.com/mitre/advMLthreatmatrix
[22] https://attack.mitre.org/

7.9 Conformity and Auditability

Models of the high-risk category are supposed to undergo conformity testing in order to be eligible to receive the CE marking[23], which is a prerequisite to placing models on the European market. In order to fulfill the compliance and auditability requirements of a heavily regulated domain, the model governance framework should be as automated, transparent, and complete as possible. **Model logging, metrics, and audits** are very important in order to prove compliance with requirements. This includes collected and visually prepared model information, for example displaying metrics in dashboards, model and data versioning, and audit results (tested components of the validation in the development phase). Compliance with security requirements is another prerequisite for passing conformity testing: permissions, authorized access to ML applications and authentications have to be put into place. Each domain is subject to different regulations, and there is no one-size-fits-it-all solution. Compliance is very complex and often requires years of expertise. This makes it even more important to involve compliance and security experts in the model governance strategy from the very beginning to make sure that important considerations are not being overlooked.

7.9.1 Variant 2: Many Models and Little Regulation

This variant applies if the application domain is not highly regulated, the ML model is not in a high-risk category, and the associated business risk is low. In this case, companies need MLOps and model governance to manage and operationalize models. With legal requirements being weaker and the focus on MLOps being stronger than in the first variant, model governance forms part of MLOps and is not a stand-alone framework, unlike in the variant presented previously. To understand how model governance can be integrated into MLOps, an overview of MLOps is helpful. This *Google* paper *"Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning*[24] divides MLOps into six integrated and iterative processes: ML Development, Training Operationalization, Continuous Training, Model Deployment, Prediction Serving, Continuous Monitoring, and Data and Model Management. We rely on this MLOps framework

[23] https://planit.legal/das-ki-gesetz-der-eu-entwurf-und-diskussionsstand/

[24] https://services.google.com/fh/files/misc/practitioners_guide_to_mlops_whitepaper.pdf

to demonstrate the integration of model governance and MLOps for the second variant (many models and low regulation).

Ideally, the result of the ML development should not be the model, but the formalization of the training pipeline (which encompasses the data- and model-engineering pipeline). When the monitoring machine detects dropping performance or distribution shifts, a trigger kicks off the (continuous) training pipeline: the model is re-trained with new data and then re-deployed. The training pipeline consists of a data engineering component (data entry, data validation, data transformation) and a model component (model training, model evaluation, model validation) and should be versioned and tested as well. As a first step in the training, new training data are loaded from the data repository. Data and feature repositories standardize the definition and saving of the data, ensure data consistency for training and inference, and support data preprocessing and feature engineering processes. The data pass through all steps of the data engineering process and are then used to retrain the model. After model training, the model is evaluated, validated, and saved as a new model candidate in the model registry. All metadata and artifacts that are being produced during the training run are saved in the ML metadata and artifact repository. ML artifacts include statistics and data schemas, trained models, and metrics. ML metadata are the information about these artifacts (pipeline run ID, trigger, process type, start and end time, status, environmental configurations, and input parameter values).

Model registries manage the life cycle of ML models. After being registered, model governance processes check whether the model candidate is ready to be deployed into the productive system (model deployment). After successful deployment, the model provides predictions for every input (prediction serving). Model governance permanently monitors the performance of the productive system (continuous monitoring) and collates all relevant metrics in an independent report (e.g. accuracy). As a result of the monitoring, performance drops or changing input data are immediately diagnosed, an alert is being sent, and/or the CI/CD training pipeline is triggered to produce a new model candidate.

Figure 4 gives an overview shows how data and model management are implemented as cross-cutting processes in the MLOps life cycle:

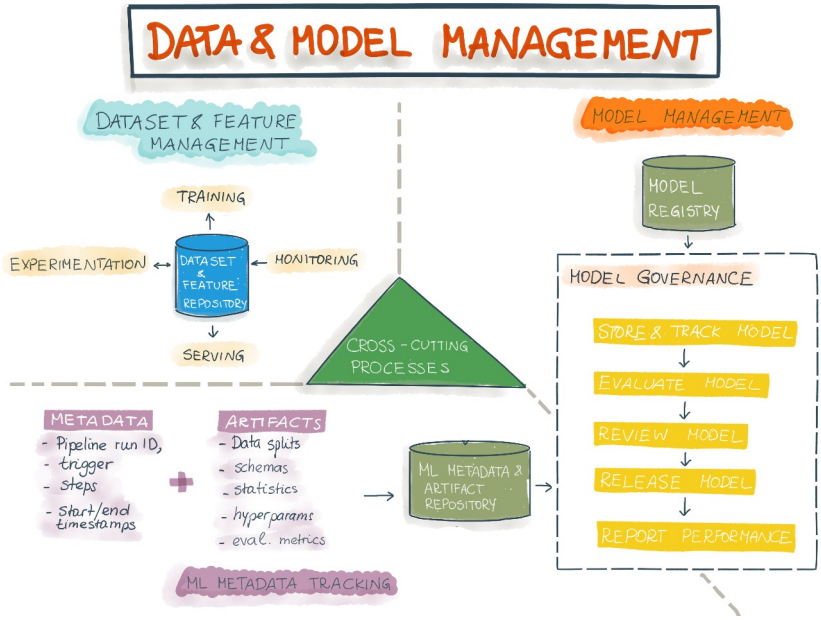

Figure 7.4: Data and model management are cross-purpose processes in the MLOps life cycle (source: ml-ops.org)

7.10 Model Governance as Part of Model Management

Model governance encompasses the **recording, auditing, validation, approval, and monitoring of models**. In this variant, model governance is the final supervisory authority to approve a model for being deployed into the production environment. The list below summarizes the model governance components with the necessary tasks and artifacts. For the implementation of these tasks, model governance uses information from the ML metadata, the artifact repository, and the model registry.

- Storage/Versioning of models
- Evaluation and Explainability
- Testing

- Release
- Report (summary, visualization, highlighting of metrics) for quality assurance of the productive model

The model registry saves all model versions to ensure reproducibility and accountability (similar to the model versioning in the "observation and control of model governance" component).

The evaluation of a model is very important as well. To evaluate a model candidate, it can be released using shadow deployment. Then, its performance can be compared to the current productive model by comparing performance and business metrics (similar to the validation of the development phase).

Model audits are another fundamental aspect. Any model changes must be checked and approved to control risks in different categories (for example business, financial, legal, security, data protection, reputational, and ethical risks) as described in this end-to-end framework for the internal auditing of algorithms. The auditing component in this model governance framework can be considered to be a less strict variant of the certificate of conformity. Finally, a report contains the summary, visualization, and highlighting of model performance metrics collected during the monitoring process.

7.10.1 Variant 3: Few Models and Little Regulation

This variant applies to companies that use ML models in non-regulated industries which do not belong to the high-risk category according to the EU draft and where the business risk is quite low. The low number of ML models can either mean that ML does not play an important role in the business strategy of the company or that the company is still in the early stage of AI maturity.

With a limited scope of regulations and a low number of models, companies do not have to consider too many aspects. The lack of requirements and the low number of models allows model governance to be optional, although it is still recommended to use it for quality assurance. The components of the development phase are also relevant for this scenario.

7.10.2 Variant 4: Few Models and Strict Regulation

This scenario applies to companies operating in strictly regulated industries that are only using a small number of models. The only difference to variant 1 is the low number of models, which makes MLOps less important for this use case. However, the close integration of model governance frameworks with MLOps remains the same: in heavily regulated domains, model governance must cover the complete MLOps life cycle, even if only a few models are in use (see variant 1).

7.11 Summary – The Main Components of Model Governance

Although the strength of regulation and the number of models determine how model governance should be implemented, all variants have the following aspects in common:

- Comprehensive model documentation or reports. This includes the report of metrics by using appropriate visualization techniques and dashboards
- Versioning of all models to create transparency for stakeholders (explainability and reproducibility)
- Auditing of ML systems (automated approval auditing or CE certification as part of conformity testing)
- Comprehensive data documentation to guarantee high data quality and adherence to data protection
- Management of ML metadata
- Validation of ML models
- Continuous monitoring and logging of model metrics

7.12 Conclusion

The compliance of ML systems with legal requirements is by no means an abstract problem but one that can be solved technically because MLOps already provides

the appropriate infrastructure on which we can build to implement model governance processes. Moreover, it should be clear that we need MLOps and Model Governance to deploy ML models successfully.

8 What we offer

8.1 Consulting, Development, and Operations

The widespread hype around Machine Learning and Artificial Intelligence can create the impression that developing software with ML models is easy. However, various studies show that nearly 80% of ML projects fail. We are happy to support you in your machine learning development initiative—whether in the early exploration phase, during product development, or with all operational aspects.

AI Products with Domain-driven Design

Are you up to the challenge of developing innovative data-driven software solutions? Do you wonder where AI can be used in product development? Nowadays, there are countless models available through APIs and open-source solutions that can be used without having to train your own model. It's all there. Commodity AI is possible. How do you get started? Where in your product makes sense to integrate AI? What features are now possible that weren't achievable or too expensive before?

If this speaks to you, then this training is for you! Let our experienced trainers introduce you to the practical application of Artificial Intelligence and Machine Learning. Learn how to identify and validate AI/ML use cases, and gain an in-depth understanding of the right tools and strategies for successful implementation and deployment. Throughout the workshop, we use the Domain-driven Design methodology, which requires no prior knowledge.

More information: https://www.socreatory.com/en/trainings/ddd4ml

We advise honestly, think innovatively, and are passionate about development — the result: successful software solutions, infrastructures, and business models.

As a technology company, we focus on strategy and technology consulting, software architecture and development, methodology and technology training, and platform infrastructures.

With over 170 employees at locations in Germany and Switzerland, we support companies and organizations in designing and implementing complex projects and improving existing software systems.

We are involved in open-source projects and the iSAQB e.V., and pass on our knowledge and experience at conferences and meetings as well as in numerous books and professional articles.

About the authors

Alexander Kniesz

Alexander started in 2017 as a working student. Since 2021 he now works as a consultant at INNOQ. Since his master's degree in data science, his focus is on topics like data processing, machine learning, and AI. With those specializations, he is also interested in other software development topics to help bringing ML into production. Therefore he also assists in topics like Machine Learning Operations.

Anja Kammer

Anja Kammer is a Senior Consultant at INNOQ and supports companies on their journey to the cloud. In addition to providing advice on development processes and platforms, she develops cloud-native web applications in cross-functional teams. She is also an accredited trainer and co-curator for the iSAQB Advanced Level module CLOUDINFRA.

Dr. Larysa Visengeriyeva

Larysa Visengeriyeva is Head of Data and AI at INNOQ. She received her doctorate in Augmented Data Quality Management at the TU Berlin. She is working on the operationalization of Machine Learning (MLOps), data architectures such as Data Mesh and Domain-Driven Design.

@visenger

Training ML models can be time-consuming, but the real challenge lies in integrating an ML system into a production environment—in other words, into the software product that users interact with.

An ML system is comprised of three main elements: the training data, the ML model, and the code used to train the models. We apply DevOps principles to ML systems (MLOps) to combine the development and operations of ML. MLOps, as an extension of DevOps, focuses on automating and monitoring every step of integrating ML systems into software projects.

In this primer, we explain the fundamentals and principles of MLOps, aiming to provide insights into MLOps processes from an engineering perspective.

innoq.com

INNOQ