MLOPS

THIS DOCUMENT IS ONLY FOR INTERNAL REFERENCE DO NOT PUBLISH ON PUBLIC FORUM.

1. INTRODUCTION

MLOps, short for Machine Learning Operations, is a set of practices aimed at streamlining the deployment, monitoring, and management of machine learning models in production environments. Here are some key areas and best practices within MLOps

1. **Version Control**: Just like in traditional software development, version control is crucial in MLOps. Tools like Git help track changes to code, data, and model versions over time.

2. **Automation**: Automate repetitive tasks such as data preprocessing, model training, evaluation, and deployment using tools like Jenkins, Airflow, or Kubernetes.

3. **Model Packaging**: Package models along with their dependencies in a reproducible and portable manner using containerization tools like Docker. This ensures consistency across different environments.

4. **Continuous Integration/Continuous Deployment (CI/CD)**: Implement CI/CD pipelines to automate testing and deployment of models. This ensures that changes to models or underlying infrastructure are smoothly integrated and deployed without manual intervention.

5. **Model Monitoring and Logging**: Set up monitoring and logging mechanisms to track model performance, data drift, and other metrics in production. Tools like Prometheus, Grafana, or custom logging solutions can be used for this purpose.

6. **Infrastructure as Code (IaC)**: Define infrastructure requirements (e.g., computing resources, storage) as code using tools like Terraform or AWS CloudFormation. This allows for reproducible and scalable infrastructure deployment.

7. **Model Governance and Compliance**: Implement processes and tools to ensure compliance with regulatory requirements and organizational policies regarding data privacy, security, and ethical considerations.

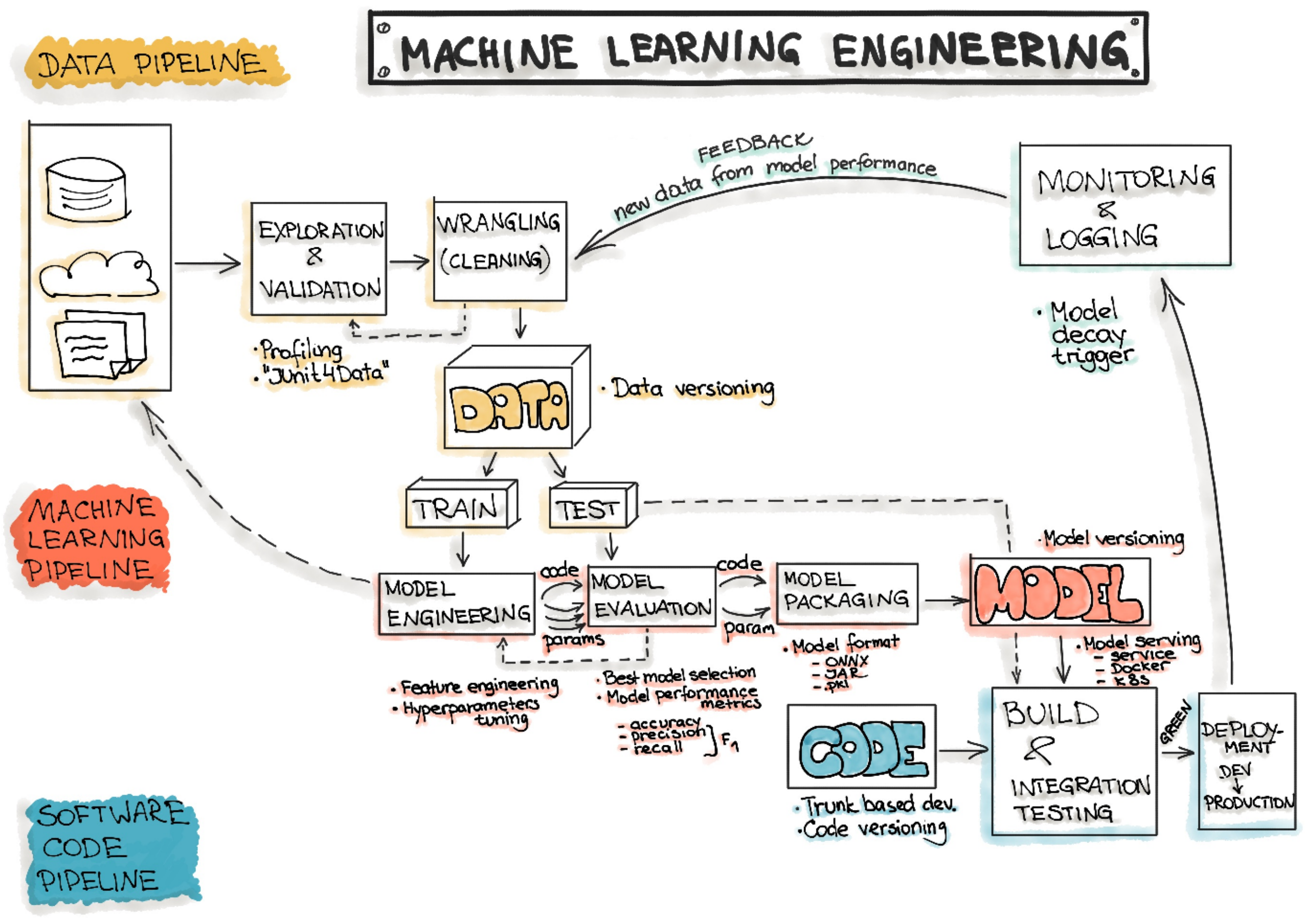
8. **Collaboration and Communication**: Foster collaboration between data scientists, engineers, and other stakeholders involved in the ML lifecycle. Clear communication and documentation of processes are essential for successful MLOps implementation.

9. **Feedback Loop and Iteration**: Establish feedback loops to gather insights from model performance in production and use them to improve future iterations of the model. This involves continuously monitoring model performance and iterating based on insights gained.

10. **Security**: Ensure that proper security measures are in place to protect sensitive data and prevent unauthorized access to models or infrastructure components.

The main aim of a machine learning project is to create a smart model using gathered information and special computer programs, but it's not easy to make these projects work well. Every machine learning program has to handle three important things: **data, model, and code.**

2. THREE LEVELS OF ML SOFTWARE



Essential technical methodologies - involved in the development of the Machine Learning-based software

- ✓ **Data Engineering**: data acquisition & data preparation,
- ✓ **ML Model Engineering**: ML model training & serving, and
- ✓ **Code Engineering**: integrating ML model into the final product.

1.1 Data Engineering

The initial step in any data science workflow involves **acquiring** and **preparing** the data for analysis. Typically, data is sourced from various **resources** and arrives in different **formats**.

DATA PREPARATION:

According to Gartner, data preparation is an **iterative** and **agile process** involving **exploration**, **combination**, **cleaning**, and **transformation** of **raw data** into **curated datasets** for various use cases such as data integration, data science, and analytics/business intelligence (BI). It's reported to be the most **resource** and **time-intensive phase**.

Data preparation is **critical** in the data science workflow as it's essential to prevent the **propagation** of data errors to subsequent phases, particularly **data analysis**, which could lead to **incorrect insights** derived from the data.

Sequence of operations on the data leading to training and testing datasets for ML algorithms:

A Data Ingestion:

- Involves **collecting data** using various frameworks and formats like Spark, HDFS, CSV, etc.
- May include tasks like **synthetic data generation** or **data enrichment**.

B Data Exploration:

- Includes **data profiling** to gather information about the **content** and **structure** of the data.
- Outputs **metadata** such as **maximum**, **minimum**, and **average values**.

C Data Validation:

- Utilizes **user-defined error detection functions** to **scan** the dataset and identify **errors**.

D Data Wrangling (Cleaning):

- Involves **re-formatting** specific **attributes** and **rectifying data errors** such as **missing values imputation**.

E Data Labeling:

- An operation within the **Data Engineering pipeline** where each **data point** is assigned to a **specific category**.

F Data Splitting:

- Involves dividing the data into **training**, **validation**, and **test datasets**. These subsets are crucial during the **core machine learning stages** to produce the **ML model**.

This sequence of operations ensures that data is properly prepared and structured to be used effectively in machine learning algorithms, thereby facilitating the generation of **accurate** and **reliable models**.

1.2 Model Engineering

Phase of writing and executing **machine learning algorithms** to obtain an **ML model**. The **Model Engineering pipeline** includes a number of operations that lead to a final model:

A Model Training

- The process of applying the **machine learning algorithm** on **training data** to train an ML model.
- Includes **feature engineering** and **hyperparameter tuning** for the model training activity.

B Model Evaluation

- Validating the trained model to ensure it meets **original codified objectives** before serving the ML model in **production** to the **end-user**.

C Model Testing

- Performing the final **"Model Acceptance Test"** by using the **hold backtest dataset**.

D Model Packaging

- The process of exporting the final ML model into a specific format (e.g., **PMML**, **PFA**, or **ONNX**).
- Which describes the model, in order to be consumed by the **business application**.

1.3 Model Deployment

Need to deploy model as part of a **business application** such as a **mobile** or **desktop application**.

- The ML models require various **data points** (**feature vector**) to produce **predictions**.
- The final stage of the ML workflow is the integration of the previously engineered ML model into existing software.

Includes the following operations:

- **Model Serving**
  - The process of addressing the ML model artifact in a **production environment**.

- **Model Performance Monitoring**
  - The process of observing the ML model performance based on **live** and previously **unseen data**, such as **prediction** or **recommendation**.
  - Interested in **ML-specific signals**, such as **prediction deviation** from previous model performance.
  - Signals might be used as triggers for **model re-training**.

- **Model Performance Logging**
  - Every **inference request** results in the **log-record**.

3. MACHINE LEARNING LIFECYCLE

Cyclic iterative process with **instructions**, and **best practices** to use across defined **phases** while developing an **ML workload**.

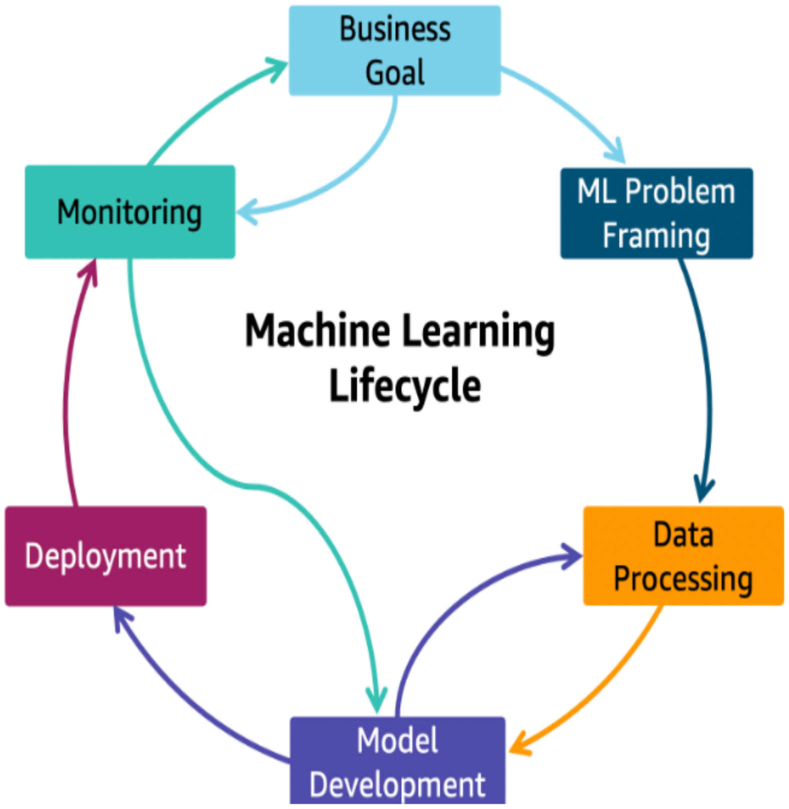
- Adds **clarity** and **structure** for making a machine learning project successful.

• The **end-to-end machine learning lifecycle process** includes the following **phases**:

- **Business goal identification**
- **ML problem framing**
- **Data processing** (**data collection**, **data preprocessing**, **feature engineering**)
- **Model development** (**training**, **tuning**, **evaluation**)
- **Model deployment** (**inference**, **prediction**)
- **Model monitoring**

The phases of the ML lifecycle are not necessarily **sequential** in nature and can have **feedback loops** to interrupt the cycle across the lifecycle phases.

Phases



3.1 BUSINESS GOAL

Should have a clear idea of the problem, and the **business value** to be gained by solving that problem.

Must be able to measure business value against specific **business objectives** and **success criteria**.

The **most important phase**, particularly challenging when considering ML solutions because ML is a constantly evolving technology!. After **success criteria** is defined, evaluate the organization's ability to move toward that target.

- The target should be **achievable** and provide a clear path to production.
- Involve all **relevant stakeholders** from the beginning to align them to this target.
- Include any **new business processes** that will result from this initiative.

Steps in this phase:

- Understand **business requirements**.
- Form a **business question**.
- Review a project's ML feasibility and **data requirements**.
- Evaluate the **cost** of data acquisition, training, inference, and wrong predictions.
- Review **proven or published work** in similar domains, if available.
- Determine key **performance metrics**, including acceptable errors.
- Define the **machine learning task** based on the business question.
- Identify **critical, must-have features**.
- Design small, focused **POCs** to validate all of the preceding.
- Evaluate if bringing in **external data sources** will improve model performance.
- Establish **pathways to production**.
- Consider new **business processes** that may come out of this implementation.
- Align relevant stakeholders with this initiative.

3.2 ML PROBLEM FRAMING

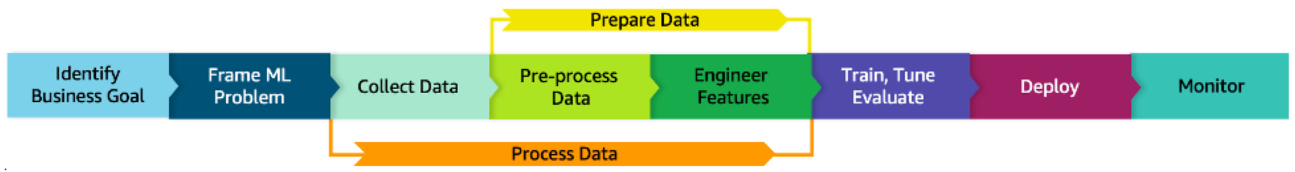
The business problem is framed as a **machine learning problem**.

- What is observed and what should be predicted (known as a **label** or **target variable**).
- Determining what to predict and how **performance** and **error metrics** must be optimized is a key step in this phase.

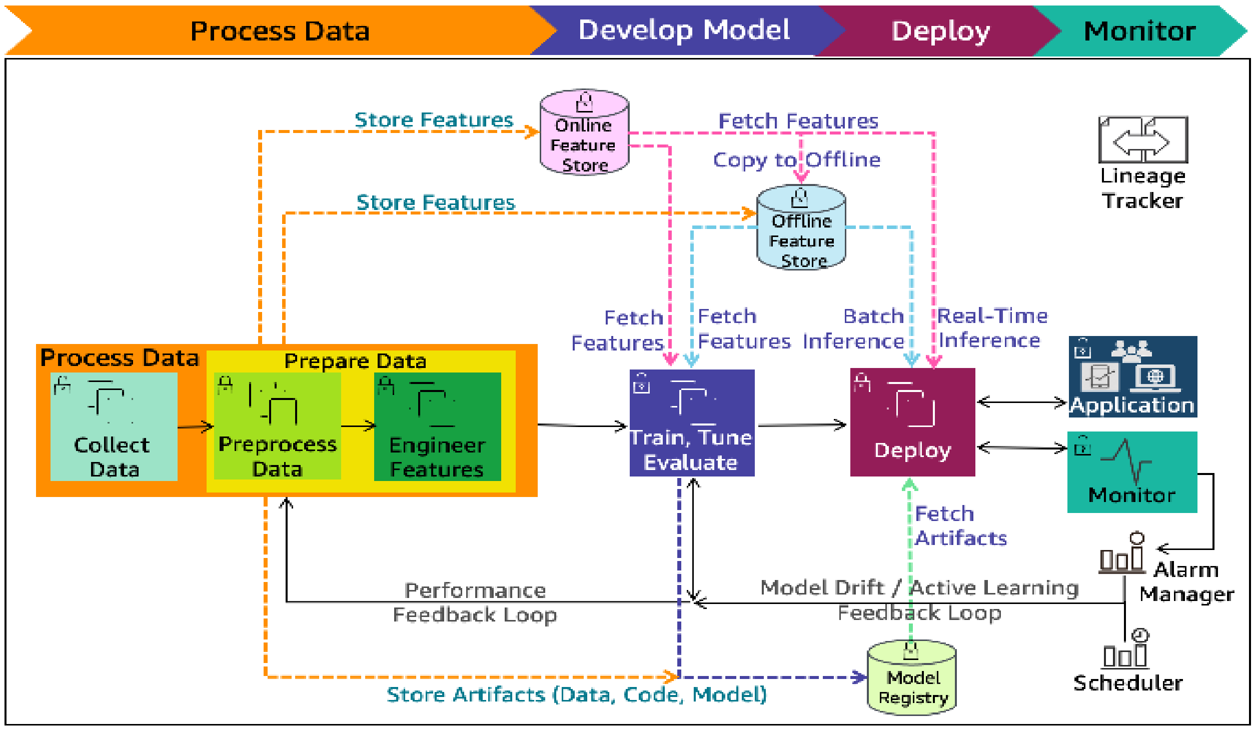
Steps in this phase:

- Define criteria for a **successful outcome** of the project.
- Establish an **observable** and **quantifiable performance metric** for the project, such as **accuracy**.
- Help ensure **business stakeholders** understand and agree with the defined **performance metrics**.
- Formulate the ML question in terms of **inputs**, **desired outputs**, and the **performance metric** to be optimized.
- Evaluate whether ML is the right approach.
    - Some **business problems** don't need ML; simple **business rules** can do a much better job.
    - For other **business problems**, there might not be sufficient **data** to apply ML as a solution.
- Create a strategy to achieve the **data sourcing** and **data annotation** objective.
- Start with a **simple model** that is easy to interpret, and which makes **debugging** more manageable.

1.11 ML LIFECYCLE ARCHITECTURE



ML lifecycle with data processing sub-phases included



ML lifecycle with detailed phases and expanded components

- **Online/Offline feature store**
  - Reduces **duplication** and **rerun** of **feature engineering** code across teams and projects.
  - **Online store** with **low-latency retrieval capabilities** is ideal for **real-time inference**.
  - **Offline store** should maintain a history of feature values and is suited for **training** and **batch scoring**.

- **Model registry**
  - A repository for storing ML model artifacts including **trained model** and related metadata (**data**, **code**, **model**).
  - Enables **lineage** for ML models as it can act as a **version control system**.

- **Performance feedback loop**
  - Automates model performance evaluation tasks initiated from the model development to data processing phase.

- **Model drift feedback loop**
  - Automates model update re-training tasks initiated from the production deployment to data processing phase.

• **Alarm manager**
- Receives the alerts from the **model monitoring system**.
- Runs actions by publishing **notifications** to services that can deliver alerts to target applications to handle them.
- The model update re-training pipeline is one such **target application**.

• **Scheduler**
- A scheduler can initiate a re-training at **business defined intervals**.

• **Lineage tracker**
- Enables **reproducible machine learning experiences**.
- Enables re-creating the ML environment at a specific point-in-time, reflecting the versions of all resources and environments at that time.
- Collects references to traceable **data**, **model**, and **infrastructure resource changes**.

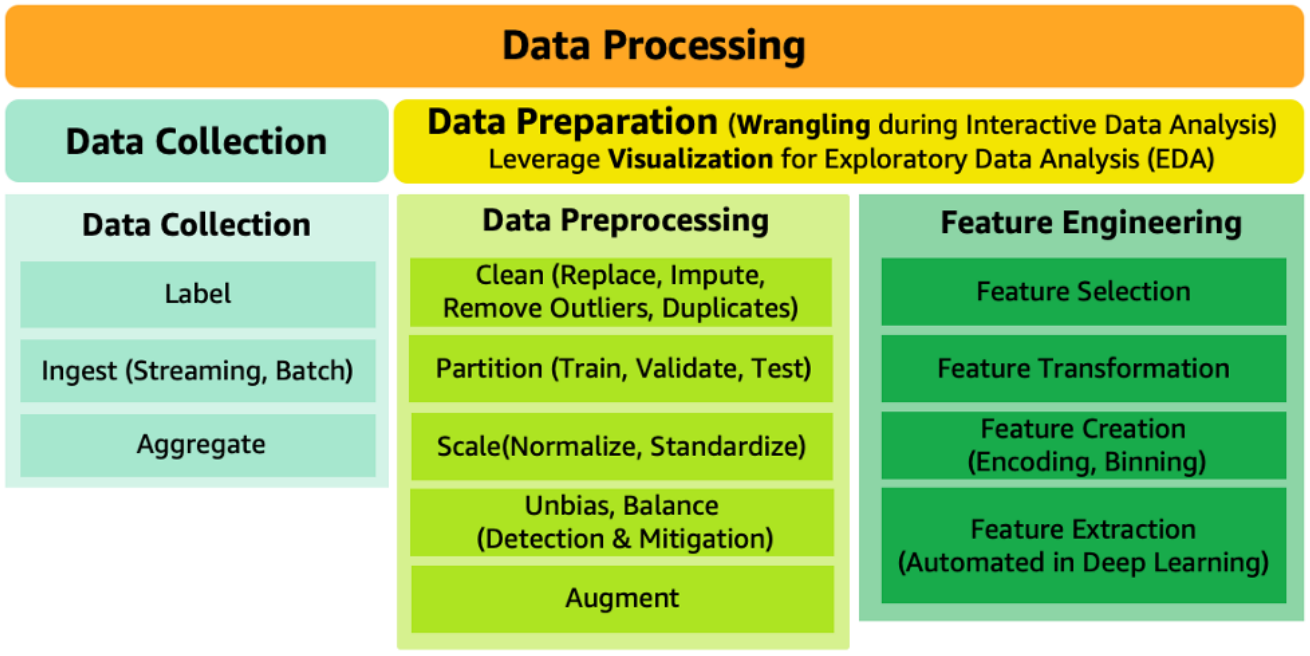
3.3 DATA PROCESSING

Training an accurate ML model requires **data processing** to convert data into a **usable format**.
• **Includes** collecting data, preparing data, and **feature engineering** that is the process of **creating**, **transforming**, **extracting**, and **selecting variables** from data.

In ML workloads, the data (inputs and corresponding desired output) serves **important functions** including:

- Defining the **goal** of the system: the output representation and the relationship of each output to each input, by means of input/output pairs.
- **Training** the algorithm that associates inputs to outputs.
- **Measuring** the **performance** of the trained model, and **evaluating** whether the performance target was met.

- **Building baselines** to monitor the performance of the models deployed to production.


Data Processing
Data Collection
Data Preparation (Wrangling during Interactive Data Analysis)
Leverage Visualization for Exploratory Data Analysis (EDA)
Data Collection
Label
Ingest (Streaming, Batch)
Aggregate
Data Preprocessing
Clean (Replace, Impute, Remove Outliers, Duplicates)
Partition (Train, Validate, Test)
Scale(Normalize, Standardize)
Unbias, Balance (Detection & Mitigation)
Augment
Feature Engineering
Feature Selection
Feature Transformation
Feature Creation (Encoding, Binning)
Feature Extraction (Automated in Deep Learning)

Data processing includes **data collection** and **data preparation**.

- Data preparation includes **data preprocessing** and **feature engineering**.
- Data wrangling is data preparation during the interactive data analysis and model development.
- Data Visualization can help with **exploratory data analysis (EDA)**.
- EDA can help with understanding data, **sanity checks**, and validating the quality of the data.

The same sequence of data processing steps that you apply to the **training data** is also applied to the **inference requests**.

3.3.1 DATA COLLECTION

One of the first steps in the ML lifecycle is to identify what data is needed. Then evaluate the various means available for **collecting** that data to train the model.

Activities in the data collection phase:

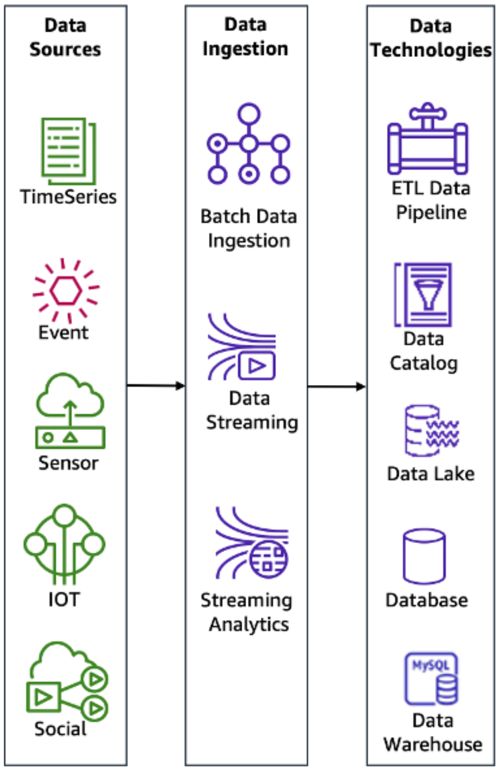
- **Label** — Data for which already know the target answer is called **labeled data**.
- If labels are missing, then some effort is required to label it (manual or automated).
- **Ingest & Aggregate**: Data collection includes ingesting and aggregating data from multiple data sources.
- Some components of the ingest and aggregate:
    - **Data sources** include time-series, events, sensors, IoT devices, and social networks, depending on the nature of the use case.
    - **Data ingestion** processes and technologies capture and store data on storage media.
    - Can occur in real-time using streaming technologies or historical mode using batch technologies.
    - **Data storage technologies** vary from transactional (SQL) databases, to data lakes and data warehouses.
    - **ETL pipeline technology** automates and orchestrates the data movement and transformations across cloud services and resources.
    - A **data lake technology** enables storing and analyzing structured and unstructured data.

Data Collection

Label

Ingest (Streaming, Batch)

Aggregate



3.3.2 DATA PREPARATION

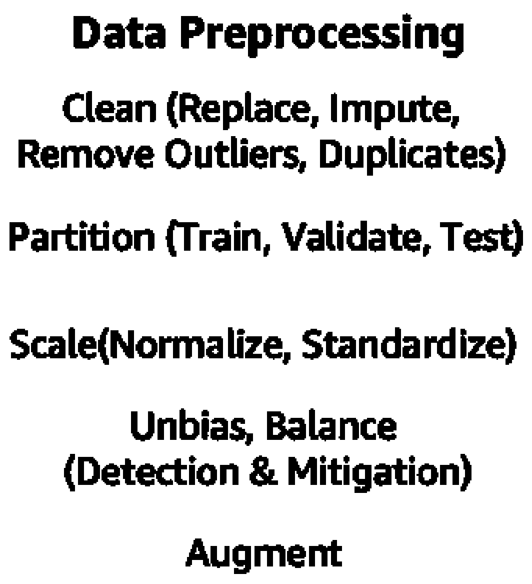
3.3.2.1 Data preparation

ML models are only as good as the **data** that is used to train them. **Ensure** that suitable **training data** is available and is **optimized** for learning and **generalization**.

Data preparation includes **data pre-processing** and **feature engineering**.

- A **key aspect** to understanding data is to **identify patterns** which are often not evident with data in tables.
- Exploratory data analysis (**EDA**) with visualization tools can help quickly gain a deeper understanding of data.
- Prepare data using **wrangler tools** for **interactive data analysis** and **model building**.
- The **no-code/low-code**, **automation**, and **visual capabilities** improve the **productivity** and **reduce** the **cost** for interactive analysis.

Data pre-processing puts data into the **right shape** and **quality** for training.
Many **strategies**: **datacleaning**, **balancing**, **replacing**, **imputing**, **partitioning**, **scaling**, **augmenting**, and **unbiasing**.



Data Preprocessing

Clean (Replace, Impute, Remove Outliers, Duplicates)

Partition (Train, Validate, Test)

Scale(Normalize, Standardize)

Unbias, Balance (Detection & Mitigation)

Augment

Clean (replace, impute, remove **outliers** and **duplicates**)

- Remove **outliers** and **duplicates**, replace inaccurate or irrelevant data.
- Correct missing data using imputation techniques that will minimize **bias** as part of data cleaning.

Partition

- To prevent ML models from **overfitting** and evaluate trained model accurately, randomly split data into **train**, **validate**, and **test sets**.
- Care is needed to avoid **data leakage**.
- Data leakage happens when information from hold-out test dataset leaks into the training data.
- One way to avoid data leakage is to **remove duplicates** before splitting of the data.

Scale (normalize, standardize)

- Having features on a similar scale close to **normally distributed** will ensure that each feature is equally important.
- Make it easier for most ML algorithms including K-Means, KNN, PCA, gradient descent.
- **Standardization** better handles the **outliers**.

Unbias, balance (detection & mitigation)

- Detecting and mitigating **bias** will help avoiding inaccurate model results.
- Biases are imbalances in the accuracy of predictions across different groups, such as age or income bracket.
- Biases can come from the data or algorithm used to train your model.

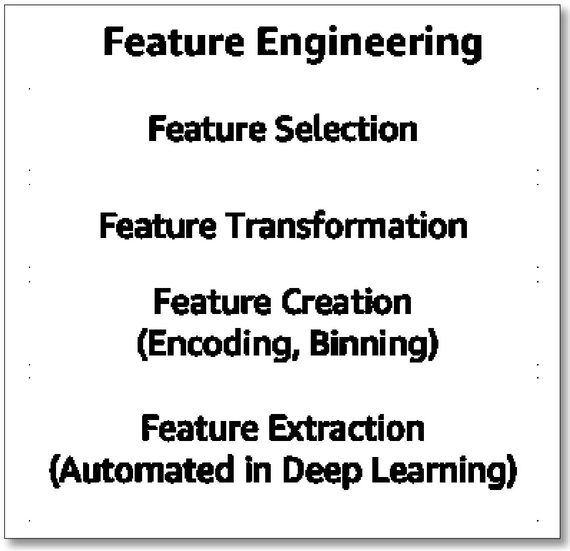
Augment

- Increases the amount of data artificially by **synthesizing** new data from existing data.
- Can help **regularize** and **reduce overfitting**.

3.3.2.2 FEATURE ENGINEERING

Every unique attribute of the data is considered a **feature**.
Feature engineering is a process to **select** and **transform variables** when creating a **predictive model** using machine learning or statistical modelling.

- Includes **feature creation**, **feature transformation**, **feature extraction**, and **feature selection**.
- With **deep learning**, the feature engineering is **automated** as part of the algorithm learning.


Feature Engineering
Feature Selection
Feature Transformation
Feature Creation (Encoding, Binning)
Feature Extraction (Automated in Deep Learning)

• **Feature creation**

- Is creating new features from existing data to help with better predictions.
- Examples of feature creation techniques include: **one-hot-encoding**, **binning**, **splitting**, and **calculated features**.

• **Feature transformation and imputation**

- Manage replacing **missing features** or features that are not valid.
- Some techniques include: forming Cartesian products of features, **non-linear transformations** (such as binning numeric variables into categories), and creating **domain-specific features**.

- **Feature extraction**
  - Involves reducing the amount of data to be processed using **dimensionality reduction techniques**.
  - Reduces the amount of **memory** and **computing power** required, while still accurately maintaining original data characteristics.
  - Techniques include: **PCA**, **ICA**, and **LDA**.
- **Feature selection**
  - Process of selecting a subset of extracted features which is relevant and contributes to minimizing the **error rate** of a trained model.
  - **Feature importance score** and **correlation matrix** can be factors in selecting the most relevant features for model training.

3.4 MODEL DEVELOPMENT

Consists of **model building**, **training**, **tuning**, and **evaluation**. Includes creating a **CI/CDpipeline** that **automates** the **build**, **train**,and **release** to **staging** and **production environments**.

Model building, training, tuning
In this phase, select a **machine learning algorithm** that is appropriate for the problem and then **train** the ML model.

- Provide the algorithm with the **training data**.
- Set an **objective metric** for the ML model to optimize on.
- Set the **hyperparameters** to optimize the training process.

Model training, **tuning**, and **evaluation** require prepared data and **engineered features**.

Feature selection: Features are selected as part of the data processing after a **bias strategy** is implemented.

Building code: Model development includes building the algorithm and its supporting code.

- Should support **version control**, and **continuous build**, **test**, and **integration** through a pipeline.

Algorithm selection: Selecting the right algorithm involves running many experiments with parameter tunings across available options.

- Factors to consider when evaluating each option can include **accuracy**, **explainability**, **training/prediction time**, **memory requirements**.

• **Model training** (data training parallel, model training parallel):

- The process of training an ML model involves providing an ML algorithm with training data to learn from.
- **Distributed training** enables splitting large models and training datasets across computing instances to reduce runtime to a fraction of it takes to do manually.
- **Model parallelism** is the process of splitting a model up between multiple devices or nodes.
- **Data parallelism** is the process of splitting the training set into mini-batches evenly distributed across nodes; each node only trains the model on a fraction of the total dataset.

• **Debugging/profiling**:

- A machine learning training job can have problems including: system bottlenecks, **overfitting**, saturated activation functions, and vanishing gradients. These problems can compromise model performance.
- A debugger provides **visibility** into the ML training process through **monitoring**, **recording**, and **analyzing data**; captures the state of a training job at periodic intervals.

• **Validation metrics**:

- Typically, a training algorithm computes several metrics, such as **loss** and **prediction accuracy**; determine if the model is learning and generalizing well for making predictions on unseen data.
- Metrics reported by the algorithm depend on the business problem and the ML technique used.

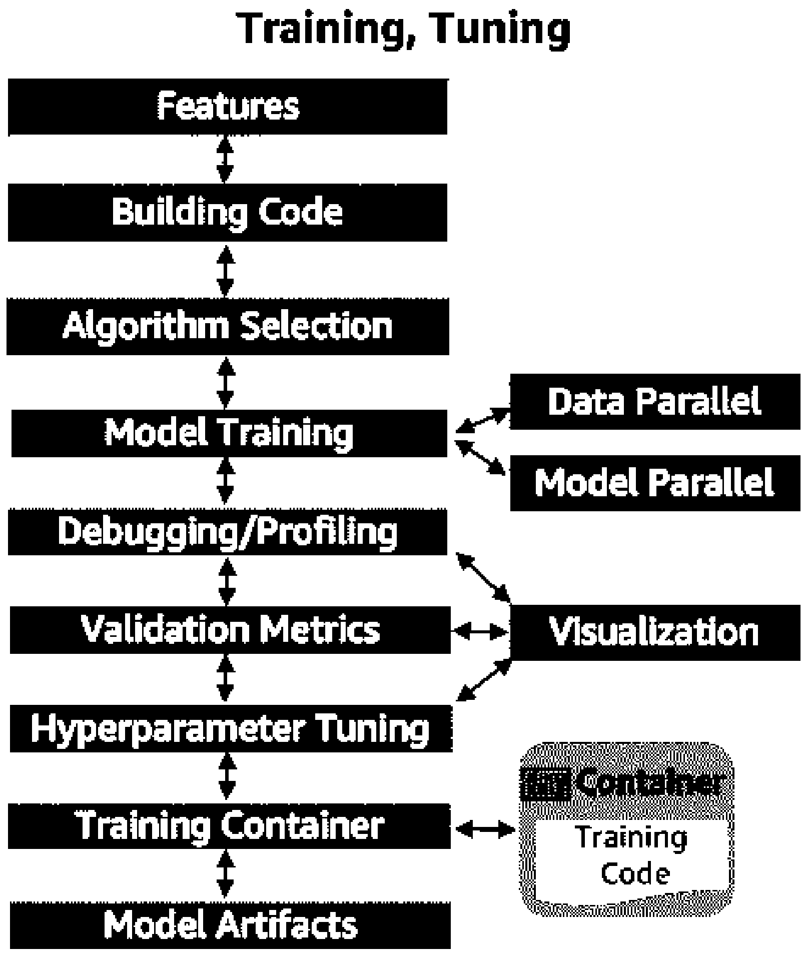
• **Hyperparameter tuning**:

- Settings that can be tuned to control the behavior of the ML algorithm are referred to as **hyperparameters**.
- The number and type of hyperparameters in ML algorithms are specific to each model.
- Examples of commonly used hyperparameters include: **learning rate**, **number of epochs**, **hidden layers**, **hidden units**, and **activation functions**.

- **Hyperparameter tuning**, or optimization, is the process of choosing the optimal hyperparameters for a learning algorithm.

- **Training code container**:
  - Create container images with training code and its entire dependency stack; will enable training machine learning algorithms and deploy models quickly and reliably at any scale.
- **Model artifacts**:
  - Model artifacts are the output that results from training a model; typically consist of trained parameters, a model definition that describes how to compute inferences, and other metadata.

- **Visualization**:
  - Enables exploring and understanding data during metrics validation, debugging, profiling, and hyperparameter tuning.



ML LIFECYCLE WITH PRE-PRODUCTION PIPELINES

- The **data prepare pipeline automates** data preparation tasks.
- The **feature pipeline automates** the **storing**, **fetching**, and **copying** of the features into and from **online/offline store**.
- The **CI/CD/CT pipeline automates** the **build**, **train**, and **release** to **staging** and **production environments**.

MODEL EVALUATION

After the model has been trained, **evaluate** it for its **performance** and **accuracy**.

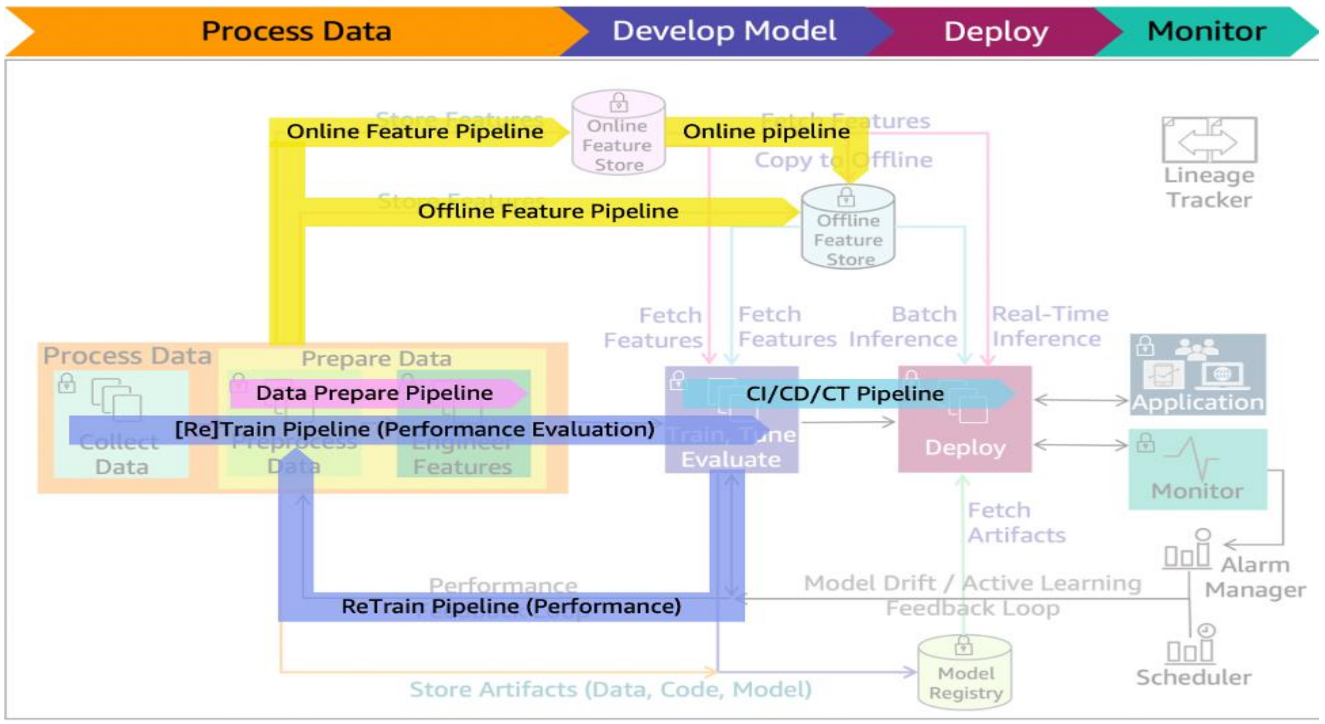
- **Generate** multiple models using different methods and **evaluate** the **effectiveness** of each model.
- For **multiclass models**, determine **error rates** for each class separately.
- Can **evaluate** model using **historical data** (**offline evaluation**) or **live data** (**online evaluation**).

• **Offline evaluation**

- The trained model is **evaluated** with a portion of the dataset that has been set aside as a **holdout set**.
- Never used for model **training** or **validation**—it's only used to **evaluate errors** in the final model.
- The **holdout data annotations** must have **high accuracy** for the **evaluation** to make sense.
- Allocate additional resources to verify the **accuracy** of the **holdout data**.

• Based on the **evaluation results**, might **fine-tune** the **data**, the **algorithm**, or both.

- May apply the concepts of **data cleansing**, **preparation**, and **feature engineering** again.

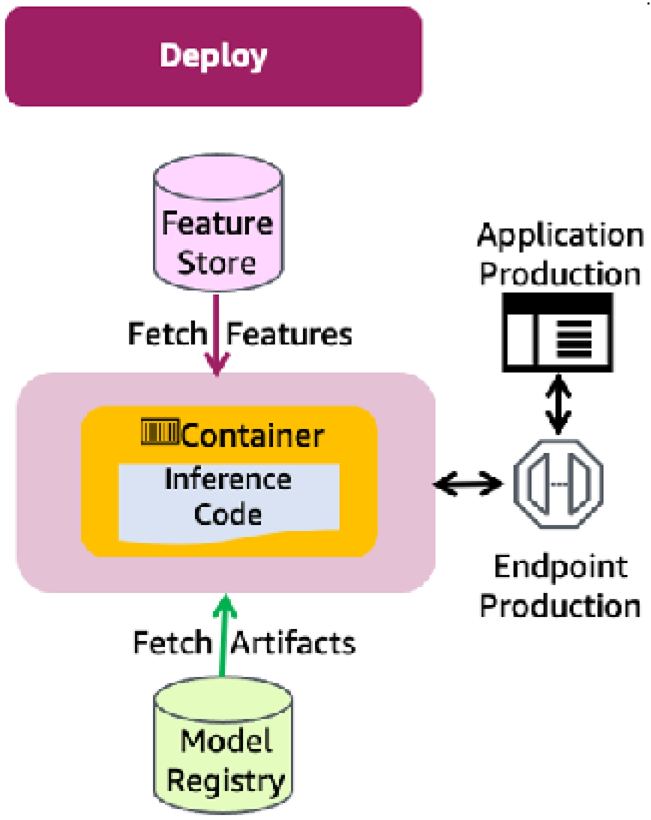


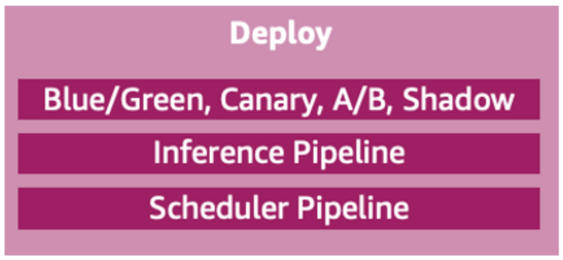
3.4 MODEL DEPLOYMENT

After the model has been **trained**, **tuned**, and **evaluated**, then can deploy the model into **production**. Can make **predictions** and **inferences** against the deployed model. Use a **manual governance process** - ensures the model has fully been **tested** and **evaluated** before it's released to production.

Deploy production phase of the ML lifecycle:

- **Features** are retrieved from the **feature store**.
- **Artifacts** are taken from the **model registry**.
- **Inference** code used container from the **container repository**.





Blue/green, **canary**, **A/B**, **shadow deployment/testing**:

- Deployment and testing strategies that reduce **downtime** and **risks** when releasing a new or updated version.
- The **blue/green deployment technique** provides two identical production environments. Can use this technique when need to deploy a new version of the model to production. Once testing is done on the green environment, live application traffic is directed to the green environment. Blue environment is maintained as backup.
- A **canary deployment** first deploys the new release to a small group of users. Other users continue to use the previous version until satisfied with the new release. Then, can gradually roll the new release out to all users.
- An **A/B testing strategy** enables deploying changes to a model. Direct a defined portion of traffic to the new model. Direct the remaining traffic to the old model, similar to canary testing, but has larger user groups and a longer time scale, typically days or even weeks.
- The **shadow deployment strategy** involves having the new version available alongside the old version. The input data is run through both versions. The older version is used for servicing the production and the new one is used for testing and analysis.

ML lifecycle with scheduler re-train, and batch/real-time Inference pipelines

- **Inference pipeline** automates capturing of the prepared data, performing **predictions** and **post-processing** for real-time or batch inferences.
- **Scheduler pipeline** ensures that the deployed model is representative of the latest **data patterns**. Re-training at intervals can minimize the risk of **data** and **concept drifts**. A scheduler can initiate a re-training at business defined intervals. Data prepare, CI/CD/CT, and feature pipelines will also be active during this process.

3.5 MONITORING

Ensures **model** is maintaining a **desired level** of **performance** through **early detection** and **mitigation**.

ML lifecycle phase – Monitoring

Post deployment monitor main components

• The **model monitoring system** must:

- **Capture data**,
- **Compare** that data to the **training set**,
- Define **rules** to detect **issues**,
- And **send alerts**.

• Process repeats on a **defined schedule**, when initiated by an **event**, or when initiated by **human intervention**.

- The issues detected in the monitoring phase include: **data quality**, **model quality**, **bias drift**, and **feature attribution drift**.

Key components of monitoring including:

• **Model explainability**:

- Uses explainability to evaluate the **soundness** of the model and if the **predictions** can be **trusted**.

• **Detect drift**:

- Detects **data** and **concept drifts**.
- **Data drift** is the significant changes to data distribution compared to data used for training.
- **Concept drift** is when the properties of target variables change.
- Any kind of drift will result in **model performance degradation**.
- Will initiate an **alert** and send it to **alarm manager system**.

• **Model update pipeline**:

- If alarm manager identifies any **violations**, it launches the **model update pipeline** for a **re-train**.
- **Data prepare**, **CI/CD/CT**, and **feature pipelines** will also be active during this process.

3.6 ADDITIONAL - LINEAGE TRACKER

Requirement:

- Enables **reproducible** machine learning experiences.
- Enables **re-creating** the ML environment at a specific point-in-time, reflecting the **versions** of all resources and environments at that time.
- Collects references to **traceable data**, **model**, and **infrastructure resource changes**.

Components:

- **System architecture** (infrastructure as code to address environment drift).
- **Data** (metadata, values, and features).
- **Model** (algorithm, features, parameters, and hyperparameters).
- **Code** (implementation, modeling, and pipeline).

Working:

- Collects changed references through alternative iterations of ML lifecycle phases.
- Alternative algorithms and feature lists are evaluated as experiments for final production deployment.
- The collected information enables going back to a specific point-in-time release and **recreate** it.

Infrastructure as code (IaC):

- Modeling, provisioning, and managing cloud computing resources (compute, storage, network, and application services) can be **automated** using IaC.
- Eliminates **configuration drift** through automation, while increasing the speed and agility of infrastructure deployments.
- IaC code changes are committed to **version-controlled repository**.

Data:

- Store data schemes and metadata in **version control systems**.
- Store the data in a storage media like a **data lake**.
- The location or link to the data can be in a configuration file and stored in code version control media.

Implementation code:

- Changes to any implementation code at any point-in-time can be stored in version control media.

Model feature list:

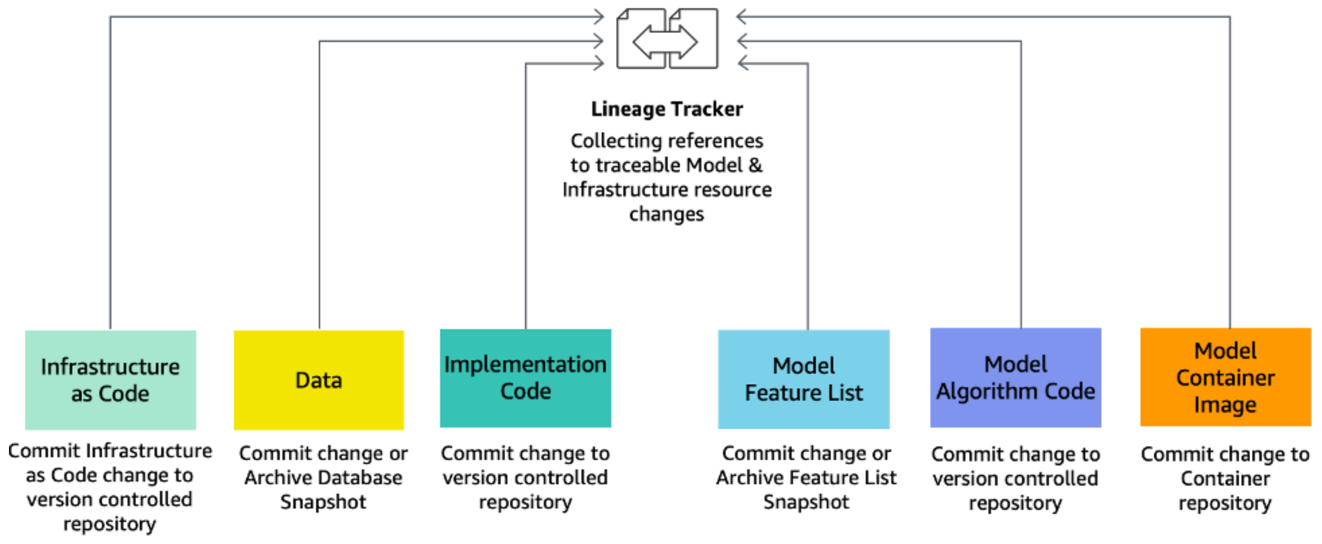
- Feature store technology referenced in the scenario architecture diagrams stores features as well as their versions for any point-in-time changes.

Model algorithm code:

- Changes to any model algorithm code at any point-in-time can be stored in version control media.

Model container image:

- Versions of model container images for any point-in-time changes can be stored in container repositories managed by container registry.


Lineage Tracker
Collecting references to traceable Model & Infrastructure resource changes
Infrastructure as Code
Commit Infrastructure as Code change to version controlled repository
Data
Commit change or Archive Database Snapshot
Implementation Code
Commit change to version controlled repository
Model Feature List
Commit change or Archive Feature List Snapshot
Model Algorithm Code
Commit change to version controlled repository
Model Container Image
Commit change to Container repository

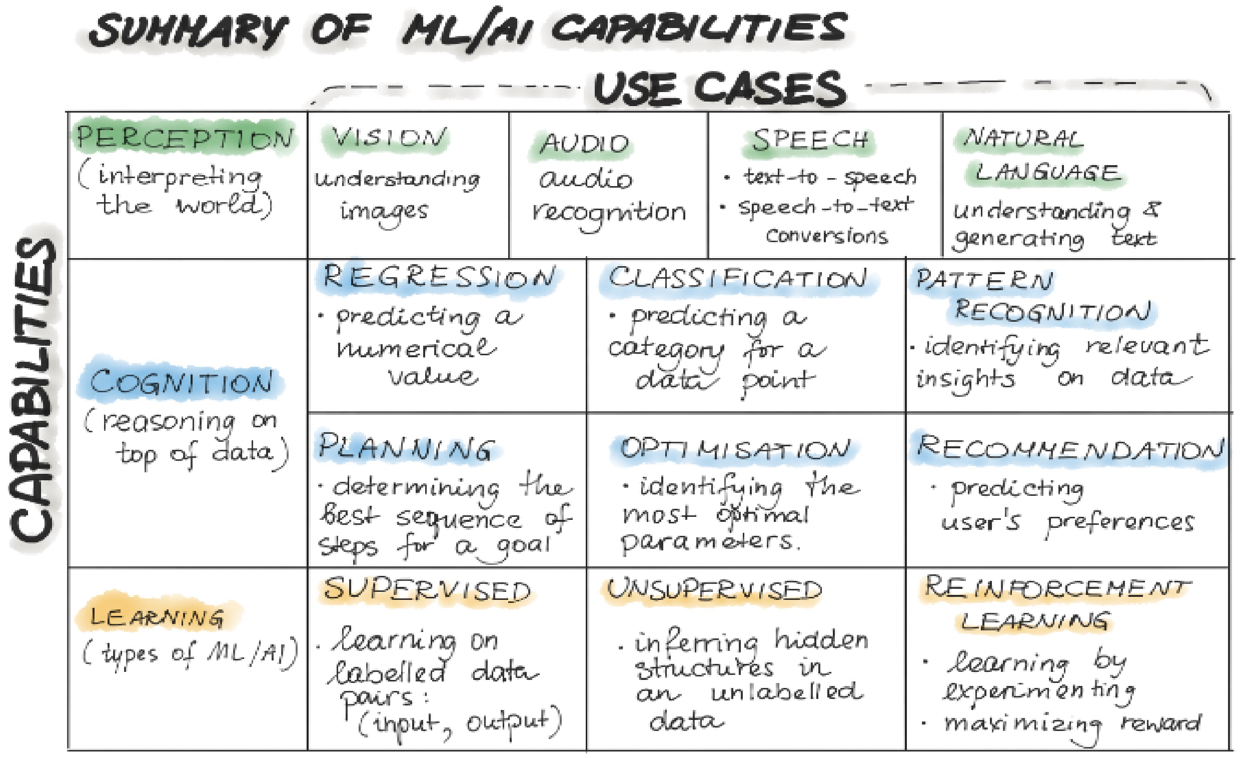
4. MOTIVATION AND DRIVERS FOR MLOPS

4.1 WHY YOU MIGHT WANT TO USE MACHINE LEARNING?

- Per Statista Digital Economy Compass 2019, two major trends are disrupting 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.
- 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.
    - Might think of the ML model as a **function** that takes some input data and produces an output (e.g., classification, sentiment, recommendation, or clusters).
- The performance of each model is evaluated by using **evaluation metrics**, such as **precision & recall**, or **accuracy**.

4.2 MODEL DEPLOYMENT

- **More and more enterprises are experimenting with ML**.
- Getting a model into the real world involves more than just building it.
    - Need to incorporate the trained ML model into the **core codebase**.
    - Need to **deploy** the ML model into **production**.
    - Only by deploying models, other software systems can **supply data** to these, get **predictions** which are in turn populated back into the software systems.
- The full advantage of ML models is only possible through **ML model deployment**.



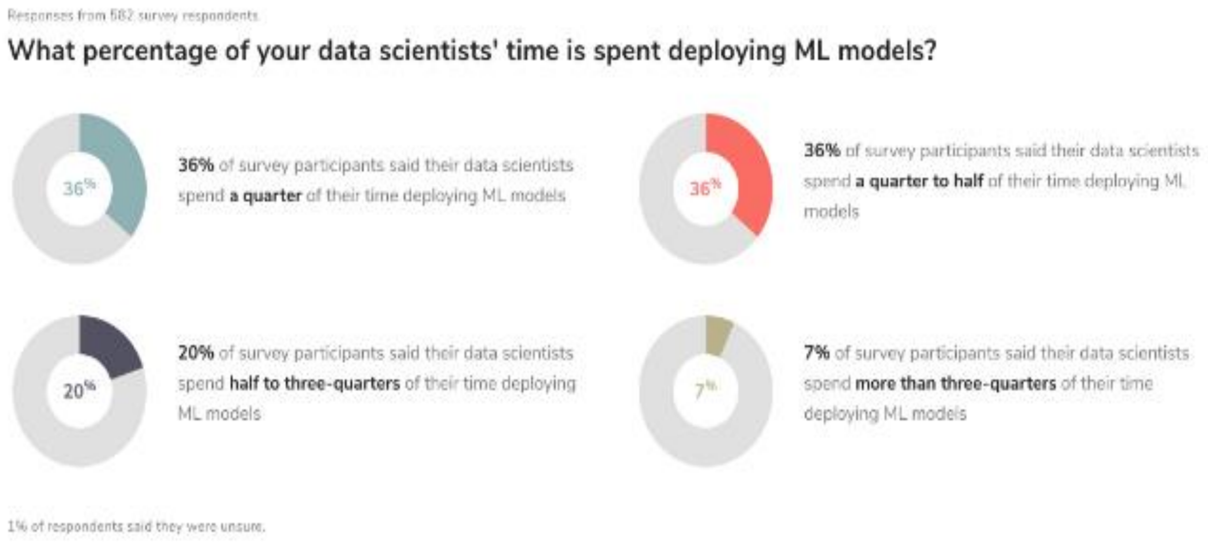
SUMMARY OF ML/AI CAPABILITIES

CAPABILITIES	USE CASES			
PERCEPTION (interpreting the world)	VISION understanding images	AUDIO audio recognition	SPEECH • text-to-speech • speech-to-text conversions	NATURAL LANGUAGE understanding & generating text
COGNITION (reasoning on top of data)	REGRESSION • predicting a numerical value	CLASSIFICATION • predicting a category for a data point	PATTERN RECOGNITION • identifying relevant insights on data	
	PLANNING • determining the best sequence of steps for a goal	OPTIMISATION • identifying the most optimal parameters.	RECOMMENDATION • predicting user's preferences	
LEARNING (types of ML/AI)	SUPERVISED • learning on labelled data pairs: (input, output)	UNSUPERVISED • inferring hidden structures in an unlabelled data	REINFORCEMENT LEARNING • learning by experimenting • maximizing reward	

Table Source: "The AI Organization" by David Carmona

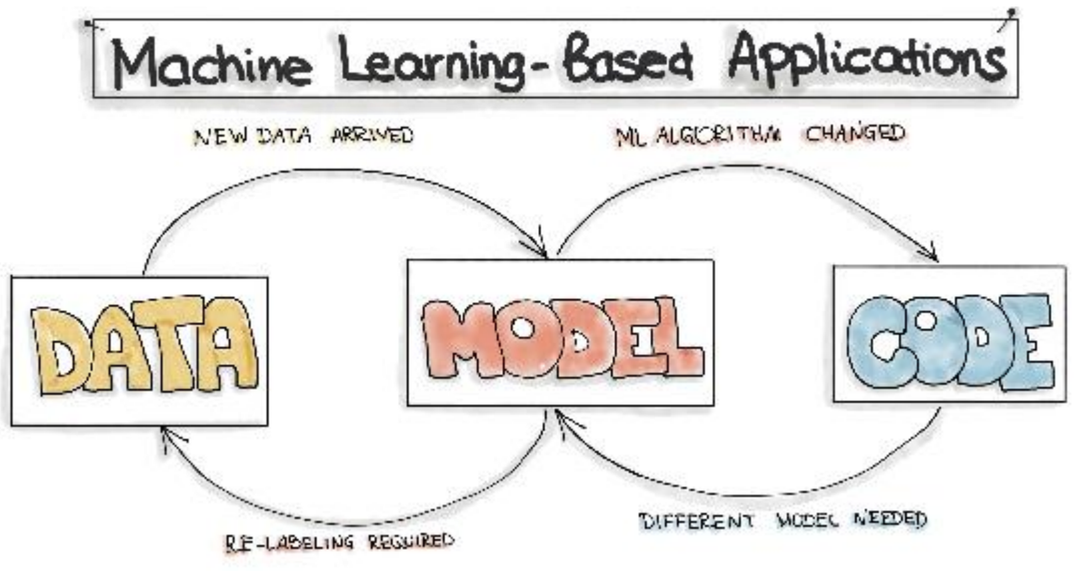
4.3 DEPLOYMENT GAP

- According to a report by Algorithmia "2020 State of Enterprise Machine Learning",
  - A survey of nearly 750 people including **machine learning practitioners**, **managers for machine learning projects**, and **executives** at tech firms.
  - Many companies haven't figured out how to achieve their **ML/AI goals**.
  - Only **22 percent** of companies that use machine learning have successfully deployed an ML model into **production**.
- Because bridging the gap between **ML model building** and practical **deployments** is still a **challenging task**!
- Per survey:
  - 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**.
  - The main challenges people face when developing ML capabilities are **scale**, **version control**, **model reproducibility**, and **aligning stakeholders**.



4.4 SCENARIOS OF CHANGE THAT NEED TO BE MANAGED

- The reason for the **deployment gap** is that the development of machine learning-based applications is fundamentally different from the development of traditional software.
- The complete development pipeline includes **three levels of change**:
  - **Data**
  - **ML Model**
  - **Code**.
- 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**.
- Aka the **"Changing Anything Changes Everything"** principle.



4.5 ISSUES INFLUENCING ML MODELS IN PRODUCTION

- **Data quality**:
    - Since ML models are built on data, they are sensitive to the semantics, amount, and completeness of incoming data.
- **Model decay**:
    - The performance of ML models in production degenerates over time because of changes in the real-life data that has not been seen during the model training.
- **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!
- There is a need to establish effective practices and processes around designing, building, and deploying ML models into production.
    - Welcome to **MLOps**!

4.6 MLOPS DEFINITION

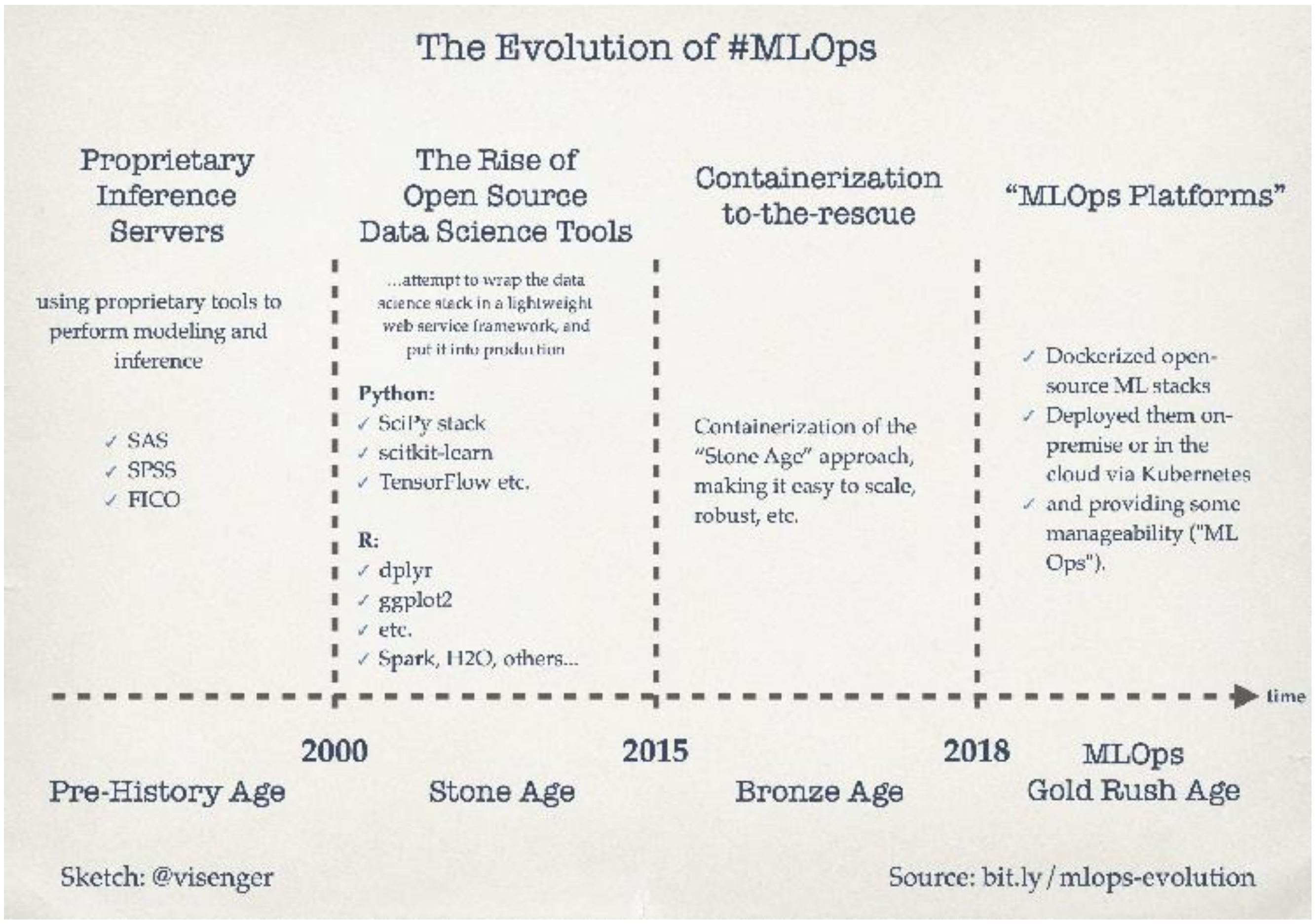
- **Per MLOps SIG**:
    - The term MLOps is defined as "the extension of the DevOps methodology to include Machine Learning and Data Science assets as first-class citizens within the DevOps ecology."
- Alternatively, can use the definition of **Machine Learning Engineering (MLE)**:
    - Involves the use of scientific principles, tools, and techniques of machine learning and traditional software engineering to design and build complex computing systems.
- Encompasses all stages from **data collection**, to **model building**, to make the model available for use by the product or the consumers.

4.7 MLOPS CAPABILITIES

Per Continuous Delivery Foundation SIG MLOps:

- **MLOps**:
    - Aims to unify the release cycle for machine learning and software application release.
    - Enables automated testing of machine learning artifacts (e.g., **data validation**, **ML model testing**, and **ML model integration testing**).
    - Enables the application of agile principles to machine learning projects.
    - Enables supporting machine learning models and datasets to build these models as **first-class citizens** within CI/CD systems.
    - Reduces **technical debt** across machine learning models.
- MLOps must be a **language-, framework-, platform-, and infrastructure-agnostic** practice.

4.7 THE EVOLUTION OF THE MLOPS

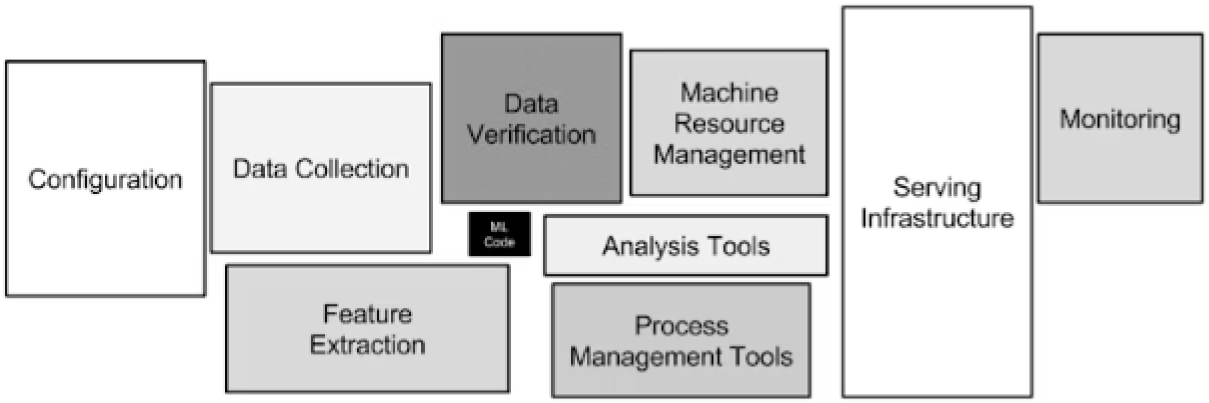

The Evolution of #MLOps
Proprietary Inference Servers
using proprietary tools to perform modeling and inference
✓ SAS
✓ SPSS
✓ FICO
The Rise of Open Source Data Science Tools
...attempt to wrap the data science stack in a lightweight web service framework, and put it into production
Python:
✓ SciPy stack
✓ scitkit-learn
✓ TensorFlow etc.
R:
✓ dplyr
✓ ggplot2
✓ etc.
✓ Spark, H2O, others...
Containerization to-the-rescue
Containerization of the "Stone Age" approach, making it easy to scale, robust, etc.
"MLOps Platforms"
✓ Dockerized open-source ML stacks
✓ Deployed them on-premise or in the cloud via Kubernetes
✓ and providing some manageability ("ML Ops").
time
2000
2015
2018
Pre-History Age
Stone Age
Bronze Age
MLOps Gold Rush Age
Sketch: @visenger
Source: bit.ly/mlops-evolution

5. DEVOPS VS MLOPS

5.1 What's Different?

Software performs actions in response to inputs, and in this ML and mainstream programming are alike.

- **Traditional software** codifies actions as explicit rules:
    - Simplest programming examples tend to be 'hello world' programs.
    - Control structures are added for more complex actions.
    - Inputs are via keyboard, outputs are mostly text.
- **ML** does not codify explicitly:
    - Rules are indirectly set by capturing patterns from data.
    - More suitable for numerical problems (e.g., predicting salary from experience, education, location).



ML is different from traditional programming as it needs to consider:

- The training data and code together drive fitting.
- Trained models vary by ML toolkit and type.
- Retraining may be necessary.
- Data volumes can be large, and training can take a long time.
- The data scientist's working process is exploratory.
- Leads to different workflows for traditional programming and ML development.

5.2 Traditional programming workflow

1. Prepare User Story
2. Write code
3. Submit Merge Request
4. Tests run automatically
5. Review and merge
6. New version builds
7. Built executable deployed to environment
8. Further tests
9. Promote to next environment
10. More tests etc.
11. PROD
12. Monitor - stack traces or error codes

High-level MLOps workflow

- Data inputs and outputs pre-processed.
- Data scientist tries stuff locally with a slice of data.
- Collaboration often in Jupiter notebooks & git.
- Model may be pickled/serialized.
- Integrate into a running app (e.g., add REST API).
- Integration test with app.
- Rollout and monitor performance metrics.
- Driver for a build might be a code change or new data.

Training, testing, serving, and monitoring:

- **Testing**:
    - Not likely to be a simple pass/fail, but quantifiable performance.
- **Learning / Training**:
    - Can differ depending on online or offline learning.
- **Monitoring**:
    - Particularly challenging and may involve business decisions.

The role of **MLOps** is to support the whole flow of training, serving, rollout, and monitoring.

6. PEOPLES OF MLOPS

Role	Responsibilities	MLOps Requirements
Subject matter experts	Provide business questions, goals, or KPIs.	- Understand deployed model performance in business terms. - Mechanism for flagging model results not aligning with expectations.
Data scientists	Build operationalizable models and assess quality.	- Automated model packaging and delivery. - Develop tests for model quality and visibility into model performance.
Data engineers	Optimize data retrieval and use.	- Visibility into performance of deployed models. - Full details of individual data pipelines.
Software engineers	Integrate ML models into applications.	- Versioning, automatic tests, and parallel work on applications.
DevOps engineers	Build operational systems and manage CI/CD pipelines.	- Seamless integration of MLOps into DevOps strategy and deployment pipeline.
Model risk managers/ auditors	Minimize risk and ensure compliance.	- Robust, likely automated, reporting tools on all models and compliance with requirements.
Machine learning architects	Ensure scalable ML model pipelines.	- High-level overview of models and ability to adjust infrastructure.
ML Engineer	Design, build, and productionize ML models, considering responsible AI and collaborating with other roles.	- Proficiency in model architecture, data pipeline interaction, and familiarity with application development, infrastructure management, data engineering, and data governance.
MLOps Engineer	Design and implement cloud solutions, build MLOps pipelines, model review, testing, validation, automation, documentation.	- Ability to design and implement cloud solutions, experience with MLOps Frameworks, programming languages, Linux, and familiarity with ML frameworks.

7. MLOPS FEATURES / KEY COMPONENTS:

MLOps affects many different roles across the organization, in turn, many parts of the **machine learning life cycle**. Introduces the five key components of MLOps:

- **Development**
- **Deployment**
- **Monitoring**
- **Iteration**
- **Governance**

7.1 Model Development

Establishing Business Objectives, Data Sources, and Exploratory Data Analysis
Establishing Business Objectives

The process typically starts with a **business objective**, which can be as simple as reducing **fraudulent transactions** to < 0.1% or having the ability to identify people's faces on their social media photos. **Business objectives** naturally come with **performance targets**, **technical infrastructure requirements**, and **cost constraints**. All of these factors can be captured as **KPIs**, which will ultimately enable the business performance of models in production to be monitored. **ML projects** are generally part of a larger project that in turn impacts technologies, processes, and people means part of establishing objectives also includes **change management**, which may even provide some guidance for how the ML model should be built.

DATA SOURCES AND EXPLORATORY DATA ANALYSIS

With clear **business objectives** defined, the search for suitable input data starts. Finding data sounds simple, but in practice, it can be the most arduous part of the journey. The constraints of **data governance** bring even more questions. **Data** is the essential ingredient to power ML algorithms, always helps to build an understanding of the patterns in data before attempting to train models. **EDA techniques** can help build hypotheses about the data, identify data cleaning requirements, and inform the process of selecting potentially significant features. **EDA** can be carried out visually for intuitive insight and statistically if more rigor is required.

FEATURE ENGINEERING AND SELECTION, TRAINING AND EVALUATION

Feature Engineering and Selection
EDA leads naturally into **feature engineering** and **feature selection**. **Feature engineering** is the process of taking raw data from the selected datasets and transforming it into **“features”** that better represent the underlying problem to be solved. **“Features”** are arrays of numbers of fixed size, as it is the only object that ML algorithms understand. **Feature engineering** includes data cleansing, which can represent the largest part of an ML project in terms of time spent.

Training and Evaluation
The process of **training and optimizing** a new ML model is iterative. Several algorithms may be tested, features can be automatically generated, feature selections may be adapted, and algorithm hyperparameters tuned. In addition to—or in many cases because of—its iterative nature, **training** is also the most intensive step of the ML model life cycle when it comes to computing power. Keeping track of the results of each experiment when iterating becomes complex quickly. An **experiment tracking tool** can greatly simplify the process of remembering the data, the features selection, and model parameters alongside the performance metrics.

7.2 Reproducibility

While many experiments may be short-lived, **significant versions** of a model need to be saved for possible later use.

- The challenge here is **reproducibility**, which is an **important concept** in experimental science in general.
- The aim in ML is to save enough information about the **environment** the model was developed in so that the model can be reproduced with the **same results** from scratch.
- Without reproducibility, **data scientists** have little chance of being able to confidently **iterate** on models. Worse, they are unlikely to be able to hand over the model to **DevOps** to see if what was created in the lab can be faithfully reproduced in production.
- True reproducibility requires **version control** of all the assets and parameters involved, including the **data** used to train and evaluate the model, as well as a record of the **software environment**.

7.3 Productionalization and Deployment

Productionalizing and deploying models is a **key component** of MLOps that presents an entirely different set of **technical challenges** than developing the model.

The domain of the **software engineer** and the **DevOps team** entails organizational challenges in managing the **information exchange** between the **data scientists** and these teams, which must not be underestimated.
Without **effective collaboration** between the teams, **delays** or **failures to deploy** are inevitable!

Model Deployment Types and Contents
It's essential to ask: What exactly is going into production, and what does a model consist of?

There are commonly two types of model deployment:

- **Model-as-a-service**, or live-scoring model: Typically, the model is deployed into a simple framework to provide a **REST API endpoint** that responds to requests in real time.
- **Embedded model**: The model is packaged into an application, which is then published. A common example is an application that provides **batch-scoring** of requests.

The content of to-be-deployed models depends on the technology chosen but typically comprises a set of code (commonly Python, R, or Java) and **data artifacts**. Any of these can have **version dependencies** on runtimes and packages that need to match in the production environment because the use of different versions may cause model predictions to differ.

One approach to reducing dependencies on the production environment is to **export the model** to a portable format such as PMML, PFA, ONNX, or POJO. This aims to improve model **portability** between systems and simplify deployment but comes at a cost: each format supports a limited range of algorithms, and sometimes the portable models behave in subtly different ways than the original.

Model Deployment Requirements
Rapid, automated deployment is always preferred to labor-intensive processes!

For short-lifetime, self-service applications, there often isn't much need to worry about **testing and validation**. It's even possible to handle simple user interfaces with frameworks like Flask when using this lightweight deployment mode.
In customer-facing, mission-critical use cases, a more robust **CI/CD pipeline** is required, which typically involves:

- Ensuring all coding, documentation, and sign-off standards have been met
- Re-creating the model in something approaching the production environment
- Revalidating the model accuracy
- Performing explainability checks
- Ensuring all governance requirements have been met
- Checking the quality of any data artifacts
- Testing resource usage under load
- Embedding into a more complex application, including integration tests

7.4 Different concerns

Once a model is **deployed to production**, it is **crucial** that it **continues to perform well** over time. However, **good performance** can have **different meanings** to different stakeholders, particularly to the **DevOps team**, **data scientists**, and **the business**.

DevOps Concerns:

- The concerns of the DevOps team are familiar, including questions such as:
    - Is the model **efficiently completing tasks**?
- Is it **utilizing memory and processing resources sensibly**?
    - Overall, the **expertise** of DevOps teams in **resource monitoring and management** can be **applied** to ML models effectively.
- **Data Scientist Concerns:**
- Data scientists are interested in **monitoring ML models** for a **unique reason**: they can **degrade over time** due to their dependence on the **training data**. This is an inherent challenge in **machine learning**, requiring **model retraining** when performance becomes unacceptable.

Business Concerns:

- The business takes a **holistic approach** to monitoring, considering questions such as:
    - Is the model **delivering value** to the enterprise?
- Do the **benefits outweigh the costs** of development and deployment?
    - Monitoring **KPIs** identified for the original business objective is crucial. **Automated monitoring** of these metrics is desirable, although it is often complex to implement.

7.5 Iteration and Life Cycle

Iteration

Developing and deploying **improved versions** of a model is **essential** within the MLOps life cycle, posing one of the more **challenging** aspects.

- There are **various reasons** to develop a new model version, including **model performance degradation** due to **model drift**, adjustments to **refined business objectives** and **KPIs**, or simply **innovative enhancements** by data scientists.
- In some fast-paced business environments, **new training data** emerges daily, prompting **automated daily retraining** and redeployment to ensure the model reflects recent experiences.
- Retraining an existing model with the **latest training data** represents the simplest scenario for **iterating** a new model version. Subsequently, comparing **metrics** with the current live model version is crucial, often necessitating evaluation on the same development dataset.
- If significant variations between models are detected, automated redeployment should be **avoided**, and manual intervention sought, highlighting the importance of careful **validation**.

The Feedback Loop

In large enterprises, **DevOps best practices** typically dictate that the **live model scoring environment** and the **model retraining environment** remain distinct, potentially compromising the evaluation of new model versions.

- One approach to mitigating this uncertainty is **shadow testing**, where the new model version operates alongside the existing one. In this setup, all live scoring is conducted by the incumbent model, while the results from the new model are logged for comparison.
- Another strategy involves **A/B testing**, where both models are deployed into the live environment, with input requests split between them. This allows for statistically meaningful comparisons, though careful planning of the test is essential for drawing accurate conclusions.

7.6 Governance

Governance encompasses the **controls** placed on a business to fulfill its responsibilities to stakeholders, including shareholders, employees, and the public, as well as regulatory bodies. It encompasses **financial**, **legal**, and **ethical** obligations, all rooted in the principle of **fairness**.

- **Legal obligations** are straightforward and have long existed, predating the rise of machine learning. Regulations, often targeting specific industries like finance and pharma, impose constraints on businesses.
- Recent regulations, such as the 2016 EU General Data Protection Regulation (**GDPR**) and the 2018 California Consumer Privacy Act (**CCPA**), aim to safeguard personal data and privacy rights. Governments are increasingly focusing on regulating machine learning to mitigate its potential negative impacts.

Data and Process Governance:

- With growing public activism, businesses are embracing the concept of **Responsible AI**, emphasizing ethical, transparent, and accountable AI applications.
- Applying effective governance to **MLOps** is challenging due to the complexity of processes, opacity of technology, and reliance on data.
- Governance initiatives in **MLOps** primarily focus on two categories: **Data governance** ensures appropriate data use and management, while **Process governance** defines well-structured processes to address governance considerations throughout the model life cycle, maintaining accurate records.

8. CD AND AUTOMATION PIPELINES IN ML

8.1 Data Science and ML in Real World

Gaining importance in businesses, becoming core capabilities for solving complex real-world problems, transforming industries, and delivering value in all domains.
Currently, the ingredients for applying effective ML are:

- Large datasets
- Inexpensive on-demand compute resources
- Specialized accelerators for ML on various cloud platforms
- Rapid advances in different ML research fields (such as computer vision, natural language understanding, and recommendations AI systems)

Many businesses are investing in their data science teams and ML capabilities to develop predictive models that can deliver business value to their users.

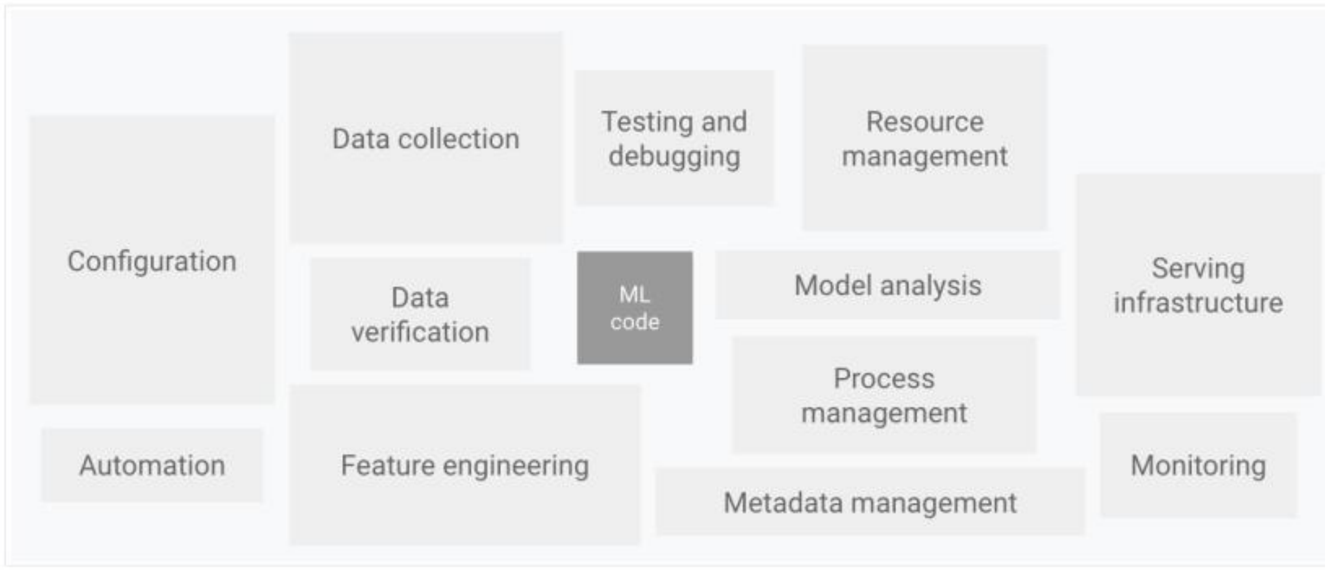


Figure 1. Elements for ML systems. Adapted from Hidden Technical Debt in Machine Learning Systems (https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf).

Real Challenge

Continuously deploy and operate. Data scientists can implement and train an ML model with predictive performance on an offline holdout dataset. The real challenge isn't building an ML model; the challenge is building an integrated ML system and to continuously operate it in production.

MLOps is an ML engineering culture and practice that aims at unifying ML system development (**Dev**) and ML system operation (**Ops**).
Practicing MLOps means advocating for automation and monitoring at all steps of ML system construction, including integration, testing, releasing, deployment, and infrastructure management.

Elements of ML System
There can be many pitfalls in operating ML-based systems in production. Some are summarized in *Machine Learning: The high-interest credit card of technical debt*.
Only a small fraction of a real-world ML system is composed of the ML code. Required surrounding elements are vast and complex.
The rest of the system is composed of configuration, automation, data collection, data verification, testing and debugging, resource management, model analysis, process and metadata management, serving infrastructure, and monitoring.
To develop and operate complex systems like these, we need to apply DevOps principles to ML systems (MLOps).

8.2 DevOps versus MLOps

DevOps - A popular practice in developing and operating large-scale software systems. Provides benefits such as shortening the development cycles, increasing deployment velocity, and ensures dependable releases.
To achieve these benefits, introduces two concepts in the software system development:

- **Continuous Integration (CI)**
  - Practice of merging all developers' working copies to a shared mainline several times a day.
  - Ensures that no errors arise without developers noticing them and correcting them immediately.
- **Continuous Delivery (CD)**
  - Software engineering approach in which teams produce software in short cycles, ensuring reliable releases at any time.
  - Aims at building, testing, and releasing software with greater speed and frequency, reducing cost, time, and risk.
  - Requires a straightforward and repeatable deployment process.

MLOps systems differ from other software systems in various aspects:

Team Skills:

- ML projects involve data scientists or ML researchers who focus on exploratory data analysis, model development, and experimentation.
- They may lack experience as software engineers capable of building production-class services.

Development:

- ML is experimental, requiring experimentation with different features, algorithms, and configurations while maintaining reproducibility and code reusability.

Testing:

- Involves more than typical unit and integration tests, including data validation and model quality evaluation.

Deployment:

- Requires deploying a multi-step pipeline to automate model retraining and deployment, adding complexity compared to deploying offline-trained ML models.

Production:

- ML models can experience reduced performance due to evolving data profiles, requiring monitoring and automatic retraining.

ML systems differ from other software systems in the process:

- While similar practices like continuous integration and delivery apply, there are notable differences.
- CI involves testing and validating not only code and components but also data and models.
- CD is about deploying a system (ML training pipeline) that automatically deploys another service (model prediction service).
- Continuous Training (CT) is unique to ML systems, focusing on automatically retraining and serving models.

Data Science Steps for ML

Steps can be completed manually or by an automatic pipeline:

- **Data Extraction:** Select and integrate relevant data from various sources.
- **Data Analysis:** Perform exploratory data analysis.
- **Data Preparation:** Clean, transform, split, and engineer features.
- **Model Training:** Implement algorithms with prepared data to train ML models.
- **Model Evaluation:** Assess model quality on a holdout test set.
- **Model Validation:** Confirm if the model is suitable for deployment.
- **Model Serving:** Deploy to serve predictions in the target environment.
- **Model Monitoring:** Monitor predictive performance for potential iterations in the ML process.

8.3 Maturity of the ML Process

MLOps Levels
The level of automation defines the maturity of the ML process and reflects the velocity of training new models given new data or implementations.
Three levels:

- **MLOps Level 0:** Manual process.
- **MLOps Level 1:** Automation of ML pipeline.
- **MLOps Level 2:** Automation of CI/CD pipeline.

8.4 MLOps Level 0: Manual Process

Data scientists and ML researchers build state-of-the-art models, but their process for building and deploying ML models is entirely manual.

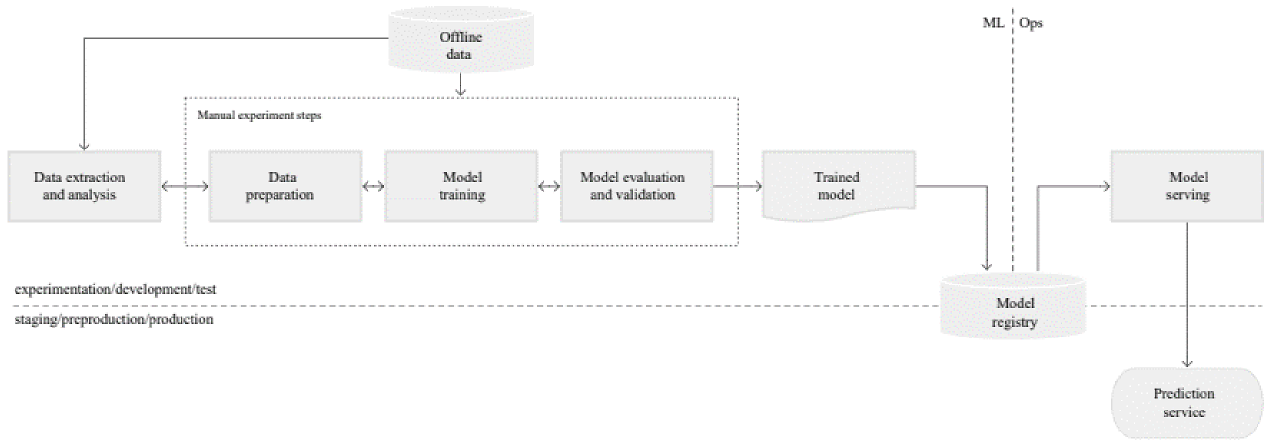


Figure 2. Manual ML steps to serve the model as a prediction service.

Characteristics:
- **Manual Process:** Script-driven and interactive.
- Every step is manual, including data analysis, preparation, training, and validation.
- Driven by experimental code in notebooks until a workable model is produced.

Disconnection Between ML and Operations:

- Separation between data scientists and engineers deploying the model.
- Data scientists hand over trained models to engineering for deployment.
- Can lead to training-serving skew.

Infrequent Release Iterations:

- Assumes few model changes managed by the data science team.

No CI:

- Testing usually part of notebooks or scripts.
- Artifacts like trained models are source-controlled.

No CD:

- CD not considered due to infrequent model version deployments.

Deployment Focus:

- Concerned with deploying trained models as prediction services.

Lack of Active Performance Monitoring:

- Doesn't track or log model predictions for performance degradation detection.

Common in Many Businesses:

- Manual, data-scientist-driven process may suffice when models are rarely changed.
- Models often break in real-world deployment.

To Address Challenges:

- Actively monitor model quality in production.
- Frequent retraining with recent data to capture evolving patterns.
- Continuously experiment with new implementations.
- MLOps practices for CI/CD and CT are helpful.

8.5 MLOps Level 1: ML Pipeline Automation

The aim is to perform continuous training of the model by automating the ML pipeline.

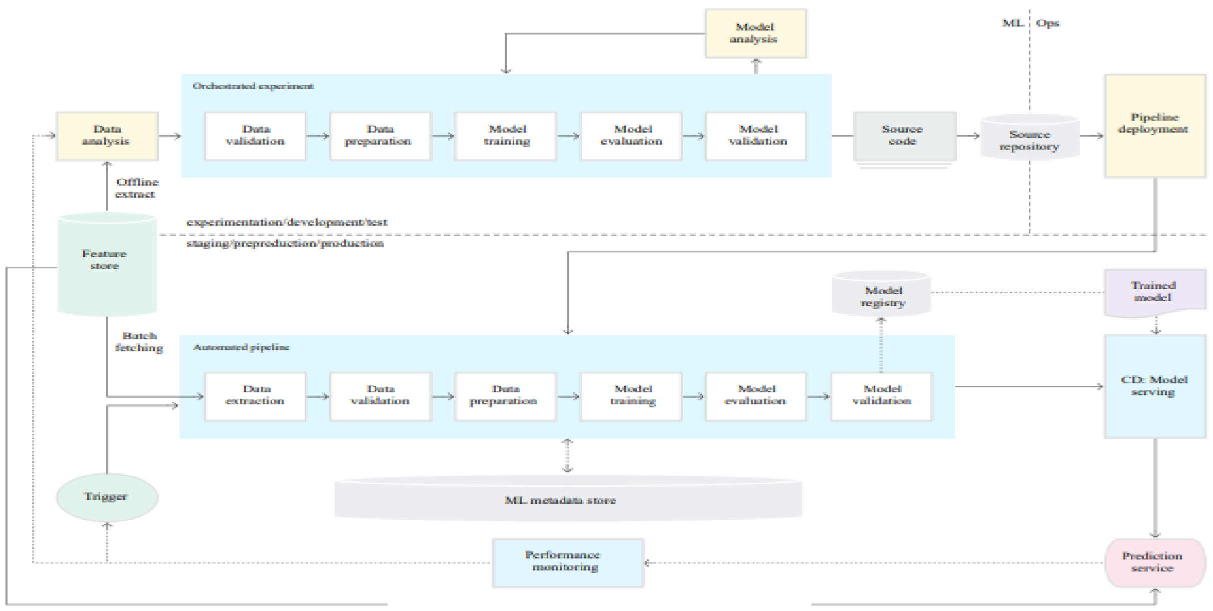


Figure 3. ML pipeline automation for CT.

Characteristics:

- **Rapid Experimentation:**
  - Orchestrates steps of the ML experiment.
  - Automated transition between steps allows for rapid iteration and better readiness for production.
- **Continuous Training (CT) in Production:**
  - Model is automatically trained using fresh data based on live pipeline triggers.
- **Experimental-Operational Symmetry:**
  - Pipeline implementation used in development mirrors that in preproduction and production.
  - Key aspect of MLOps practice for unifying DevOps.
- **Continuous Delivery of Models:**
  - ML pipeline continuously delivers prediction services from new models trained on fresh data.
  - Automated deployment serves the trained and validated model for online predictions.
- **Modularized Code for Components and Pipelines:**
  - Components must be reusable, composable, and potentially shareable.
  - Source code modularization is essential.

- Components ideally containerized to decouple execution environment and ensure reproducibility.
- **Pipeline Deployment:**
    - In Level 0, only the trained model is deployed to production.
    - In Level 1, the entire training pipeline is deployed, automatically and recurrently serving the trained model as the prediction service.

DATA VALIDATION

When an ML pipeline is deployed to production, triggers execute the pipeline, expecting new, live data to generate a new model version trained on this fresh data. Automated **data validation** and **model validation** steps become crucial in the production pipeline.

Data Validation

Before model training, data validation determines whether to retrain the model or halt the pipeline's execution. The pipeline automatically decides based on identified issues such as:

- **Data Schema Skews:** Anomalies in the input data, indicating data that doesn't conform to the expected schema. Pipeline should halt for investigation if unexpected features, missing features, or unexpected feature values are detected.
- **Data Values Skews:** Significant changes in statistical properties, indicating shifting data patterns, triggering the need for model retraining to adapt to these changes.

Model Validation

After successfully training the model with new data, model validation occurs to assess and validate the model before promoting it to production. This involves:

- Generating evaluation metric values using the trained model on a test dataset to evaluate its predictive quality.
- Comparing evaluation metric values of the newly trained model with the current model to ensure consistent performance across various data segments.
- Testing the model for deployment, including infrastructure compatibility and adherence to the prediction service API.

In addition to offline model validation, a newly deployed model undergoes online model validation, typically through a canary deployment or A/B testing setup, before handling online traffic.

FEATURE STORE

A **feature store** is an optional but highly beneficial component that serves as a centralized repository for standardizing the definition, storage, and access of features for both training and serving in machine learning pipelines.

Key Features:

- **Centralized Repository:** Acts as a single source of truth for feature definitions, ensuring consistency across the ML workflow.
- **API Support:** Provides an API for both high-throughput batch serving and low-latency real-time serving of feature values to accommodate various workload requirements.

Benefits for Data Scientists:

- **Feature Reuse:** Facilitates the discovery and reuse of existing feature sets, reducing duplication of effort.
- **Consistency:** Prevents the creation of similar features with different definitions by maintaining features and their metadata.
- **Up-to-Date Feature Values:** Ensures access to the most current feature values from the feature store.
- **Prevents Skew:** Mitigates training-serving skew by using consistent feature sets across experimentation, continuous training, and online serving.

Workflow Integration:

- **Experimentation:** Data scientists can extract offline feature sets from the feature store for experimentation.
- **Continuous Training:** Automated ML pipelines fetch up-to-date feature values from the feature store for training tasks.
- **Online Prediction:** Prediction services retrieve relevant feature values in batches from the feature store for online predictions, such as customer demographics, product features, and session aggregations.

Metadata Management

Metadata management involves recording essential information about each execution of the ML pipeline. This metadata serves various purposes, including maintaining data and artifacts lineage, ensuring reproducibility, facilitating comparisons, and aiding in debugging errors and anomalies.

Recorded Metadata:

- **Pipeline and Component Versions:** Versions of the pipeline and its components executed during each run.
- **Execution Details:** Start and end dates, duration of each pipeline step, and the executor of the pipeline.
- **Parameter Arguments:** Parameters passed to the pipeline during execution.
- **Artifact Pointers:** Pointers to artifacts generated by each pipeline step, including locations of prepared data, validation anomalies, computed statistics, and extracted vocabulary from categorical features.
- **Previous Model Pointer:** Pointer to the previous trained model, useful for rolling back to a previous model version or producing evaluation metrics for comparison.
- **Model Evaluation Metrics:** Metrics produced during the model evaluation step for both training and testing sets, facilitating performance comparison between newly trained and previous models.

ML Pipeline Triggers

Automation Possibilities:

ML production pipelines can be automated to retrain models with new data based on various triggers depending on the use case:

1. On Demand:

- **Definition:** Manual execution of the pipeline as needed.

2. On a Schedule:

- **Description:** Systematic availability of new, labeled data for the ML system at regular intervals (e.g., daily, weekly, or monthly).
- **Retraining Frequency:** Depends on how frequently data patterns change and the cost associated with retraining models.

3. On Availability of New Training Data:

- **Scenario:** New data becomes available on an ad-hoc basis when collected and made accessible in the source databases.

4. On Model Performance Degradation:

- **Situation:** Retraining occurs when noticeable performance degradation in the model is observed.

5. On Significant Changes in Data Distributions (Concept Drift):

- **Observation:** Noticeable changes in data distributions of features used for prediction.
- **Implication:** Indicates that the model has become stale and needs retraining with fresh data.

Challenges:

- Suitable for deploying new models based on new data rather than new ML ideas.
- Limited deployment of new pipeline implementations.
- Management of only a few pipelines.
- Manual testing of pipelines and components.

When to Use:

- When experimenting with new ML ideas and rapidly deploying new implementations of ML components.
- Management of numerous ML pipelines in production.
- Requirement for a CI/CD setup to automate the build, test, and deployment of ML pipelines.

MLOps Level 2: CI/CD Pipeline Automation

MLOps Level 2 involves the implementation of the ML pipeline using Continuous Integration/Continuous Delivery (CI/CD) practices.

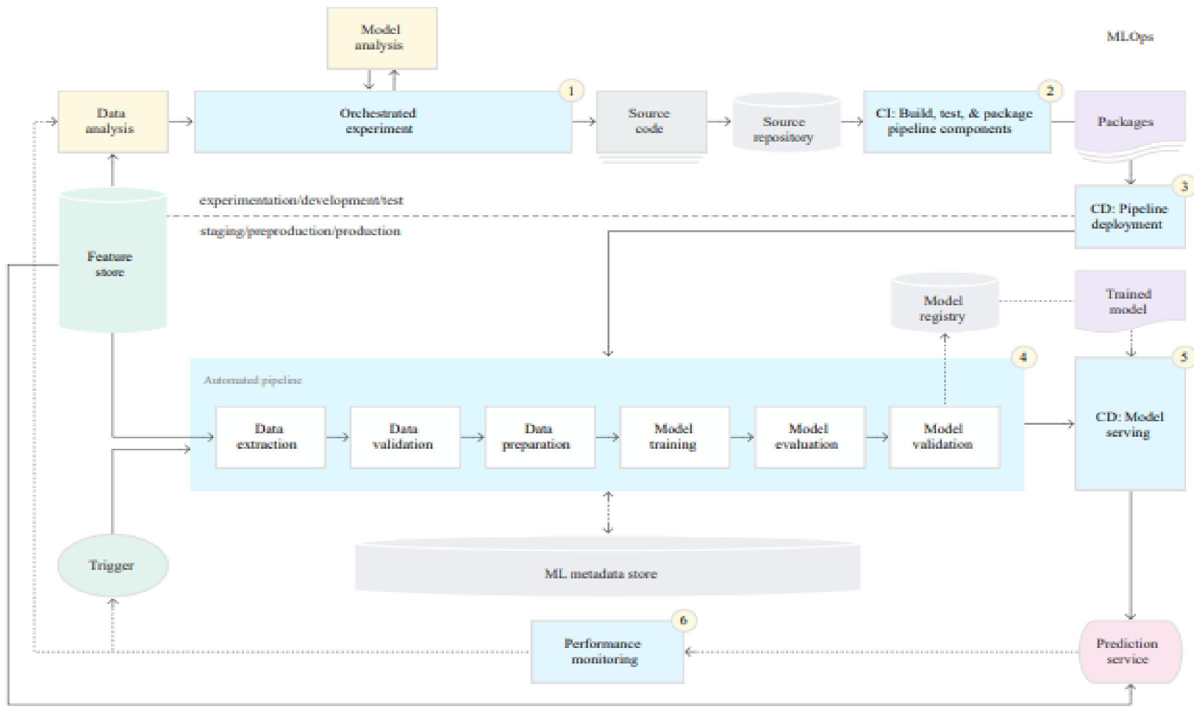


Figure 4. CI/CD and automated ML pipeline.

STAGES OF THE ML CI/CD AUTOMATION PIPELINE:

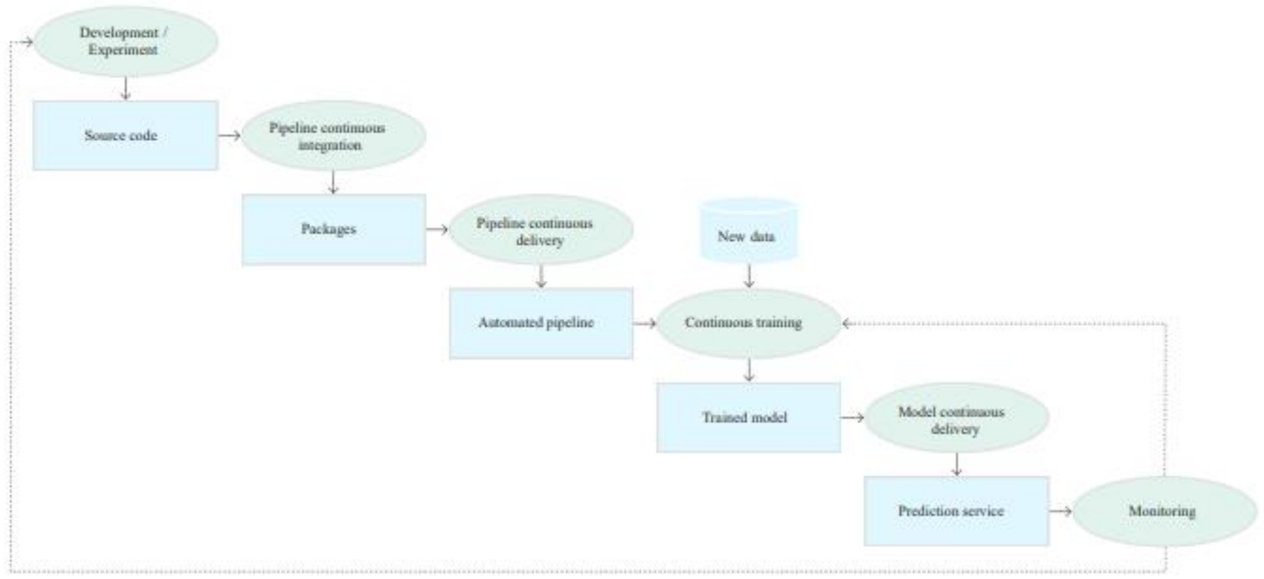


Figure 5. Stages of the CI/CD automated ML pipeline.

Development and Experimentation:

- **Objective:** Iteratively try out new ML algorithms and modeling techniques.
- **Output:** Source code of the ML pipeline steps pushed to a source repository.

Pipeline Continuous Integration:

- **Process:** Build source code and run various tests.
- **Output:** Pipeline components (packages, executables, artifacts) for deployment.

Pipeline Continuous Delivery:

- **Action:** Deploy artifacts produced by the CI stage to the target environment.
- **Output:** Deployed pipeline with the new model implementation.

Automated Triggering:

- **Initiation:** Automatically execute the pipeline in production based on a schedule or trigger.
- **Output:** Trained model pushed to the model registry.

Model Continuous Delivery:

- **Function:** Serve the trained model as a prediction service for generating predictions.
- **Output:** Deployed model prediction service.

Monitoring:

- **Task:** Collect statistics on model performance based on live data.
- **Output:** Trigger for pipeline execution or initiation of a new experiment cycle.

Manual Processes:

- Data analysis and model analysis steps remain manual processes for data scientists before initiating a new experiment iteration.

Continuous Integration:

- Triggers when new code is committed or pushed to the source code repository.
- Includes unit testing of feature engineering logic, model methods, and pipeline components.
- Ensures integration between pipeline components is functioning correctly.

Continuous Delivery:

- Involves continuous delivery of new pipeline implementations to the target environment.
- Verification of model compatibility with the target infrastructure before deployment.
- Testing of prediction service performance, data validation, and predictive performance targets.
- Deployment to different environments: test environment (triggered by code push to development branch), pre-production environment (triggered by merging code to main branch after reviewer approval), and manual deployment to production environment after successful runs on pre-production.

Key Considerations:

- Ensuring compatibility with target infrastructure.
- Validating prediction service performance and data integrity.
- Automated deployment to test and pre-production environments, with manual deployment to production after successful testing.

Summary

Implementing **Machine Learning** (**ML**) in a production environment extends beyond merely deploying a model as an API for **prediction**. It involves deploying a comprehensive **ML pipeline** capable of automating model **retraining** and **deployment** processes.

Establishing a **Continuous Integration/Continuous Delivery (CI/CD)** system enables the automatic testing and deployment of new **pipeline implementations**. This approach facilitates adaptation to rapid changes in **data** and the **business environment**.

Transitioning all processes from one level to another immediately is unnecessary. Instead, organizations can gradually implement these practices to enhance the automation of **ML system development** and **production**.

9. ALL THE OPS

MLOps, **ModelOps**, **DataOps**, and **AIOps** are rapidly gaining importance as organizations seek to harness the power of **artificial intelligence** (**AI**), **machine learning** (**ML**), and **big data**. Each approach enables organizations to construct robust systems capable of efficiently processing large volumes of data.

AllOps

- **MLOps**: Focuses on continuous delivery cycles for **ML models** through automated pipelines.
- **ModelOps**: Manages model development from conception to deployment.
- **DataOps**: Provides tools for efficient data processing pipelines.
- **AIOps**: An **AI-driven operations platform** automating IT processes like incident resolution.

DEVOPS	**DEVOPS** INTEGRATES **SOFTWARE DEVELOPMENT** AND **IT OPERATIONS** TO SHORTEN DEVELOPMENT LIFECYCLES AND ENSURE CONTINUOUS DELIVERY WITH HIGH SOFTWARE QUALITY.
DEVSECOPS	EMPHASIZES INTEGRATING **SECURITY** INTO **DEVOPS** INITIATIVES DUE TO THE SHARED RESPONSIBILITY FOR SECURITY THROUGHOUT THE DEVELOPMENT PROCESS.
DATAOPS	**DATAOPS** APPLIES **AGILE DEVELOPMENT**, **DEVOPS**, AND **LEAN MANUFACTURING** PRINCIPLES TO **DATA ANALYTICS** AND **OPERATIONS**, FACILITATING COLLABORATION BETWEEN **DATA ENGINEERS** AND **DEVOPS ENGINEERS** TO ENHANCE ANALYTICS INITIATIVES.
MLOPS	STANDARDIZES AND STREAMLINES **ML MODEL** LIFECYCLES IN PRODUCTION, MANAGING THE MOVEMENT OF MODELS, DATA, AND OUTCOMES AMONG SYSTEMS.
MODELOPS	INCORPORATES **MLOPS** TO MANAGE **ML MODELS** ENTERPRISE-WIDE, OPERATIONALIZING ALL **AI MODELS** BEYOND JUST **ML MODELS**, REQUIRING SIMILAR SKILLS AS **MLOPS** WITH ADDITIONAL **AI-SPECIFIC EXPERTISE**.
AIOPS	UTILIZES **MACHINE LEARNING** AND **AI TECHNOLOGIES** TO AUTOMATE **IT OPERATIONS** PROCESSES WITHIN ORGANIZATIONS, DIFFERENTIATING FROM **MLOPS** BY FOCUSING ON A BROADER SCOPE OF AUTOMATION WITHIN **IT OPERATIONS DEPARTMENTS**.

10. MLOPS LIFE-CYCLE, PROCESS, AND CAPABILITIES

Google Cloud's AI Adoption Framework

- Google Cloud's AI Adoption Framework offers guidance for **technology leaders** aiming to establish an effective **artificial intelligence (AI)** capability to drive business transformation.
- It addresses AI challenges concerning **people**, **data**, **technology**, and **process**, structured across six themes: **learn**, **lead**, **access**, **secure**, **scale**, and **automate**.

Scale concerns utilizing cloud-managed **ML services** capable of scaling with **large amounts of data** and numerous data processing and ML jobs, reducing operational overhead.

Automate focuses on the ability to deploy, execute, and operate technology for **data processing** and **ML pipelines** in production efficiently, frequently, and reliably.

Organizations can leverage this framework to pinpoint gaps in constructing an integrated **ML platform**, emphasizing the **scale** and **automate** themes from Google's AI Adoption Framework.
The decision to adopt each of these processes and capabilities, and to what extent, hinges on your **business context**. For instance, determining the **business value** the framework offers compared to the **cost** of acquiring or building capabilities, such as the investment in engineering hours, is crucial.

Models don't make it into **production**. If they do, they break because they fail to adapt to changes in the environment.

- Only one in two organizations has moved beyond **pilots** and proofs of concept.
  - Moreover, **72%** of organizations starting **AI pilots** before 2019 have not deployed even a single application in production.
  - Algorithmia's survey found that **55%** of companies have not deployed an **ML model**.
- Issues include:
  - High manual and one-off work.
  - Lack of reusable or reproducible components.
  - Difficulties in handoffs between **data scientists** and **IT**.
- Deloitte identified **talent** and **integration** issues as factors that can stall or derail **AI initiatives**.

- Challenges in **deployment**, **scaling**, and **versioning** efforts still hinder teams from extracting value from ML investments.
- Capgemini Research noted the top three challenges faced by organizations in achieving deployments at scale: lack of mid- to senior-level talent, lack of change-management processes, and lack of strong governance models.

Complexities of ML Application

- Preparing and maintaining high-quality **data** for **training ML models**.
- Tracking models in production to detect performance degradation.
- Ongoing experimentation of new data sources, **ML algorithms**, and **hyperparameters**.
- Continuously retraining models on fresh data.
- Avoiding **training-serving skews** and handling concerns about model fairness and adversarial attacks.

ML Engineering

- ML systems cannot be built **ad hoc** or isolated from other **IT initiatives** like **DataOps** and **DevOps**.
- Requires adopting and applying sound **software engineering** practices tailored to ML's unique operationalization challenges.
- Organizations need an automated and streamlined ML process to successfully deploy models in production, manage risk, and ensure alignment with business goals.
- **ML engineering** is essential for building **ML-enabled systems**, focusing on the development and operationalization of production-grade ML systems.
- Provides a superset of **software engineering** to address the unique complexities of practical ML applications.

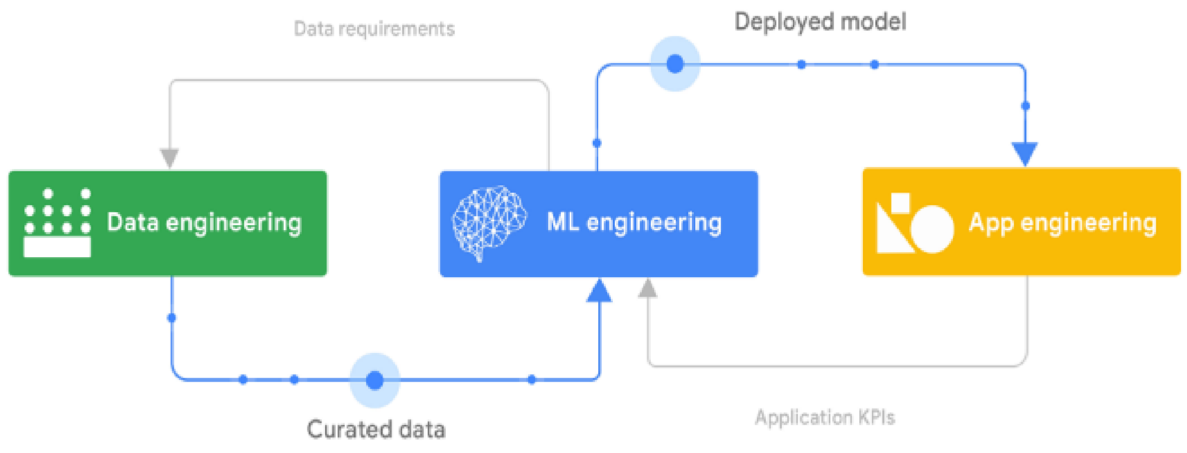
MLOps

- **MLOps** is a **methodology** for **ML engineering** that unifies **ML system development** (the **ML element**) with **ML system operations** (the **Ops element**).
- It advocates formalizing and (when beneficial) automating critical steps of ML system construction.
- **MLOps** provides a set of **standardized processes** and **technology capabilities** for building, deploying, and operationalizing ML systems **rapidly** and **reliably**.
- **MLOps** supports ML development and deployment similarly to how **DevOps** and **DataOps** support **application engineering** and **data engineering** (analytics)!

  - When deploying an ML model, one must also consider changes in the data, model, user behavior, etc.
- **MLOps practices** can result in several benefits over systems that do not follow **MLOps practices**:
  - Shorter **development cycles**, leading to shorter **time to market**.
  - Improved **collaboration** between teams.
  - Increased **reliability**, **performance**, **scalability**, and **security** of ML systems.
  - Streamlined **operational** and **governance processes**.
  - Enhanced **return on investment** of ML projects.

Building an ML-Enabled System

Multifaceted undertaking that combines **data engineering**, **ML engineering**, and **application engineering** tasks.



The relationship of data engineering, ML engineering, and app engineering

Data Engineering involves:

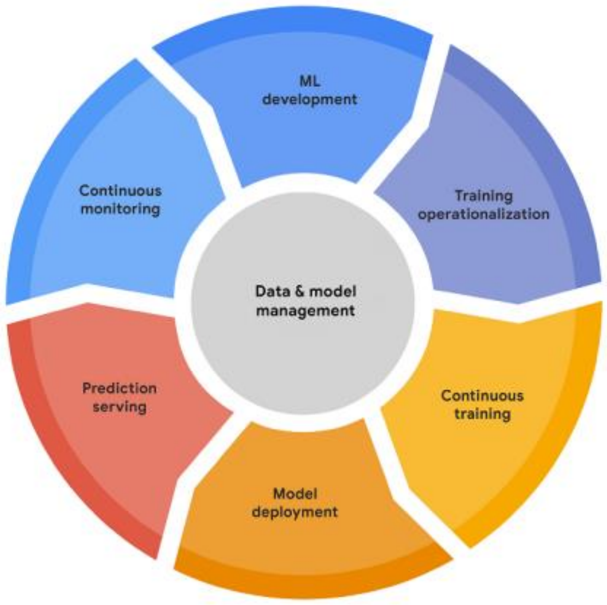
- Ingesting, integrating, curating, and refining data to facilitate a broad spectrum of operational tasks, **data analytics**, and **ML tasks**.
- Robust data engineering processes and technologies are essential for downstream **business intelligence**, **advanced analytics**, or **ML projects**.

ML models are built and deployed in production using curated data created by the **data engineering team**:

- They are components of and support a large range of **application systems**.
- Integrating an ML model into an application involves ensuring effective usage, monitoring performance, and tracking relevant **business KPIs** to understand its impact and adapt accordingly.

The processes can consist of:

- **ML development**:
    - Experimenting and developing a robust and reproducible model training procedure (**training pipeline code**).
- **Training operationalization**:
    - Automating the process of packaging, testing, and deploying repeatable and reliable training pipelines.
- **Continuous training**:
    - Repeatedly executing the training pipeline in response to new data or code changes, potentially with new training settings.

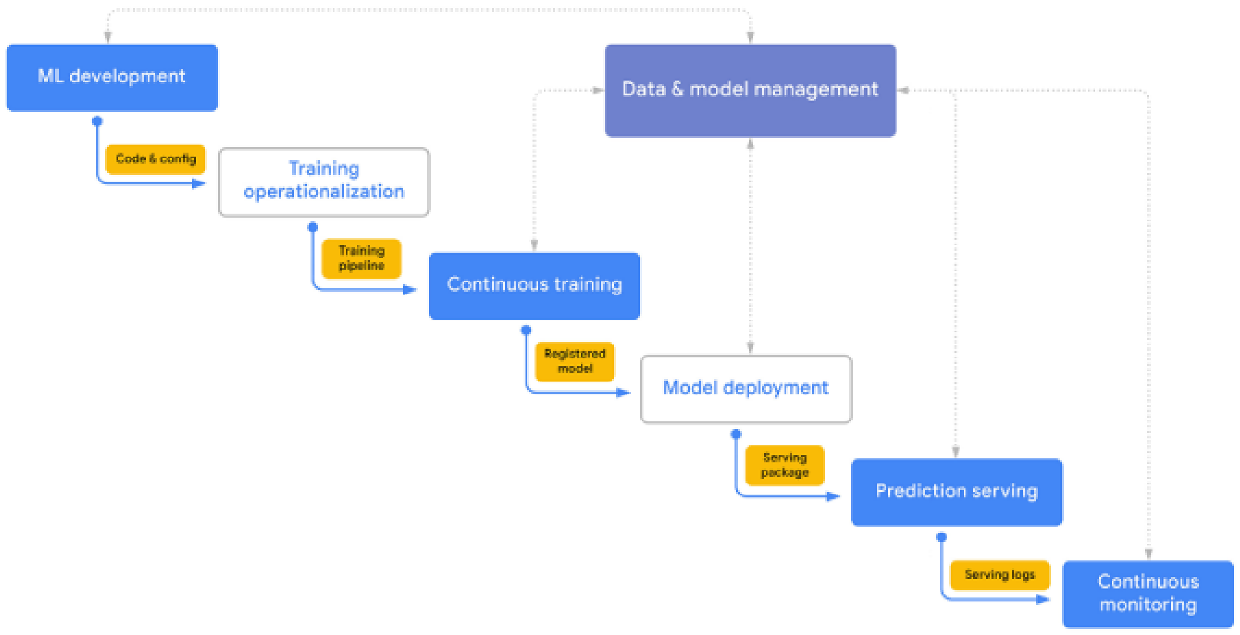


The MLOps lifecycle

- **Model deployment**:
    - Packaging, testing, and deploying a model to a serving environment for online experimentation and production serving.
- **Prediction serving**:
    - Serving the model deployed in production for inference.
- **Continuous monitoring**:
    - Monitoring the effectiveness and efficiency of a deployed model.
- **Data and model management**:
    - Central, cross-cutting function governing ML artifacts to support auditability, traceability, and compliance.
    - Promoting shareability, reusability, and discoverability of ML assets.

MLOps: An End-to-End Workflow

A simplified but canonical flow for how the **MLOps processes** interact with each other, not a waterfall workflow that has to sequentially pass through all the processes.



The MLOps process

1. The core activity during **ML development** phase is **experimentation**:
    - Data scientists and ML researchers prototype model architectures and training routines.
    - They create labeled datasets and utilize features and other reusable ML artifacts.
    - The primary output is a formalized **training procedure**, including data preprocessing, model architecture, and model training settings.

2. If the ML system requires **continuous training**:
    - The training procedure is operationalized as a **training pipeline**.
    - This requires a **CI/CD routine** to build, test, and deploy the pipeline to the target execution environment.
3. The **continuous training pipeline** is executed repeatedly based on retraining triggers and produces a model as output:

    - The model is retrained as new data becomes available or if model performance decay is detected.
    - Other training artifacts and metadata produced by a training pipeline are also tracked.
    - If the pipeline produces a successful model candidate, that candidate is then tracked by the **model management** process as a registered model.

4. The registered model is annotated, reviewed, and approved for release and is then **deployed to a production environment**:
    - This might be relatively opaque if using a no-code solution.
    - It can involve building a custom CI/CD pipeline for **progressive delivery**.

5. The deployed model serves predictions using the **deployment pattern** specified (online, batch, or streaming predictions):
    - The serving runtime can generate **model explanations** and capture serving logs to be used by the **continuous monitoring process**.

6. The **continuous monitoring process** monitors the model for predictive effectiveness and service:
    - The primary concern is detecting model decay, such as data and concept drift.
    - It can also monitor efficiency metrics like latency, throughput, hardware resource utilization, and execution errors.

MLOps Capabilities

A **core set** of **technical capabilities** that are generally required for **MLOps**.

- To effectively implement the **key MLOps processes**:
    - Organizations need to establish a set of **core technical capabilities**.
    - These can be provided by a single **integrated ML platform** or by combining **vendor tools** tailored to specific tasks.
- Processes are typically deployed in stages rather than all at once:
    - For instance, organizations may start with processes for **ML development**, **model deployment**, and **prediction serving**.
    - Continuous training and monitoring might not be necessary initially, especially for organizations piloting a small number of ML systems.

Foundational and ML Specific

- **Foundational capabilities**:
    - Essential to support any IT workload, including reliable, scalable, and secure compute infrastructure.
    - Organizations with existing investments in these capabilities can benefit by leveraging them for ML workflows.
    - May span multiple clouds or operate partially on-premises, ideally with advanced capabilities like specialized ML accelerators.
- **Standardized configuration management** and **CI/CD capabilities**:
    - Necessary for building, testing, releasing, and operating software systems rapidly and reliably, including ML systems.
- **MLOps capabilities**:
    - Encompass **experimentation**, **data processing**, **model training**, **model evaluation**, **model serving**, **online experimentation**, **model monitoring**, **ML pipeline**, and **model registry**.
- **Cross-cutting capabilities**:
    - Enable integration and interaction, such as an **ML metadata** and **artifact repository** and an **ML dataset** and **feature repository**.

MLOps Capability **- Data Processing**

- Allows to **prepare** and **transform** large amounts of data for **ML** at **scale**:
    - In **ML development**, continuous **training pipelines**, and **prediction serving**.
- **Key functionalities** include:
    - **Support interactive execution** for quick experimentation and **long-running jobs** in production.
    - **Data connectors** to a wide range of data sources and services, **data encoders** and **decoders** for various data structures and formats.
    - **Rich** and **efficient data transformations** and **ML feature engineering** for structured and unstructured data.
    - Support for **scalable batch** and **stream data processing** for ML training and serving workloads.

MLOps Capability **- Model Training**

- Efficiently and cost-effectively runs powerful algorithms for training **ML models**.
- Model training should **scale** with the size of both models and datasets.
- **Key functionalities**:
    - Support for common **ML frameworks** and custom runtime environments.
    - **Large-scale distributed training** with different strategies for multiple GPUs and workers.
    - On-demand use of **ML accelerators**.
    - Efficient **hyperparameter tuning** and **target optimization** at scale.
    - Ideally, provides built-in **automated ML** (**AutoML**) functionality, including automated feature selection, engineering, and model architecture search.

MLOps Capability **- Model Evaluation**

- Assesses the effectiveness of a model:
    - Interactively during experimentation and automatically in production.
- **Key functionalities**:
    - Batch scoring of models on evaluation datasets at scale.
    - Computation of pre-defined or custom evaluation metrics for the model on different data slices.
    - Tracking of trained-model predictive performance across different continuous-training executions.
    - Visualization and comparison of performances of different models.
    - Tools for **what-if analysis** and identifying **bias** and **fairness** issues.
    - Enables model behavior interpretation using various **explainable AI** techniques.

MLOps Capability **- Model Serving**

- Allows you to **deploy** and **serve** models in **production environments**.
- **Key functionalities** include:
    - Support for **low-latency**, near-real-time (**online**) prediction and high-throughput **batch** (**offline**) prediction.
    - Built-in support for common **ML serving frameworks** and custom runtime environments.
    - Enable **composite prediction routines** for hierarchical or simultaneous model invocations.
    - Efficient use of **ML inference accelerators** with **autoscaling** to match spiky workloads and balance cost with latency.
    - Support **logging** of prediction serving requests and responses for analysis.

MLOps Capability **- Online Experimentation**

- Helps you understand how newly trained models perform in **production settings** compared to current models before releasing the new model.
- For example, using a small subset of the serving population:
    - Utilize online experimentation to understand the impact of a new recommendation system on click-throughs and conversion rates.
    - Integrate results of online experimentation with the **model registry capability** to facilitate decisions about releasing the model to production.
- Enhances reliability of ML releases by deciding to discard ill-performing models and promote well-performing ones.
- **Key functionalities** include:
    - Support for **canary** and **shadow deployments**.
    - Traffic splitting and **A/B tests**.
    - **Multi-armed bandit (MAB) tests**.

MLOps Capability **- Model Monitoring**

- Lets you track the **efficiency** and **effectiveness** of deployed models in production to ensure **predictive quality** and **business continuity**.
- Informs if models are **stale** and need investigation and updates.
- **Key functionalities** include:
    - Measurement of **model efficiency metrics** like latency and serving-resource utilization.
    - Detection of **data skews**, including schema anomalies, data, concept shifts, and drifts.
    - Integration of monitoring with the **model evaluation capability** for continuous assessment of the deployed model's effectiveness performance when ground truth labels are available.

MLOps Capability **- Online Experimentation**

- Enables you to understand how **newly trained models** perform in **production settings** compared to the current models before releasing the new model.
- For example, using a small subset of the serving population:
    - Utilize **online experimentation** to understand the impact of a new recommendation system on click-throughs and conversation rates.
    - Integrate results with the **model registry capability** to facilitate the decision about releasing the model to production.
- Enhances the reliability of ML releases by aiding in deciding to discard ill-performing models and promote well-performing ones.
- **Key functionalities** include:

    - Support for **canary** and **shadow deployments**.
    - Traffic splitting and **A/B tests**.
    - Support for **multi-armed bandit (MAB) tests**.

MLOps Capability **- ML Pipelines**

- Allows you to instrument, orchestrate, and automate complex ML training and prediction pipelines in production.
- **ML workflows** coordinate different components, with each performing a specific task in the pipeline.
- **Key functionalities** include:
    - Trigger pipelines on demand, on a schedule, or in response to specified events.
    - Enable local interactive execution for debugging during ML development.
    - Integration with the **ML metadata tracking capability** to capture pipeline execution parameters and produce artifacts.
    - Provide a set of built-in components for common ML tasks and allow custom components.
    - Run on different environments, including local machines and scalable cloud platforms.
    - Optionally, provide GUI-based tools for designing and building pipelines.

MLOps Capability **- Model Registry**

- Governs the lifecycle of ML models in a central repository, ensuring the quality of production models and enabling model discovery.
- **Key functionalities** include:
    - Registering, organizing, tracking, and versioning trained and deployed ML models.
    - Storing model metadata and runtime dependencies for deployability.
    - Maintaining model documentation and reporting, such as using **model cards**.
    - Integration with the **model evaluation and deployment capability** and tracking online and offline evaluation metrics.
    - Governing the model launching process: review, approval, release, and rollback.

MLOps Capability **- Dataset and Feature Repository**

- Lets you **unify** the definition and the storage of the **ML data assets**.
- Having a **central repository** of **fresh**, **high-quality data assets** enables **shareability**, **discoverability**, and **reusability**.
    - Provides **data consistency** for **training** and **inference**.
  - Saves time on **data preparation** and **feature engineering**, which typically take up a significant amount of time.
- **Key functionalities** include:
  - Enable **shareability**, **discoverability**, **reusability**, and **versioning** of data assets.
  - Allow **real-time ingestion** and **low-latency serving** for **event streaming** and **online prediction** workloads.
  - Allow **high-throughput batch ingestion** and serving for **extract, transform, load (ETL)** processes, model training, and scoring workloads.
  - Enable **feature versioning** for point-in-time queries.
  - Support various data modalities, including **tabular data**, **images**, and **text**.
- **ML data assets** can be managed at the **entity features level** or at the **full dataset level**.
    - For example, a **feature repository** might contain an entity called **customer**, which includes features like age group, postal code, and gender.
  - A **dataset repository** might include a **customer churn dataset**, which includes features from the customer and product entities, as well as purchase- and web-activity event logs.

MLOps Capability **- ML Metadata and Artifact Tracking**

- Various types of **ML artifacts** are produced in different processes of the **MLOps lifecycle**, including descriptive statistics and data schemas, trained models, and evaluation results.
- **ML metadata** is the information about these artifacts, including their location, types, properties, and associations to experiments and runs.
- The **ML metadata** and **artifact tracking capability** is foundational to all other **MLOps capabilities**:
  - Enables **reproducibility** and **debugging** of complex ML tasks and pipelines.
- **Key functionalities** include:
  - Provide **traceability** and **lineage tracking** of ML artifacts.
  - Share and track experimentation and pipeline parameter configurations.
  - Store, access, investigate, visualize, download, and archive ML artifacts.
  - Integrate with all other **MLOps capabilities**.

11. INFRASTRUCTURE: STORAGE AND COMPUTE

What does **infrastructure** mean?

ML systems are **complex**. There are **more components**, and a **request** might jump **20-30 hops** before response. But when a **problem** occurs, where is it?
More complex systems require **better infrastructure**.

The set of **fundamental facilities** and **systems** that support the **sustainable functionality** of households and firms.

ML infrastructure:
The set of fundamental facilities that support the **development** and **maintenance** of ML systems. Every company's **infrastructure needs** are different.

Infrastructure Layers

STORAGE LAYER:

- Where **data** is **collected** and **stored**.
- Simplest form: **HDD**, **SSD**.
- More complex forms: **data lake**, **data warehouse**.
- Examples: **S3**, **Redshift**, **Snowflake**, **BigQuery**.

Most companies use **storage provided by other companies** (e.g., **cloud**).
Storage has become so **cheap** that most companies just **store everything**.

COMPUTE LAYER:

- **Engine** to **execute** your **jobs**.
- **Compute resources** a company has access to.
- Mechanism to determine how these resources can be used.

Simplest form: a single **CPU/GPU core**. Most common form: **cloud compute**.

Compute unit:

- Compute layer can be sliced into smaller compute units to be used **concurrently**.
- A CPU core might support **2 concurrent threads**.
- Each thread is used as a compute unit to execute its own job.

Compute layer: how to execute jobs:
1. Load data into **memory**.
2. Perform operations on that data.
    - Operations: **add**, **subtract**, **multiply**, **convolution**, etc.

To add arrays A and B:
1. Load A & B into memory.
2. Perform addition on A and B.

Compute layer: memory:
- **Amount of memory**.
- Straightforward.
- An instance with **8GB of memory** is more **expensive** than an instance with **2GB of memory**.

I/O bandwidth: speed at which data can be **loaded into memory**.
Compute layer: speed of ops:
- Most common metric: **FLOPS** (Floating Point Operations Per Second).

Utilization:
- Utilization = actual FLOPS / peak FLOPS.
- Dependent on how fast data can be loaded into memory.

Public Cloud vs. Private Data Centers:
- Like storage, compute is largely **commoditized**.

Benefits of cloud:
- Easy to get started.
- Appealing to variable-sized workloads.

Drawbacks of cloud: cost:
- Cloud spending: ~50% cost of revenue.

Cloud repatriation:
- Process of moving workloads from cloud to private data centers.

Multicloud strategy:
- To optimize cost.
- To avoid cloud vendor lock-in.

12. DEVELOPMENT ENVIRONMENTS

Development Environment: Notebook

- Text editors & notebooks
- Where you write code, e.g., **VSCode**, **Vim**

Notebook: **Jupyter notebooks**, **Colab**

- Also works with arbitrary artifacts that aren't code (e.g., images, plots, tabular data)
- **Stateful**
- Only need to run from the failed step instead of from the beginning

Development Environment @Netflix

- Notebook at Netflix

VERSIONING

- **GIT**: CODE VERSIONING
- **DVC**: DATA VERSIONING
- **WANDB**: EXPERIMENT VERSIONING

CI/CD TEST SUITE

- TEST YOUR CODE BEFORE PUSHING IT TO STAGING/PROD
- **EV ENV**: UNDERESTIMATED
- **STANDARDIZE DEV ENVIRONMENTS**
    - STANDARDIZE DEPENDENCIES WITH VERSIONS
    - STANDARDIZE TOOLS & VERSIONS
    - STANDARDIZE HARDWARE: CLOUD DEV ENV

SIMPLIFY IT SUPPORT

- SECURITY: REVOKE ACCESS IF LAPTOP IS STOLEN
- BRING YOUR DEV ENV CLOSER TO PROD ENV
- MAKE DEBUGGING EASIER
    - DEV TO PROD
    - ELASTIC COMPUTE: CAN STOP/START INSTANCES AT WILL
    - HOW TO RECREATE THE REQUIRED ENVIRONMENT IN A NEW INSTANCE?

ANSWER: CONTAINER

- STEP-BY-STEP INSTRUCTIONS ON HOW TO RECREATE AN ENVIRONMENT IN WHICH YOUR MODEL CAN RUN:
    - INSTALL THIS PACKAGE
    - DOWNLOAD THIS PRETRAINED MODEL
    - SET ENVIRONMENT VARIABLES
    - NAVIGATE INTO A FOLDER, ETC.

- **CONTAINER ORCHESTRATION**
- HELP DEPLOY AND MANAGE CONTAINERIZED APPLICATIONS TO A SERVERLESS CLUSTER
- SPINNING UP/DOWN CONTAINERS

13. RUNTIME ENVIRONMENTS

Preparing for Production: Ensuring that a solution tested in the lab functions effectively in real-world scenarios is pivotal, especially for machine learning models. The shift from **development** to **production environments** introduces significant complexities and risks, demanding thorough understanding and testing to mitigate potential issues.

Runtime Environments

Transitioning a model into **production** hinges on its technical feasibility. **Production environments** vary widely, ranging from custom-built services to dedicated platforms like TensorFlow Serving or Kubernetes clusters. Automated deployment processes are favored in **MLOps systems**, influencing the choice of **runtime environments**.

Ideal Scenario vs Reality

While the ideal scenario entails seamless migration of models from **development** to **production**, this isn't always feasible. Challenges often arise, necessitating adjustments or even complete reimplementations. Delays in **production deployment** are not uncommon due to insufficient tooling and processes.

Adaptation from Development to Production

The degree of adaptation required varies widely. In some cases, models can seamlessly transition between environments, while in others, extensive rework or even rewriting is necessary. Inadequate tooling and processes can significantly impede progress.

Tooling Considerations

Early consideration of the **production format** is crucial, as it impacts both the model and the **production pipeline**. Establishing **tooling** during model development, preferably before completion, is essential to prevent bottlenecks in the **validation process**.

Performance Considerations

Performance disparities between **development** and **production environments** are common, particularly in latency-sensitive applications. Optimizing model performance through techniques like **quantization** and **pruning** is essential, especially for resource-constrained environments.

Data Access Before Validation and Launch
Access to relevant data is critical for model validation and deployment. While some data can be bundled with the model, real-time access to databases is often necessary. Managing data access and configuration complexities requires robust **tooling** and collaboration. Addressing these challenges is paramount before proceeding with **validation and deployment**.

14. MACHINE LEARNING PLATFORM

The Need
Navigating the machine learning lifecycle can be daunting, from data gathering to model deployment and management. With the proliferation of ML-powered applications, the burden on data scientists and ML engineers to scale models becomes overwhelming. Supporting these operations requires minimizing the engineering overhead associated with building, deploying, and maintaining models.

Machine Learning Platforms
These platforms aim to streamline MLOps by consolidating components from development to production. They provide the necessary tools and infrastructure for teams to operate models at scale while adhering to standard engineering and MLOps principles.

What is a Machine Learning Platform?
A Machine Learning (ML) platform standardizes the technology stack for data teams, reducing complexities and enabling seamless collaboration across projects and workflows. It facilitates MLOps at every stage of the ML project lifecycle, ensuring scalability as the number of models in production grows.

MLOps Principles:
Effective ML platforms should address key MLOps principles, including reproducibility, versioning, automation, monitoring, testing, collaboration, and scalability. These principles guide the design and goals of the platform, ensuring its efficacy in managing ML workflows.

14.1 Users of Machine Learning Platforms

1. **Subject Matter Experts (SMEs):** Non-developer experts in the business problem, collaborating across the ML lifecycle to ensure data alignment with business needs and model performance.
2. **ML Engineers and Data Scientists:** Responsible for model development, experimentation, and deployment, often blurring the lines between roles due to evolving workflows.
3. **DevOps Engineers:** Responsible for CI/CD pipeline management, ensuring operationalization and integration of ML models within the organizational stack.

4. **Others:** Data engineers, analytics engineers, and data analysts may also utilize the platform for data integration and business intelligence.

ML Platform Architecture

- **Data Stack and Model Development Stack:** Includes components like data and feature stores, experimentation, model registry, and ML metadata repositories.
- **Model Deployment and Operationalization Stack:** Encompasses production environments, model serving, monitoring, and responsible AI and explainability.
- **Workflow Management Component:** Involves model deployment CI/CD pipelines, training formalization, orchestrators, and test environments.
- **Core Technology Stack:** Comprises programming languages, collaboration tools, libraries/frameworks, and infrastructure/compute resources.

Key Components

1. **Data and Feature Store:** Centralized storage for sharing and accessing features, ensuring consistency across model development and inference.
2. **Experimentation Component:** Tracks model iterations and parameters, facilitating informed decision-making during training.
3. **Model Registry:** Organizes and stores validated training models and associated metadata for efficient management and deployment.
4. **ML Metadata and Artifact Repository:** Stores model performance metrics and artifacts, aiding comparison and testing in production environments.

Building an effective ML platform requires careful consideration of users' needs, adherence to MLOps principles, and integration of essential components across the ML lifecycle. By addressing these aspects, organizations can streamline their ML workflows and achieve scalable and efficient model operations.

14.2 Considerations when Deciding on the Scope of the ML Platform

Enterprise Machine Learning Platform vs Startup ML Platform:
The choice between an **enterprise-scale** and a **startup-scale ML platform** is pivotal, especially considering the unique needs of **data scientists** in large organizations. It's imperative to address two fundamental themes when mastering **MLOps**: **Transparency** and pure **efficiency**. In **enterprise settings**, where the stakes are high, **speed** and **agility** become paramount.

Differences in ML Platforms at Start-ups Compared to Enterprises:

- **Reasonable Scale ML Platform:** Tailored for companies operating within **moderate-scale parameters**, both in terms of **revenue** and **team size**. These platforms cater to models that generate significant **revenue** but may not reach the scale of industry giants. They typically involve a finite **computing budget** and a manageable **data workload**.
- **Hyper-scale ML Platform:** Designed to support extensive **model deployments**, often dealing with **hundreds to thousands of models** simultaneously. These platforms are characteristic of large-scale enterprises with substantial **revenue streams**, extensive **user bases**, and massive **data volumes**.

Data-Sensitive ML Platform
Ensuring **data privacy** is paramount when constructing an ML platform. This involves implementing safeguards to prevent inadvertent or malicious **data leaks**, such as those caused by **adversarial attacks**. Compliance with relevant **laws**, **regulations**, and **industry standards** is non-negotiable, necessitating robust mechanisms to protect **sensitive information**.

Human-in-the-Loop ML Platforms
Despite the push for **automation**, **human oversight** remains indispensable. ML platforms must facilitate **manual evaluation** of **data** and **model quality**, particularly in identifying and rectifying unforeseen **data quality issues**. This **human involvement** is crucial, especially in scenarios where automated processes may struggle to detect nuanced problems.

Building an ML Platform for Special Industry Verticals
Flexibility and **adaptability** are key considerations when developing ML platforms for specific industry verticals. Such platforms must accommodate diverse **data types**, varied **model requirements**, and unique **legal and regulatory constraints**. Additionally, they should support **collaborative workflows** tailored to the specific needs and structures of each industry.

End-to-End vs Canonical Stack ML Platform
Organizations must weigh the decision between adopting an all-inclusive **end-to-end ML platform** or integrating individual **point solutions** for each component. This choice depends on factors such as organizational context, business requirements, and user preferences. While **end-to-end platforms** offer seamless integration, **canonical stack solutions** provide greater **flexibility** and **customization**.

Closed vs Open End-to-End ML Platform
The choice between **closed** and **open end-to-end ML platforms** hinges on factors such as **vendor lock-in**, **interoperability**, and **infrastructure compatibility**. Closed platforms offer tight integration but may limit **flexibility** and **choice**. On the other hand, **open platforms** provide greater **freedom** to customize and integrate with existing tools and infrastructure.

Ultimately, the decision between **managed** and **open-source ML platforms** is contingent upon organizational needs, existing infrastructure, and strategic considerations regarding **build vs. buy options**. It's essential to evaluate the pros and cons of each approach carefully before making a decision.

15. ML TOOLS LANDSCAPE

ML-Based Applications
At the core of **ML-based applications** lie three indispensable components: **Data**, **ML Model**, and **Code**.

MLOps / ML Engineering
MLOps, or **Machine Learning Engineering**, represents the seamless integration of **DevOps** principles with **Machine Learning** and **Data Science** assets. It elevates these assets to first-class citizens within the **DevOps** ecosystem.
Machine Learning Engineering (MLE), as described by **A. Burkov**, involves leveraging scientific principles, tools, and techniques from both **machine learning** and **traditional software engineering**. It encompasses all stages of system development, from **data collection** to **model deployment**.

Moving to Production
Transitioning **ML models** from development to production presents numerous challenges for both traditional and tech companies. These challenges include **cloud migration**, **ML pipeline creation**, **scaling**, and **handling sensitive data** at scale.
To address real-life **business problems** effectively, organizations must tackle essential tasks such as **data acquisition**, **cleaning**, **experiment tracking**, **model deployment**, and **monitoring**.

MLOps (Revisited)
MLOps involves key phases such as **data collection**, **model training**, **deployment**, and **monitoring**. It has evolved significantly to meet the growing demands of ML-driven applications.

MLOps Tools Landscape
Navigating the **MLOps tools landscape** can be challenging due to its complexity. **MLOps solutions** broadly fall into two categories: **fully managed services** and **end-to-end platforms**.

Fully Managed Services
Leading **commercial solutions** such as **Amazon Sagemaker**, **Microsoft Azure MLOps suite**, and **Google Cloud MLOps suite** offer comprehensive tools for building, training, deploying, and monitoring ML models.

End-to-End Platforms

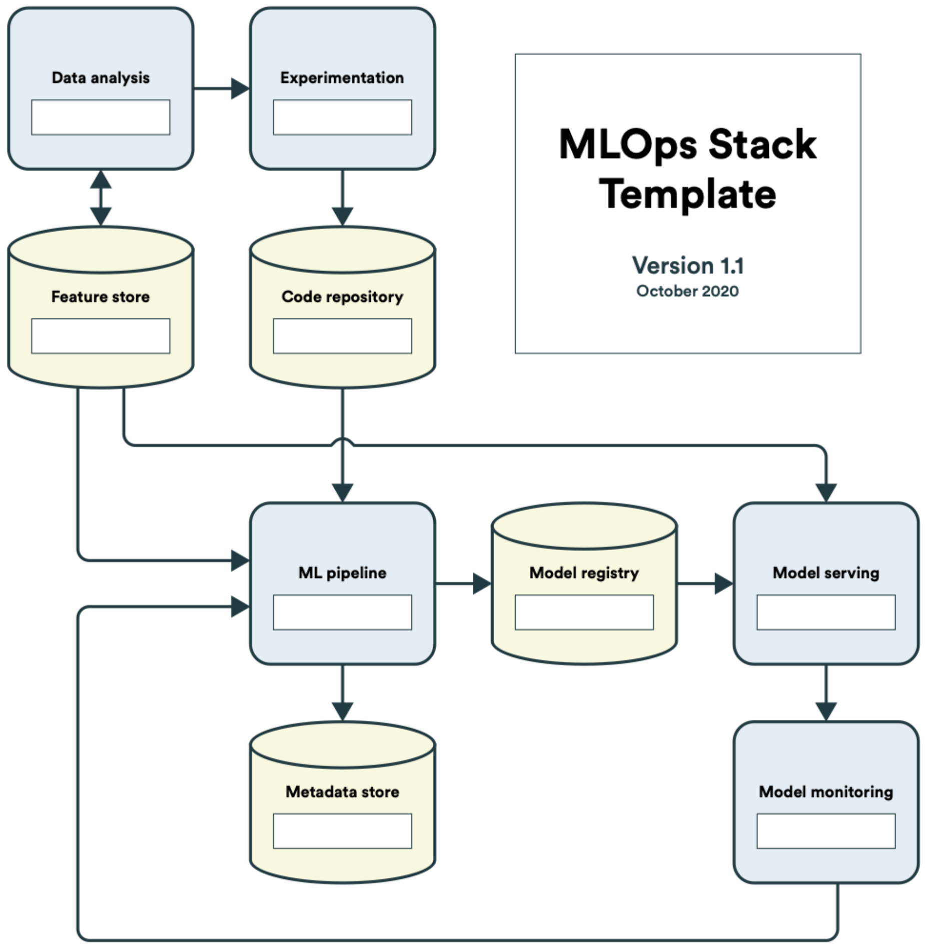
While **end-to-end solutions** provide convenience, organizations can also opt to build their **MLOps pipelines** using preferred tools. This approach offers flexibility, scalability, and customization, reducing the risk of a single point of failure (**SPOF**).

Top MLOps Tools

Popular **MLOps** **tools** include **Project** **Jupyter**, **Airflow**, **Kubeflow**, **MLflow**, and **neptune.ai**. Alternatively, organizations can develop **custom-built MLOps solutions** tailored to their specific requirements.

Industry Examples

Various industry players, including **Uber** with its **Michelangelo Platform**, have developed **MLOps architectures** to streamline their ML workflows. **Open MLOps** initiatives aim to promote collaboration and standardization within the MLOps community.



16. ML EXPERIMENT TRACKING

A Story Heard Too Often
"… So far we have been doing everything manually and sort of ad hoc. Some people are using it, some people are using that, it's all over the place. We don't have anything standardized. But we run many projects, the team is growing, and we are scaling pretty fast. So we run into a lot of problems. How was the model trained? On what data? What parameters we used for different versions? How can we reproduce them? We just feel the need to control our experiments…" – *unfortunate Data Scientist.*

A Plethora of Experiments
When developing **ML models**, you'll inevitably conduct numerous experiments. These experiments may vary in terms of the **models**, **hyperparameters**, **data**, **code**, and **environment** used, leading to diverse evaluation metrics.

Challenges of Experiment Tracking
Managing and tracking all this information can quickly become daunting, especially when you need to organize and compare experiments to identify the best-performing setup.

The Solution: ML Experiment Tracking!
Experiment tracking involves systematically recording all relevant experiment-related information, including scripts, environment configurations, data versions, parameter settings, evaluation metrics, and model artifacts.

Components of Experiment Tracking
An effective experiment tracking system typically comprises three components:

1. **Experiment Database:** Stores and logs experiment metadata.
2. **Experiment Dashboard:** Provides a visual interface for exploring experiment metadata.
3. **Client Library:** Offers methods for logging and querying data from the experiment database.

Integration with MLOps
Experiment tracking is a vital aspect of **MLOps**, which encompasses the entire ML project lifecycle. While MLOps manages the operationalization of machine learning, experiment tracking focuses on the iterative model development phase.

Why Experiment Tracking Matters

Experiment tracking enhances the ML workflow in several ways:

1. **Centralized Organization:** All experiment results are stored in a single repository, making it easy to search, compare, and reproduce experiments.
2. **Efficient Comparison:** With experiment metadata consolidated, comparing parameters, metrics, and model performance becomes seamless.
3. **Enhanced Insights:** Analyzing experiment data facilitates the discovery of new insights and ideas, fostering collaboration and innovation.
4. **Standardized Logging:** Experiment tracking ensures consistent logging of experiment details, preventing information loss or inconsistency.

Key Elements to Track in ML Experiments

- **Code:** Scripts, notebooks, or utilities used for preprocessing, training, and evaluation.
- **Environment:** Configuration files (e.g., Dockerfile, requirements.txt) defining the execution environment.
- **Data:** Versions or locations of training and evaluation data.
- **Parameters:** Configuration settings used for each experiment.
- **Metrics:** Evaluation metrics logged for train, validation, and test sets.

Setting Up Experiment Tracking

While manual methods like spreadsheets or versioning with GitHub are feasible, dedicated experiment tracking tools offer superior functionality and ease of use. These tools provide centralized repositories, streamlined logging, and powerful comparison features tailored to ML workflows.

17. ML MODEL REGISTRY

17.1 What is Model Registry?

Questions from Cross-Functional Team:

- Where can we find the best version of this model so we can audit, test, deploy, or reuse it?
- How was this model trained?
- How can we track the docs for each model to ensure they are compliant and people can know the necessary details about it, including the metadata?
- How can we review models before they are put to use or even after they have been deployed?
- How can we integrate with tools and services that make shipping new projects easier?

Understanding the Model Registry

An **ML model registry** serves as a centralized repository, facilitating effective **model management** and **documentation**. It enables clear **naming conventions**, comprehensive **metadata**, and improved **collaboration** between data scientists and operations teams, ensuring smooth **deployment** and **utilization** of trained models.

Differentiating Model Registry vs. Model Repository:

While a **model repository** is a storage location for machine learning models, a **model registry** is a more comprehensive system that tracks and manages the full lifecycle of machine learning models. However, both terms are often used interchangeably, with specific definitions varying based on context or tools and platforms being used.

Integration with Experiment Tracking

A model registry must integrate with the **experiment management system** to register models from various experiment runs, making them easier to find and work with.

Key Components of Model Registry:

- Acts as a centralized storage for effective collaboration
- Provides visibility over models and their status
- Enables model versioning and comparison
- Integrates with systems outputting trained models
- Facilitates integration with staging environments
- Supports automation for scalable software development
- Provides interfaces for downstream systems to consume models

Why Model Registry Matters:

1. **Centralized Organization:** All models and their metadata are stored in a single repository, enhancing collaboration and ease of access.
2. **Efficient Comparison:** Users can compare different versions of a model, track performance metrics, and select the best version for deployment.
3. **Version Control:** Model registry maintains a history of all registered models and their versions, ensuring traceability and retention.
4. **Integration with Pipelines:** Automation tools and custom APIs allow seamless integration with pipeline automation tools, enabling continuous training and delivery of models.
5. **Deployment Efficiency:** Integration with downstream services and deployment tools streamlines the deployment process, enhancing model deployment efficiency.

In essence, a well-implemented **ML model registry** is essential for effective model management, collaboration, and deployment, ensuring the success of machine learning projects.

17.2 Where Does a Model Registry Fit in the MLOps Stack?

- To efficiently run machine learning projects at scale, a **model registry** is likely needed in the MLOps stack.
- Depending on the level of implementation in the MLOps stack, the **needs** and **requirements** for a model registry would differ.

Position in the MLOps Stack:

- The model registry sits **between** machine learning development and deployment.

In MLOps Level 0:

- Manual process: Data scientist prepares the model artifact and metadata.
- Operations team pushes the packaged model to the staging environment for testing before deployment.
- Acts as a **central repository** for models.

In MLOps Level 1:

- Goal: Perform **continuous training** of the model by automating the ML pipeline.
- Model registry integrates with the pipeline, enabling staging and testing of models.
- Output from training is fed into the model registry for staging and deployment.

In MLOps Level 2:

- Similar to Level 1: Automated pipeline delivers trained models to the model registry.
- Model registry plays a crucial role in automated pipelines, promoting models with good metrics and archiving others.

17.3 Setting Up ML Model Registry – Build, Maintain, or Buy?

Building a Model Registry Solution:

- Requires object storage, database, API integration, and UI for ML teams.
- Consider factors like **incentive**, **human resources**, **time**, **operations**, and **cost**.

Maintaining an Existing Solution:

- Open-source solutions may require managing features like object storage and database.
- Consider factors like **type of solution**, **operations**, **cost**, **features**, **support**, and **accessibility**.

Subscribing to a Fully Managed Solution:

- No need to worry about building or maintaining a solution.
- Ensure systems and services can integrate with the registry.
- Consider factors like **industry type**, **features**, **cost**, **security**, **performance**, **availability**, **support**, and **accessibility**.

ML Model Registry Solutions Out There:
Several popular solutions offer a host of model registry features.

In summary, choosing the right approach for setting up an ML model registry depends on factors like requirements, resources, and preferences, ensuring efficient management and deployment of machine learning models in the MLOps workflow.

18. METADATA

18.2 Introduction

What is ML Metadata?

- ML involves various **models**, from **supervised classifiers** to **optimization algorithms**, all processing **data** to make **decisions**.
- Delivering a model into **production** entails **training**, **tuning**, **debugging**, **evaluating**, and **comparing** against **baselines**.
- ML metadata encompasses information about each step, including **training parameters**, **evaluation metrics**, and **dataset versions**.

Need for Metadata

- For some, **messy experimentation** is the issue, while others lack understanding of deployed **models** or struggle with orchestrating model **A/B testing** and retraining **pipelines**.
- An ML metadata store addresses these challenges and more.

Metadata about Experiments and Model Training Runs

- During **experimentation**, log **data versions**, **environment configurations**, **code versions**, **hyperparameters**, and **training metrics**.
- **Hardware metrics**, **evaluation/test metrics**, **performance visualizations**, and **model predictions** are also logged.

Metadata about Artifacts

- **Artifacts** include **datasets**, **models**, **predictions**, and other file-like objects used across projects.
- Log **references**, **versions**, **previews**, **descriptions**, **authors**, and other relevant details for each **artifact**.

Metadata about Trained Models

- Trained **models** are crucial **artifacts**, requiring logging of **model package**, **version**, **evaluation records**, **experiment versions**, and more.
- Downstream **datasets/artifacts**, **drift-related metrics**, and **hardware monitoring** are also logged.

Metadata about Pipeline

- **Pipeline metadata** aids efficient computation, logging **input/output steps**, **cached outputs**, and **pipeline triggers**.

Dataset Metadata

- **Provenance**: Origin of the data, including logs, external data provider keys, or code references.
- **Location**: Storage reference for raw data, ongoing data creation sources, or permitted data access points.
- **Responsible Person or Team**: Team accountable for downloading or creating the dataset.
- **Creation Date or Version Date**: Date of dataset creation or latest version update.
- **Use Restrictions**: Restrictions on dataset usage due to licensing or governance constraints.

Feature Metadata

- **Version Definition**: Description of feature data and processing method, updated with each version.
- **Responsible Person or Team**: Authorship or maintainer information for the feature.
- **Creation Date or Version Date**: Date of feature creation or latest version update.
- **Use Restrictions**: Restrictions on feature usage in specific contexts.

Label Metadata

- **Definition Version**: Version of label definitions used.
- **Set Version**: Version changes due to corrections or additions.
- **Source**: Origin of labels, whether licensed, human-produced, or algorithm-generated.
- **Confidence**: Confidence level of label correctness.

Pipeline Metadata

- Data about intermediate artifacts, pipeline runs, and produced binaries.
- Automatically generated by some ML training systems like ML Metadata (MLMD) in TensorFlow Extended (TFX).

Metadata Systems Overview

- Essential for tracking feature definitions, label versions, and pipeline processes.
- Building a solid metadata system is crucial; starting without one leads to regrets later.
- Options: Build one comprehensive system or multiple subsystems targeted at specific tasks.

Choices for a Metadata System

- **One System**: Simple correlations across subsystems but difficult schema management.
- **Multiple Systems**: Decoupled development but challenges in analysis and reporting across systems.

Conclusion

Metadata systems are often overlooked but are essential for effective ML system use. They unlock value and should be prioritized for development and maintenance.

18.2 Model Metadata Store

What is an ML Metadata Store?

- An ML metadata store is a **repository** for ML model-related **metadata**, facilitating logging, storage, display, comparison, and querying.
- It serves as a **central database** and **user interface** tailored for ML model metadata management.

Repository vs Registry vs Store

- A **metadata repository** stores **metadata objects** and **relationships**, while a **registry** checkpoints important metadata, and a **store** serves as a **central hub** for finding metadata.
- ML metadata store functions as a **centralized place** for **model**, **experiment**, **artifact**, and **pipeline-related metadata**.

Two Flavors of ML Metadata Store

- Depending on focus, ML metadata store treats either **pipelines** or **models** as first-class objects, impacting its functionalities.

Setting up an ML Metadata Management System

- Consider whether to **build**, **maintain open source**, or **buy** a solution based on the problem, available tools, and required effort.

19. 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.

19.1 Why Package ML Models?

MLOps enables a **systematic approach** to train and evaluate models. After models are trained and evaluated, the next steps are to bring them to **production**.
ML doesn't work like traditional software engineering, which is **deterministic** in nature. Engineering ML solutions is **non-deterministic** and involves serving ML models to make predictions or analyze data.

To serve the models, they need to be **packed** into software artifacts to be shipped to the testing or production environments. ML models need to be packaged for the following reasons:

- **Portability**
- **Inference**
- **Interoperability**
- **Deployment agnosticity**

How to Package ML Models?
ML models can be packaged in various ways depending on business and tech requirements and as per operations for ML. ML models can be packaged and shipped in three ways:

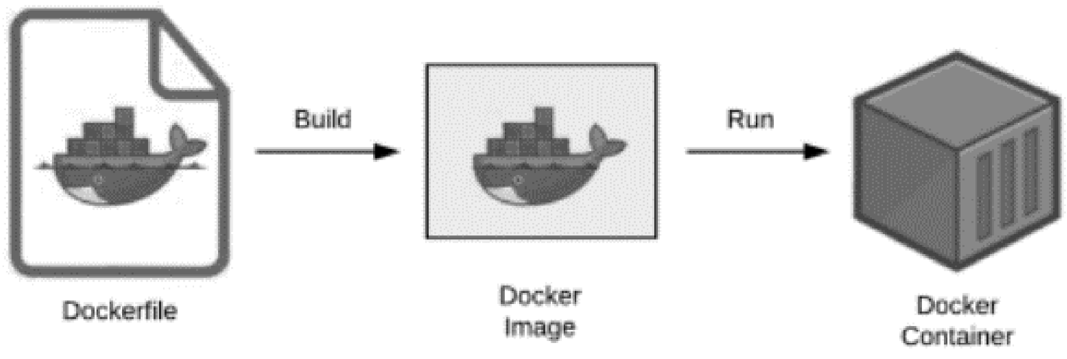
- **Serialized files**
- **Packetizing or containerizing**
- **Microservice generation and deployment**

Deployment Strategies
Common ways for wrapping trained models as deployable services, namely deploying ML models as **Docker Containers** to **Cloud Instances** or **Serverless Functions**.

19.2 Packetizing or containerizing

Every environment possesses different challenges when it comes to deploying ML models, in terms of compatibility, robustness, and scalability. Containers are a great way to standardize ML models and software modules.

Docker - industry standard at developing and orchestrating containers.



Docker artifacts

Microservices
Typically, a microservice is generated by tailoring serialized files into a containerized Docker image. Docker images can then be deployed and orchestrated into any Docker-supported environment.

Deploying ML Models as Docker Containers
As of now, there is no standard, open solution to ML model deployment! ML model inference being considered stateless, lightweight, and idempotent, containerization becomes the de-facto standard for delivery. For on-premise, cloud, or hybrid deployments, Docker is considered to be de-facto standard containerization technology.

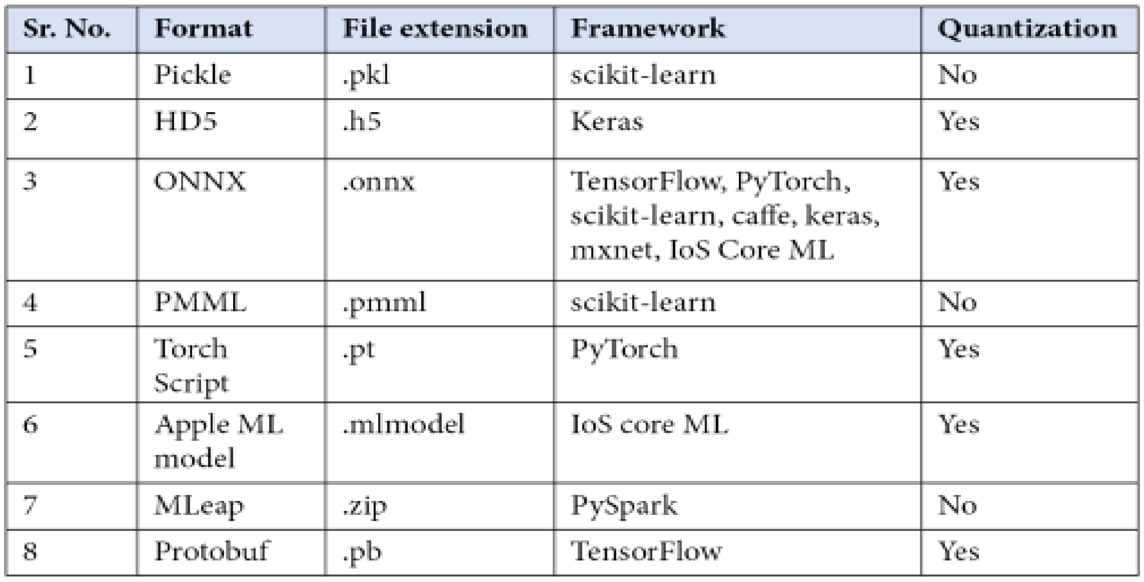
Deploying ML Models as Serverless Functions
Various cloud vendors already provide machine-learning platforms where can deploy model with their services. In order to deploy an ML model as a serverless function, the application code and dependencies are packaged into .zip files, with a single entry point function.

19.3 ML Model serialisation format

Serialization is a vital process for packaging an ML model, enabling model **portability**, **interoperability**, and model **inference**.

Serialization is the method of converting an object or a data structure into a storable artifact, such as a file or a memory buffer.

Model File Formats
In supervised machine learning, the **artifact** created after training used to make predictions on new data is called a **model**. For example, after training a **deep neural network** (DNN), the trained model is basically a file containing the layers and weights in the DNN.



Sr. No.	Format	File extension	Framework	Quantization
1	Pickle	.pkl	scikit-learn	No
2	HD5	.h5	Keras	Yes
3	ONNX	.onnx	TensorFlow, PyTorch, scikit-learn, caffe, keras, mxnet, IoS Core ML	Yes
4	PMML	.pmml	scikit-learn	No
5	Torch Script	.pt	PyTorch	Yes
6	Apple ML model	.mlmodel	IoS core ML	Yes
7	MLeap	.zip	PySpark	No
8	Protobuf	.pb	TensorFlow	Yes

Often, models can be saved in a file that can potentially be compressed, so typically model files have a **binary** file format.

- **TensorFlow** saves models as **protocol buffer files** (.pb).
- **Keras** saves models natively as **.h5** files.
- **Scikit-Learn** saves models as **pickled** python objects (.pkl).
- An older format for model serving based on XML, **predictive model markup language** (.pmml), is still usable on some frameworks, such as Scikit-Learn.

Model files are used to make predictions on new data by either batch applications or real-time model serving servers.

ML Model Serialization Formats

Various formats to distribute ML models exist. In order to achieve a **distributable** format, the ML model should be present and executable as an independent asset. For example, you might want to use a Scikit-learn model in a Spark job, which means that the ML models should work outside of the model-training environment. Two broad exchange formats for ML models:

- **Language-agnostic**
- **Vendor-specific**

Language-agnostic exchange formats

- **Amalgamation** is the simplest way to export an ML model - model and all necessary code to run are bundled as one package. For example, you can create a standalone version of an ML model by using SKompiler, which transforms trained Scikit-learn models into other forms.
- **PMML** is a format for model serving based on XML. It describes a model and pipeline in XML but does not support all of the ML algorithms.
- **PFA** (Portable Format for Analytics) is designed as a replacement for PMML. It describes an executable called a scoring engine in JSON-formatted text.
- **ONNX** (Open Neural Network eXchange) is an ML framework independent file format supported by many big tech companies. It allows any ML tool to share a single model format.
- **YAML** is used to package models as part of the MLFlow framework for ML pipelines on Spark.

Vendor-specific Exchange formats

- **Scikit-Learn**: Saves models as pickled python objects (.pkl).
- **H2O**: Allows converting models built to either POJO or MOJO.
- **SparkML**: Models can be saved in the MLeap file format and served in real-time using an MLeap model server.
- **TensorFlow**: Saves models as .pb files.
- **PyTorch**: Serves models by using their proprietary Torch Script as a .pt file.

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 in other ML frameworks need to use tools like coremltools and Tensorflow converter to translate their ML model files to the .mlmodel format for use on iOS.

20. MODEL DEPLOYMENT

Deploy is a loose term that generally means making model running and accessible. To be deployed, a **model** will have to leave the development environment. During model development, the model usually runs in a development environment and can be deployed to a staging environment for testing or to a production environment to be used by end users.

Production is a broad spectrum. For some teams, production means generating nice plots in notebooks to show to the business team. For other teams, production means keeping your models up and running for millions of users a day.

If your work is closer to the second scenario, welcome to the world of **model deployment**! Deployment is easy! If you want to deploy a model for friends to play with, all you need to do is to wrap the predict function in a POST request endpoint using **Flask** or **FastAPI**, put the dependencies this predict function needs to run in a container, and push the model and its associated container to a cloud service like **AWS** or **GCP** to expose the endpoint. You can use this exposed endpoint for downstream applications. If you're familiar with the necessary tools, you can have a functional deployment in an hour!

20.1 Machine Learning Deployment Myths

Deploying an ML model can be very different from deploying a traditional software program. This difference might cause people who have never deployed a model before to either dread the process or underestimate how much time and effort it will take.

Common myths about the deployment process:

1. **Myth 1:** You Only Deploy One or Two ML Models at a Time
2. **Myth 2:** If We Don't Do Anything, Model Performance Remains the Same
3. **Myth 3:** You Won't Need to Update Your Models as Much
4. **Myth 4:** Most ML Engineers Don't Need to Worry About Scale

Deployment: Reality

The hard parts include making the model available to millions of users with a latency of milliseconds and 99% uptime, setting up the infrastructure so that the right person can be immediately notified when something goes wrong, figuring out what went wrong, and seamlessly deploying the updates to fix what's wrong.

In many companies, the responsibility of deploying models falls into the hands of the same people who developed those models. However, this separation of responsibilities can cause high overhead communications across teams and make it slow to update the model. It also can make it hard to debug should something go wrong.

20.2 Productionalization and Deployment

A key component of **MLOps** presents an entirely different set of technical challenges than developing the model and lies in the domain of the software engineer and the DevOps team. Organizational challenges in managing the information exchange between the data scientists and these teams must not be underestimated. Without effective collaboration between the teams, delays or failures to deploy are inevitable.

Model Deployment Types and Contents
Commonly two types of model deployment: **Model-as-a-service** or **live-scoring model** and **Embedded model**.

What to-be-deployed models consist of depends on the technology chosen but typically comprises a set of code (commonly Python, R, or Java) and data artifacts. It can have version dependencies on runtimes and packages that need to match in the production environment because the use of different versions may cause model predictions to differ.

Model Deployment: Dependency Management
Exporting the model to a portable format such as **PMML**, **PFA**, **ONNX**, or **POJO** improves model portability between systems and simplifies deployment. However, each format supports a limited range of algorithms, and sometimes the portable models behave in subtly different ways than the original.

Model Export
Containerization is an increasingly popular solution to the headaches of dependencies when deploying ML models. Container technologies such as **Docker** are lightweight alternatives to virtual machines, allowing applications to be deployed in independent, self-contained environments matching the exact requirements of each model.

Model Deployment Requirements

In customer-facing, mission-critical use cases, a more robust **CI/CD pipeline** is required, which typically involves ensuring all **coding**, **documentation**, and sign-off standards have been met, re-creating the **model** in something approaching the **production environment**, revalidating the **model accuracy**, performing **explainability checks**, ensuring all **governance requirements** have been met, checking the **quality** of any **data artifacts**, testing **resource usage** under load, and embedding into a more complex **application**, including **integration tests**.

One thing is for sure: rapid, automated deployment is always preferred to labor-intensive processes. In heavily regulated industries (e.g., finance and pharmaceuticals), **governance** and **regulatory checks** will be extensive and are likely to involve manual intervention. The desire in **MLOps**, just as in **DevOps**, is to automate the **CI/CD pipeline** as far as possible, which speeds up the **deployment process**, enables more extensive **regression testing**, and reduces the likelihood of **errors** in the deployment.

20.3 Strategies for Effective Model Deployment

Deploying to **Production**
Business leaders view rapid deployment of systems into production as **key** to maximizing **business value** only true if deployment can be done smoothly and at low **risk**

Lest dive into the concepts and considerations when deploying **machine learning models** to production that impact and drive—the way **MLOps deployment processes** are built.

CI/CD Pipelines
Forms a modern philosophy of **agile software development** and a set of practices and tools to release applications more often and faster, while also better controlling **quality** and **risk**.

Ideas are decades old and already used to various extents by software engineers, different people and organizations use certain terms in very different ways.
Essential to keep in mind that these concepts should be tools to serve the purpose of delivering **quality fast** first step is always to identify the specific risks present at the organization.

CI/CD methodology should be adapted based on the needs of the team and the nature of the business. A common acronym for **continuous integration** and **continuous delivery** (or put more simply, deployment) **CI/CD for ML** CI/CD concept apply just as well to machine learning systems are a critical part of **MLOps strategy**.

An example of such pipeline could be:

1. Build the **model**
2. a. Build the **model** artifacts
3. b. Send the artifacts to long-term storage
4. c. Run basic checks (smoke tests/sanity checks)
5. d. Generate **fairness** and **explainability reports**
6. Deploy to a test environment
7. a. Run tests to validate ML performance, computational performance
8. b. Validate manually
9. Deploy to production environment
10. a. Deploy the **model** as canary
11. b. Fully deploy the **model**

CI/CD for ML

Many scenarios are possible, depend on the application, the risks from which the system should be protected and the way the organization chooses to operate.
Generally, an incremental approach to building a **CI/CD pipeline** is preferred: a simple or even naïve workflow on which a team can iterate often much better than starting with complex infrastructure from scratch.
A starting project does not have the infrastructure requirements of a tech giant can be hard to know up front which challenges deployments will present.
There are common tools and best practices, but there is no one-size-fits-all **CI/CD methodology** means the best path forward is starting from a simple (but fully functional) **CI/CD workflow** then introducing additional or more sophisticated steps along the way as quality or scaling challenges appear.

Building ML Artifacts

The goal of a **continuous integration pipeline** is to avoid unnecessary effort in merging the work from several contributors also to detect bugs or development conflicts as soon as possible.

The very first step is using centralized version control systems unfortunately, working for weeks on code stored only on a laptop is still quite common).
The most common version control system is Git, an open source software majority of software engineers across the world already use Git, increasingly being adopted in scientific computing and data science Git allows for maintaining a clear history of changes, safe rollback to a previous version of the code, multiple contributors to work on their own branches of the project before merging to the main branch, etc.

Git is appropriate for code, but not designed to store other types of assets common in data science workflows, such as large binary files (for example, trained model weights), or to version the data itself.

ML Artifact
Once code and data is in a centralized repository, a testable and deployable bundle of the project must be built are usually called **artifacts** in the context of CI/CD.
Each of the following elements needs to be bundled into an **artifact** that goes through a testing pipeline and is made available for deployment to production: Code for the **model** and its preprocessing **Hyperparameters** and configuration Training and validation data Trained **model** in its runnable form An environment including libraries with specific versions, environment variables, etc. **Documentation** Code and data for testing scenarios.

The Testing Pipeline
Testing pipeline can validate a wide variety of properties of the **model** contained in the artifact good tests should make it as easy as possible to diagnose the source issue when they fail.

Automating tests as much as possible is essential and is a key component of efficient **MLOps**.

A lack of automation or speed wastes time, also discourages the development team from testing and deploying often, which can delay the discovery of bugs or design choices that make it impossible to deploy to production.

DEPLOYMENT CONCEPTS

Integration

Process of merging a contribution to a central repository and performing more or less complex tests typically merging a Git feature branch to the main branch.

Delivery
Same as used in the continuous delivery (CD) part of CI/CD, Process of building a fully packaged and validated version of the **model** ready to be deployed to production.

Deployment
Process of running a new **model** version on a target infrastructure Fully automated deployment is not always practical or desirable.

Release
In principle, release is yet another step, directing production workload to **model** deploying a **model** version (even to the production infrastructure) does not necessarily mean that the production workload is directed to the new version multiple versions of a **model** can run at the same time on the production infrastructure.

CATEGORIES OF MODEL INFERENCES

Batch scoring, where whole datasets are processed using a **model**, such as in daily scheduled jobs **Real-time scoring**, where one or a small number of records are scored, such as when an ad is displayed on a website and a user session is scored by models to decide what to display.
In some systems, scoring on one record is technically identical to requesting a batch of one! In both cases, multiple instances of the **model** can be deployed to increase throughput and potentially lower latency.

TWO WAYS TO APPROACH MODEL DEPLOYMENT

Considerations When Sending Models to Production
When sending a new **model** version to production, first consideration is often to avoid downtime, in particular for real-time scoring.

Blue-green or red-black— deployment basic idea is that rather than shutting down the system, upgrading it, and then putting it back online, a new system can be set up next to the stable one and when it's functional, the workload can be directed to the newly deployed version and if it remains healthy, the old one is shut down).

Canary deployment idea is that the stable version of the **model** is kept in production, but a certain percentage of the workload is redirected to the new **model**, and results are monitored usually implemented for real-time scoring, but a version of it could also be considered for batch.

MAINTENANCE IN PRODUCTION

At a high level, there are three maintenance measures: Resource monitoring Health check ML metrics monitoring.
Resource monitoring Just as for any application running on a server, collecting IT metrics such as CPU, memory, disk, or network usage can be useful to detect and troubleshoot issues.
Health check Need to check if the **model** is indeed online and to analyze its latency simply queries the **model** at a fixed interval (on the order of one minute) and logs the results.

ML metrics monitoring about analyzing the accuracy of the **model** and comparing it to another version or detecting when it is going stale may require heavy computation, this is typically lower frequency.
Finally, when a malfunction is detected, a rollback to a previous version may be necessary critical to have the rollback procedure ready and as automated as possible; testing it regularly can make sure it is indeed functional.
Once a **model** is released, it must be maintained.

20.4 Model Deployment Patterns

Once the **model** has been built and thoroughly tested, it can be **deployed** means to make it available for accepting **queries** generated by the users of the **production system**.

Once the **production system** accepts the **query**, the latter is transformed into a **feature vector**. The **feature vector** is then sent to the **model** as input for **scoring**. The result of the **scoring** then is returned to the user.
A trained **model** can be **deployed** in various ways can be deployed on a **server**, or on a user's **device** can be deployed for all users at once, or to a small fraction of users.

A **model** can be **deployed** following several patterns:

- **statically**, as a part of an installable software package,
- **dynamically** on the user's **device**,
- **dynamically** on a **server**, or
- Via **model streaming**.

Static Deployment
Very similar to traditional software deployment prepare an installable binary of the entire software **model** is packaged as a resource available at the **runtime**.
Depending on the **operating system** and the **runtime environment** Objects of both the **model** and the **feature extractor** can be packaged as a part of a **dynamic-link library (DLL on Windows)**, **Shared Objects (*.so files on Linux)**, or be **serialized** and saved in the standard resource location for **virtual machine**-based systems, such as **Java** and **.Net**.

Dynamic Deployment on User's Device
Similar to a **static deployment**, in the sense the user runs a part of the system as a software application on their **device**. Difference is that in **dynamic deployment**, the **model** is not part of the binary code of the application.
Achieves better **separation of concerns**! **Pushing model updates** is done without updating the whole application running on the user's device.
Dynamic deployment can be achieved in several ways by **deploying model parameters**, by **deploying** a serialized object, and by **deploying to the browser**.

Deployment on a Server
Because of the complications with other approaches, and problems with performance monitoring, Most frequent **deployment pattern** is to place the **model** on a **server** (or servers), make it available as a **Representational State Transfer application programming interface (REST API)** in the form of a **web service**, **Google's Remote Procedure Call (gRPC) service**.

Deployment on a Virtual Machine (VM)
In a typical **web service architecture** deployed in a cloud environment predictions are served in response to canonically-formatted HTTP requests.

A **web service** running on a **virtual machine** receives a user request containing the input data, calls the **machine learning system** on that input data then transforms the output of the **machine learning system** into the output JSON or XML string.

Deployment in a Container
A more modern alternative to a **virtual-machine-based deployment** considered more resource-efficient and flexible than with **virtual machines**.

A **container** is similar to a **virtual machine** in the sense that it is also an isolated runtime environment with its own **filesystem**, **CPU**, **memory**, and **process space**. The main difference, however, is that all containers are running on the same **virtual or physical machine** share the **operating system**, while each **virtual machine** runs its own instance of the **operating system**.

Serverless Deployment
Several cloud services providers, including **Amazon**, **Google**, and **Microsoft**, offer **serverless computing** Lambda-functions on **Amazon Web Services**, and **Functions** on **Microsoft Azure** and **Google Cloud Platform**.

The **serverless deployment** consists of preparing a zip archive with all the code needed to run the **machine learning system** (**model**, **feature extractor**, and **scoring code**) must contain a file with a specific name that contains a specific function is uploaded to the **cloud platform** and registered under a unique name.

Model Streaming

Streaming works differently. All **models**, as well as the code needed to run them, are registered within a **stream-processing engine (SPE)** Apache **Storm**, Apache **Spark**, and Apache **Flink** Or, they are packaged as an application based on a **stream-processing library (SPL)**, such as **Apache Samza**, **Apache Kafka Streams**, and **Akka Streams**. Based on the notion of **data processing topology** Input data flows in as an infinite stream of data elements sent by the client Following a predefined topology, each data element in the stream undergoes a transformation in the nodes of the topology Transformed, the flow continues to other nodes. Can be seen as an inverse to the **REST API**.

21. ESSENTIALS OF MODEL SERVING

21.1 Model Serving in ML Lifecycle

Model Serving

Model built in the training phase needs to be taken into the world so that it can start **predicting**! Aka **Model Serving**.

Process of creating a structure to ensure the system can ask the **model** to make **predictions** on new examples and return those **predictions** to the people or systems that need them.

Properties of the Model Serving Runtime

The **model serving** runtime is the environment in which the **model** is applied to the input data. The **runtime** properties are dictated by the **model deployment pattern**.

However, an effective **runtime** will have several additional properties such as:

- **Security** and **Correctness**
- **Ease of Deployment**
- **Guarantees** of **Model Validity**
- **Ease of Recovery**
- **Avoidance** of **Training/Serving Skew**
- **Avoidance** of **Hidden Feedback Loops**

Security and Correctness

The **runtime** is responsible for authenticating the user's identity and authorizing their requests. Things to check are whether a specific user has authorized access to the **models** they want to run, whether the names and the values of parameters passed correspond to the **model's specification**, whether those parameters and their values are currently available to the user.

Ease of Deployment

The **runtime** must allow the **model** to be updated with minimal effort, ideally, without affecting the entire application. If the **model** was deployed as a **web service** on a physical server, then a **model update** must be as simple as replacing one **model file** with another, and restarting the **web service**. If the **model** was deployed as a **virtual machine instance** or **container**, then the instances or containers running the old version of the **model** should be replaceable by gradually stopping the running instances and starting new instances from a new image. The same principle applies to the orchestrated containers.

Ease of Recovery

An effective **runtime** allows easy recovery from errors by rolling back to previous versions. The recovery from an unsuccessful deployment should be produced in the same way, and with the same ease, as the deployment of an updated **model**. The only difference is that, instead of the new **model**, the previous working version will be deployed.

Avoidance of Training/Serving Skew

When it concerns **feature extraction**, it is strongly recommended to avoid using two different codebases, one for training the **model**, and one for **scoring** in production, even a tiny difference between two versions of **feature extractor code** may lead to suboptimal or incorrect **model performance**. The **runtime** should allow easy access to the **feature extraction code** for various needs, including **model retraining**, **ad-hoc model calls**, and production.

Key Questions for Model Serving

There are a lot of ways to create structures around the **model** that support serving, each with very different sets of trade-offs. It's useful to think through specific questions about the needs of the system:

- What will be the **load** to **model**?
- What are the **prediction latency** needs of **model**?
- Where does the **model** need to live?
- What are the **hardware** needs for **model**?
- How will the serving **model** be stored, loaded, versioned, and updated?
- What will the **feature pipeline** for serving look like?

Load to Model

QPS (Queries Per second) – need to know in serving environment what is the level of traffic that **model** will be asked to handle – when **queries** are done on demand. **Model serving predictions** to millions of daily users may need to handle thousands of **QPS**. **Model** runs **audio recognizer** on **mobile device** may run at a few **QPS**. **Model** predicting real estate prices might not be served on demand at all!

Few basic strategies to handle large traffic loads:

- Replicate **model** across many machines and run these parallel – may be on **cloud**
- Use more powerful **hardware** – accelerators like **GPU** or specialized chips
- Tune computation cost of **model** itself by using fewer features or layers or parameters
- **Model cascades** can be effective at cost reduction

Prediction Latency

Prediction latency is the time between the moment a request is made and the moment a response is received. Acceptable **prediction latency** can vary dramatically among applications and is a major determiner of **serving architecture** choices.

Where does the model need to live?

This choice has significant implications for overall **serving architecture**:

- On a local machine
- On servers owned or managed by our organization
- In the cloud
- On-device

Hardware Needs for Model

Range of computational **hardware** and chip options have emerged, enabled dramatic improvements in serving efficiency for various **model types**:

- Multicore CPUs
- Hardware accelerators – commonly **GPU**
- Specialized hardware

How will the serving model be stored, loaded, versioned, and updated?

Deciding exactly how many versions to be supported and at what capacity is important architectural choice. Requires balancing resourcing, system complexity, and organizational requirements together.

What will feature pipeline for serving look like?

Any feature processing or other data manipulation that is done to data at **training time** will almost certainly need to be repeated for all examples sent to **model** at **serving time**. Actual code used to turn incoming examples data to features of **model** may be different at **serving time** from the code used for similar tasks at **training time**. Promise is in the form of **feature stores** – handle both training and serving together in a single package. **Creating feature for model** to use at **serving time** is key source of **latency**. Feature needs to be processed at serving time as well as at training time.

22. DIFFERENT FORMS OF ML WORKFLOWS

Operating an **ML model** might assume several architectural styles.
Primarily, four **architectural patterns** are classified along two dimensions:

- **ML Model Training**
- **ML Model Prediction**

For the sake of simplicity, disregard the third dimension - **ML Model Type**, which denotes the type of **machine learning algorithm** such as

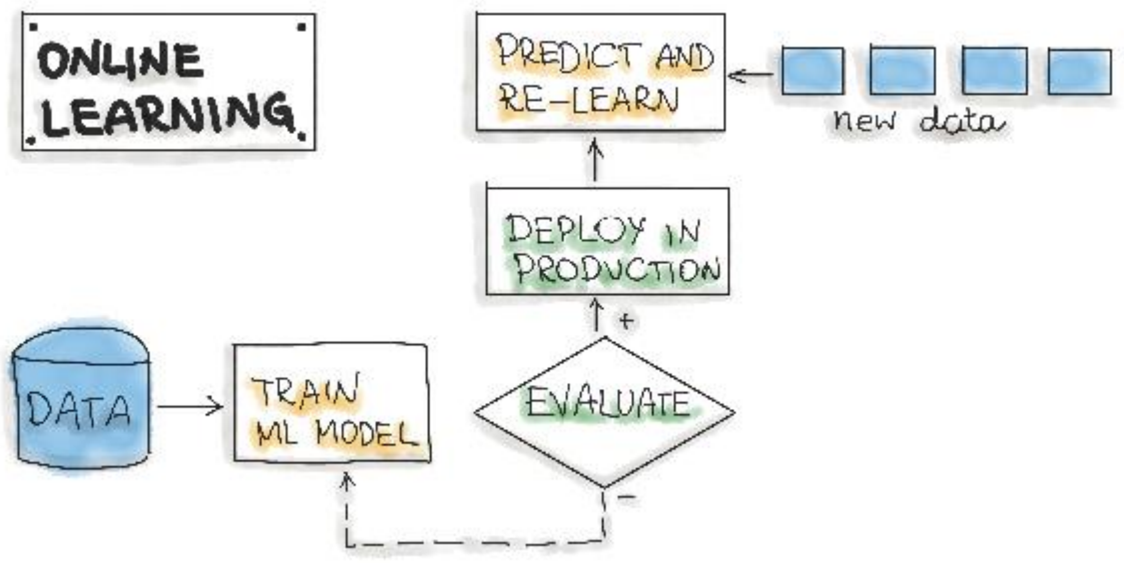
- **Supervised**
- **Unsupervised**
- **Semi-supervised**
- and **Reinforcement Learning**.

Model Training Patterns
Two ways to perform **ML Model Training**:

- **Offline learning**
- **Online learning**

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's re-trained. The **model** will see a lot of real-life data and becomes stale - 'model decay' and should be carefully monitored.

Online learning (aka dynamic learning):
The **model** is regularly re-trained as new data arrives, e.g., as data streams.
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**.



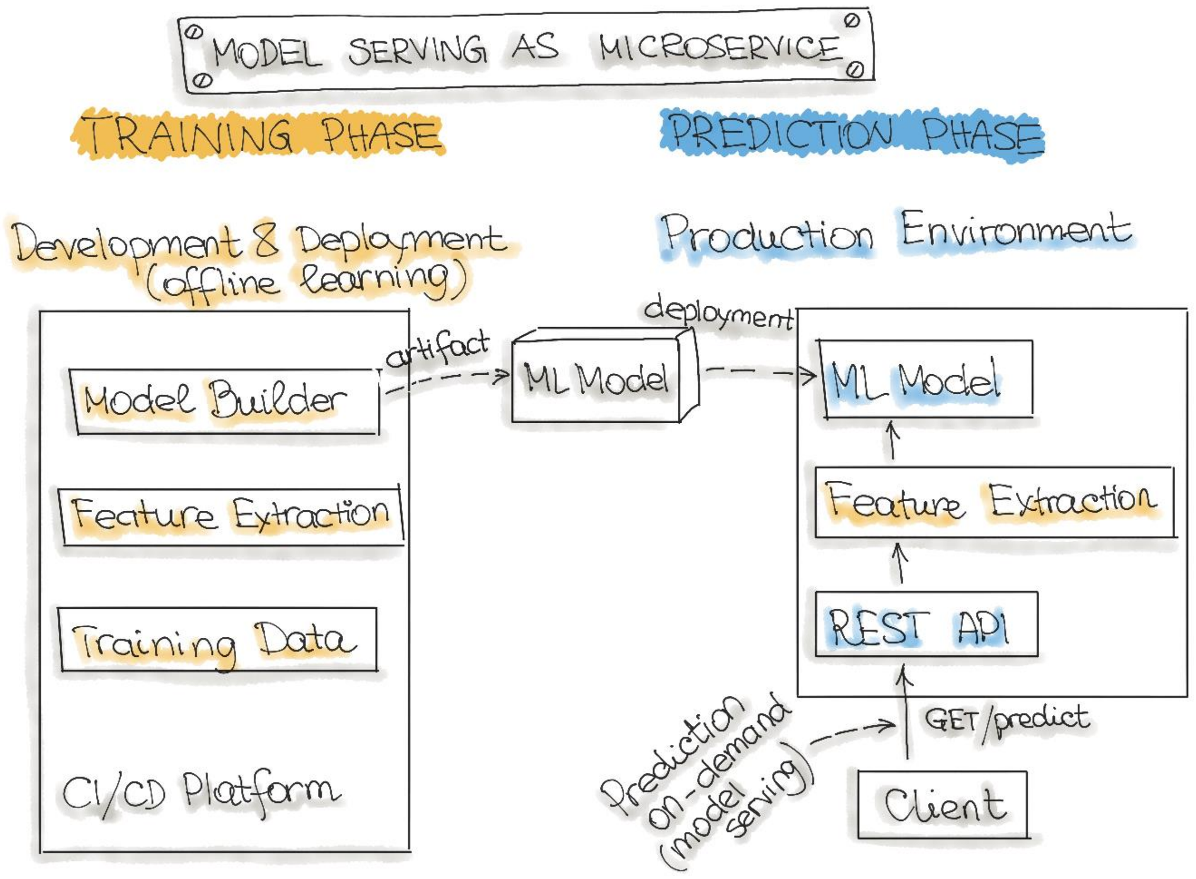
Model Prediction Patterns
Two modes to denote the mechanics of the **ML model** to make predictions:

- **Batch predictions**
- **Real-time predictions**

Batch predictions:
The deployed **ML model** makes a set of predictions based on historical input data. Often sufficient for data that is not time-dependent, or when it is not critical to obtain real-time predictions as output.

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.

ML Architecture Patterns



Forecast:
Widely spread in academic research or data science education (e.g., Kaggle or DataCamp). Used to experiment with **ML algorithms** and data as it is the easiest way to create a **machine learning system**.

Web-Service (or Microservices):
The most commonly described deployment architecture for **ML models**.
The **web service** takes input data and outputs a prediction for the input data points.

Online Learning (Real-time streaming analytics):
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.

AutoML:
AutoML is getting a lot of attention, considered the next advance for enterprise ML.
Promises training **ML models** with minimal effort and without **machine learning expertise**.

23. MODEL SERVING PATTERNS

Machine Learning Workflow: Code Deployment Pipelines
The final stage of delivering an ML project includes the following three steps:

- **Model Serving**
- **Model Performance Monitoring**
- **Model Performance Logging**

Model Serving
The process of deploying the ML model in a production environment.

Model Performance Monitoring
The process of observing the ML model performance based on live and previously unseen data, such as prediction or recommendation.

Model Performance Logging
Every inference request results in a log-record.

Model Serving Patterns
Three components should be considered when an ML model is served in a production environment:

1. **Inference:** The process of getting data to be ingested by a model to compute predictions, requiring a model, an interpreter for execution, and input data.

Deploying an ML system to a production environment includes two aspects:

1. Deploying the pipeline for automated retraining and ML model deployment.
2. Providing the API for prediction on unseen data.

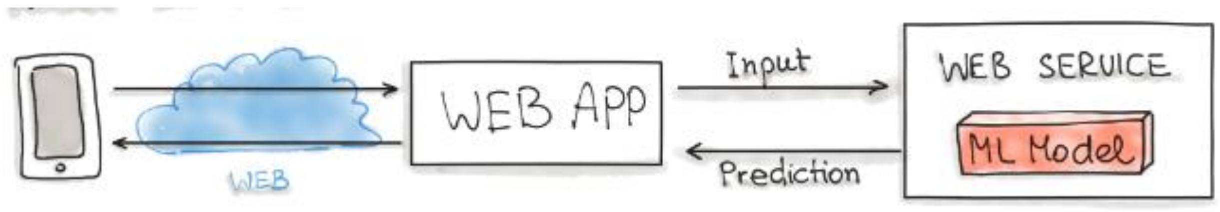
Model serving is a way to integrate the ML model into a software system.
Five patterns to put the ML model in production:

1. **Model-as-Service**
2. **Model-as-Dependency**
3. **Precompute**
4. **Model-on-Demand**
5. **Hybrid-Serving**

23.1 Machine Learning Model Serving Taxonomy

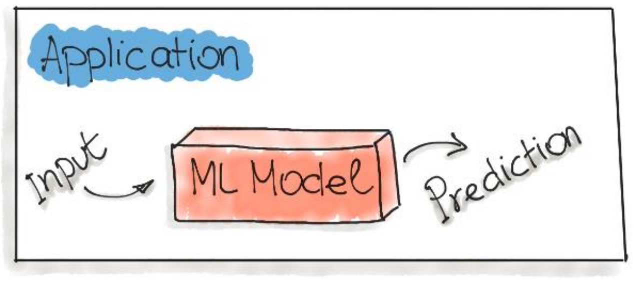
MODEL-AS-SERVICE

A common pattern for wrapping an ML model as an independent service, available through a REST API or consumed as a gRPC service.



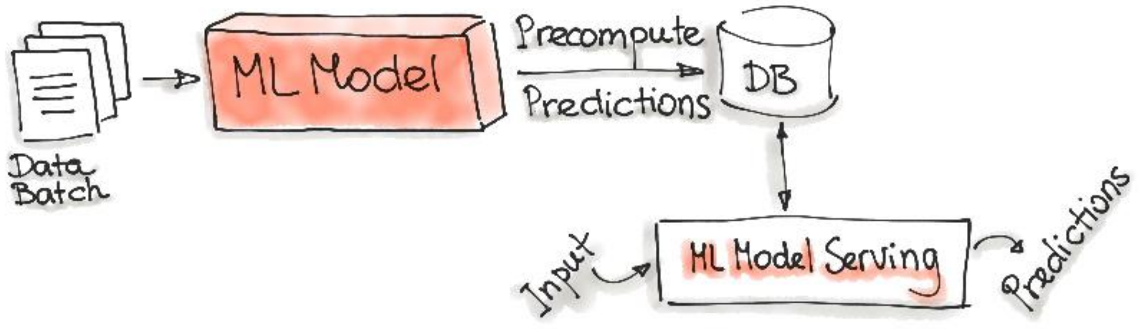
MODEL-AS-DEPENDENCY

Probably the most straightforward way to package an ML model, used for implementing the Forecast pattern.



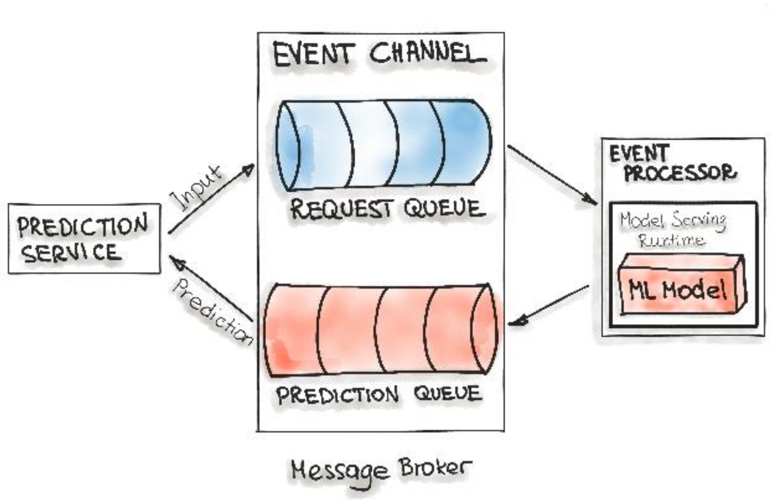
PRECOMPUTE SERVING PATTERN

Tightly related to the Forecast ML workflow, predictions are precomputed for incoming data batches and persisted in the database.



MODEL-ON-DEMAND

Treats the ML model as a dependency available at runtime, with its own release cycle and published independently. Utilizes a message-broker architecture.



HYBRID-SERVING (FEDERATED LEARNING)

Utilizes both a server-side model and user-side models, where the server model is updated periodically based on data from user-side models.

23.2 Deployment Strategies

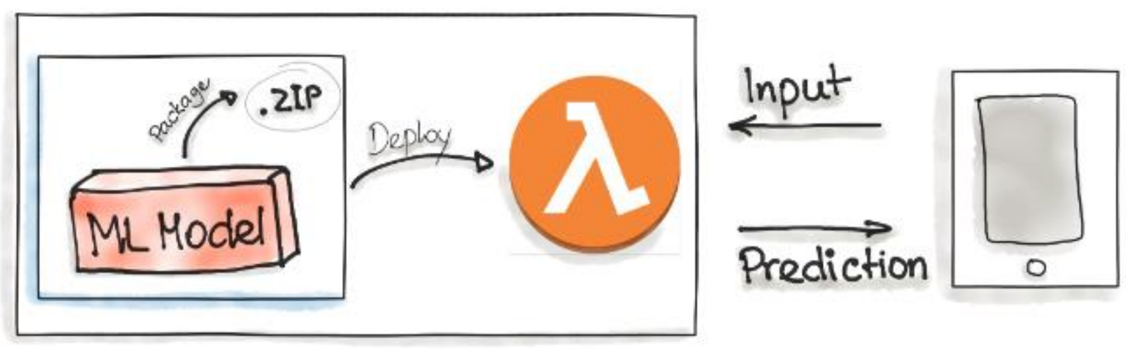
Common ways for wrapping trained models as deployable services:

DEPLOYING ML MODELS AS DOCKER CONTAINERS

Containerization becomes the de-facto standard for delivery, with Docker being the de-facto standard containerization technology.

DEPLOYING ML MODELS AS SERVERLESS FUNCTIONS

Various cloud vendors already provide machine-learning platforms, enabling deployment with their services. Attention should be paid to possible constraints of the deployed artifacts.



24. BATCH PREDICTION VERSUS ONLINE PREDICTION

Three main modes of prediction:

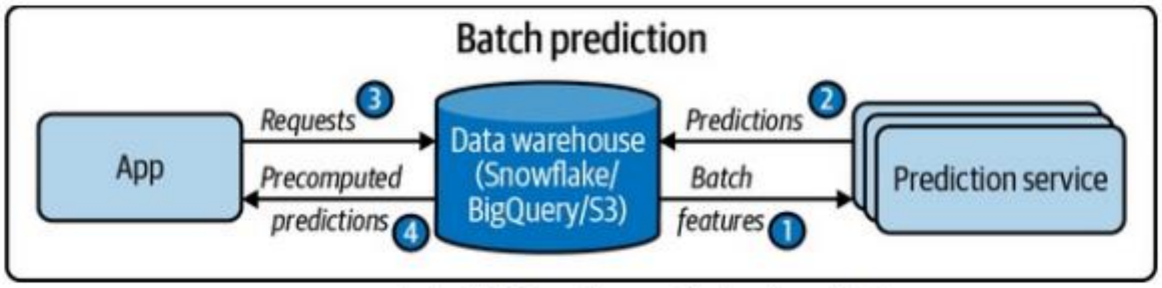
- **Batch prediction:** Uses only batch features.
- **Online prediction:** Uses only batch features (e.g., precomputed embeddings), also known as on-demand prediction.
- **Online prediction:** Uses both batch features and streaming features, also known as streaming prediction.

BATCH PREDICTION

Predictions are generated periodically or whenever triggered.
Predictions are stored and retrieved as needed, such as in SQL tables or an in-memory database.
Example:

- Netflix generates movie recommendations for all users every four hours.
- Precomputed recommendations are fetched and shown to users when they log on to Netflix.

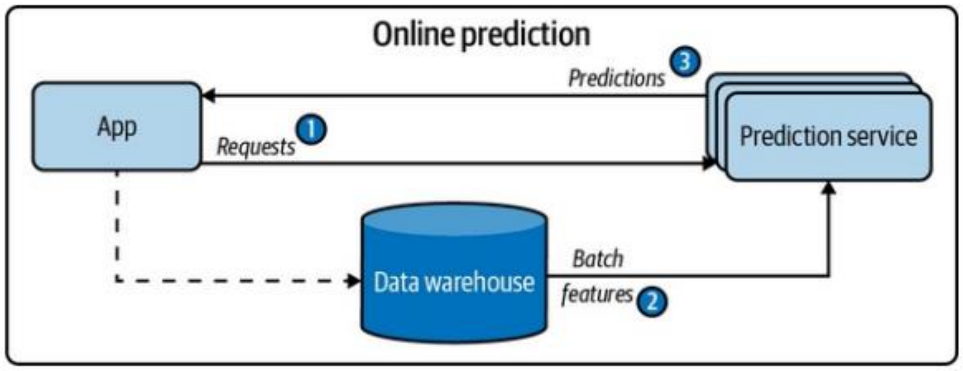
Also known as asynchronous prediction: predictions are generated asynchronously with requests.



A simplified architecture for batch prediction

ONLINE PREDICTION

Predictions are generated and returned as soon as requests arrive.
Example: Entering an English sentence into Google Translate and receiving its French translation immediately, also known as on-demand prediction.
Traditionally, requests are sent to the prediction service via RESTful APIs, also known as synchronous prediction: predictions are generated in synchronization with requests.



A simplified architecture for online prediction that uses only batch features

Using Only Batch Features
Online prediction:

- **Batch features:** Computed from historical data, such as data in databases and data warehouses (e.g., restaurant ratings).

Streaming Features
Features computed from streaming data — data in real-time transports (e.g., estimation for delivery time).
In batch prediction, only batch features are used. However, in online prediction, both batch and streaming features can be used.

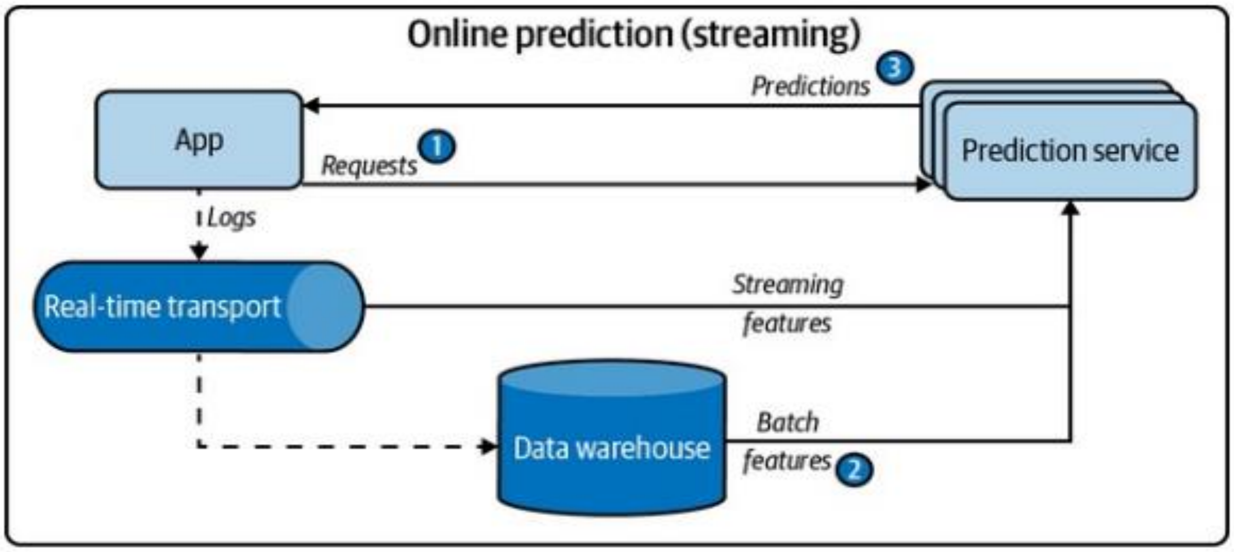
Example:
After a user places an order on DoorDash, features to estimate delivery time might include:

- The mean preparation time of this restaurant in the past.
- The number of orders in the last 10 minutes and the availability of delivery people.

Using both batch and streaming features is termed "streaming prediction."

Streaming Prediction

A simplified architecture for online prediction that uses both streaming features and batch features.



A simplified architecture for online prediction that uses both batch features and streaming features

Online vs Batch Prediction

Unifying Batch Pipeline and Streaming Pipeline.

25. CAUSES OF ML SYSTEM FAILURE

Occurs when one or more expectations of the system are violated.
In traditional software, focus is on operational expectations:

- System executes logic within expected operational metrics like latency and throughput.

For an ML system, attention is on both operational and ML performance metrics.
Example:

- English-French machine translation system:
    - Operational expectation: Translation within one-second latency.
    - ML performance expectation: Accurate translation 99% of the time.

25.1 ML system failure types

SOFTWARE SYSTEM FAILURES

Easier to detect, often accompanied by operational breakage:

- Dependency failure: Third-party software package breaks.
- Deployment failure: Errors in deployment, like using older model versions.
- Hardware failures: Issues with CPUs or GPUs.
- Downtime or crashing: Server failures lead to system downtime.

Addressing software system failures requires traditional software engineering skills.

ML-SPECIFIC FAILURES

Examples include:

- Data collection and processing issues.
- Poor hyperparameters.
- Mismatch between training and inference pipeline changes.
- Data distribution shifts affecting model performance.
- Edge cases causing catastrophic mistakes.
- Degenerate feedback loops influencing model iterations.

Production data differing from training data

Assumption:

- Model generalizes to unseen data from a stationary distribution similar to training data.

In reality:

- Real-world data distribution differs from training data due to complexity and evolution.
- Train-serving skew: Model performs well in development but poorly in deployment.

EDGE CASES

Extreme data samples causing catastrophic model failures.
Critical for safety-critical applications like self-driving cars and medical diagnosis.

Degenerate feedback loops

Feedback loop influences model iterations:

- Predictions influence feedback, affecting model updates.
- Common in recommender systems and ads click-through-rate prediction.
- Example: Song recommendation system creating bias loop based on user interactions.

Understanding and addressing these failures are crucial for deploying reliable ML systems.

26. MONITORING AND FEEDBACK LOOP

26.1 Model monitoring

When a machine learning model is deployed in production, it can degrade in **quality** fast and without warning.

Machine learning models must be monitored at two levels:

1. **Resource level**: Ensuring the model is running correctly in the production environment.
    - Is the system alive?
    - Are CPU, RAM, network usage, and disk space as expected?
    - Are requests being processed at the expected rate?
2. **Performance level**: Monitoring the pertinence of the model over time.
    - Is the model still an accurate representation of the pattern of new incoming data?
    - Is it performing as well as it did during the design phase?

Model monitoring is a crucial step in the ML model life cycle and a critical piece of **MLOps**.

Model performance monitoring
The first level is a traditional **DevOps** topic.
The latter is more complicated because how well a model performs is a reflection of the data used to train it, particularly how representative that training data is of the live request data.
Model performance monitoring attempts to track this degradation. At an appropriate time, it will also trigger the retraining of the model with more representative data.

Model Retraining
Critical for organizations to have a clear idea of deployed models' drift and accuracy by setting up a process that allows for easy monitoring and notifications.
An ideal scenario would be a pipeline that automatically triggers checks for degradation of model performance. The goal of notifications is not necessarily to kick off an automated process of retraining, validation, and deployment, but to alert the data scientist of the change; to diagnose the issue and evaluate the next course of action.

It's critical that as part of MLOps and the ML model life cycle, data scientists and their managers and the organization as a whole understand model degradation.
Every deployed model should come with monitoring metrics and corresponding warning thresholds to detect meaningful business performance drops as quickly as possible.

26.2 Monitoring and Feedback Loop

Model Degradation
Two approaches to monitor performance degradation:

1. **Ground Truth Evaluation**
2. **Input Drift Detection**

Ground Truth Evaluation
Ground truth retraining requires waiting for the label event. It's crucial for some types of models, but teams may need to wait months for ground truth labels to be available, which can mean significant economic loss if the model is degrading quickly.
Ground truth and prediction are decoupled. In many production environments, matching ground truth with corresponding observations is challenging because this information is generated and stored in different systems and at different timestamps.

Input Drift
Mathematically, samples of each dataset cannot be assumed to be drawn from the same distribution. If train the algorithm on the first dataset and then deploy it on the second one, the resulting distribution shift is called a drift.

Drift Detection in Practice
To react in a timely manner, model behavior should be monitored based on the feature values of the incoming data, without waiting for the ground truth to be available.

Causes of Data Drift

1. **Sample selection bias**
2. **Non-stationary environment**

Input Drift Detection Techniques
Two approaches: **Univariate statistical tests** and **Domain classifier**.
Prefer univariate statistical tests for proven and explainable methods.

CONTINUOUS DELIVERY FOR END-TO-END MACHINE LEARNING PROCESS

The Feedback Loop
All effective machine learning projects implement a form of data feedback loop, where information from the production environment flows back to the model prototyping environment for further improvement.
In practice, data collected in the monitoring and feedback loop is sent to the model development phase. From there, the system analyzes whether the model is working as expected. If not, an update is triggered, either automatically or manually by the data scientist.

Logging
An effective logging system is crucial for monitoring a live system, as it collects and aggregates data about its states, enabling continuous improvement of the ML system.

Model Evaluation
If model performance is degrading, data scientists decide to improve the model by retraining it. The main tasks of a model evaluation store are versioning the evolution of a logical model through time and comparing the performance between different versions of a logical model.

Online Evaluation
Critical from a business perspective but challenging technically. Two main modes: **Champion/challenger (shadow testing)** and **A/B testing**. Shadow testing is preferable whenever possible because it's simpler to understand and set up, and it detects differences more quickly.