



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Three Levels of ML Software

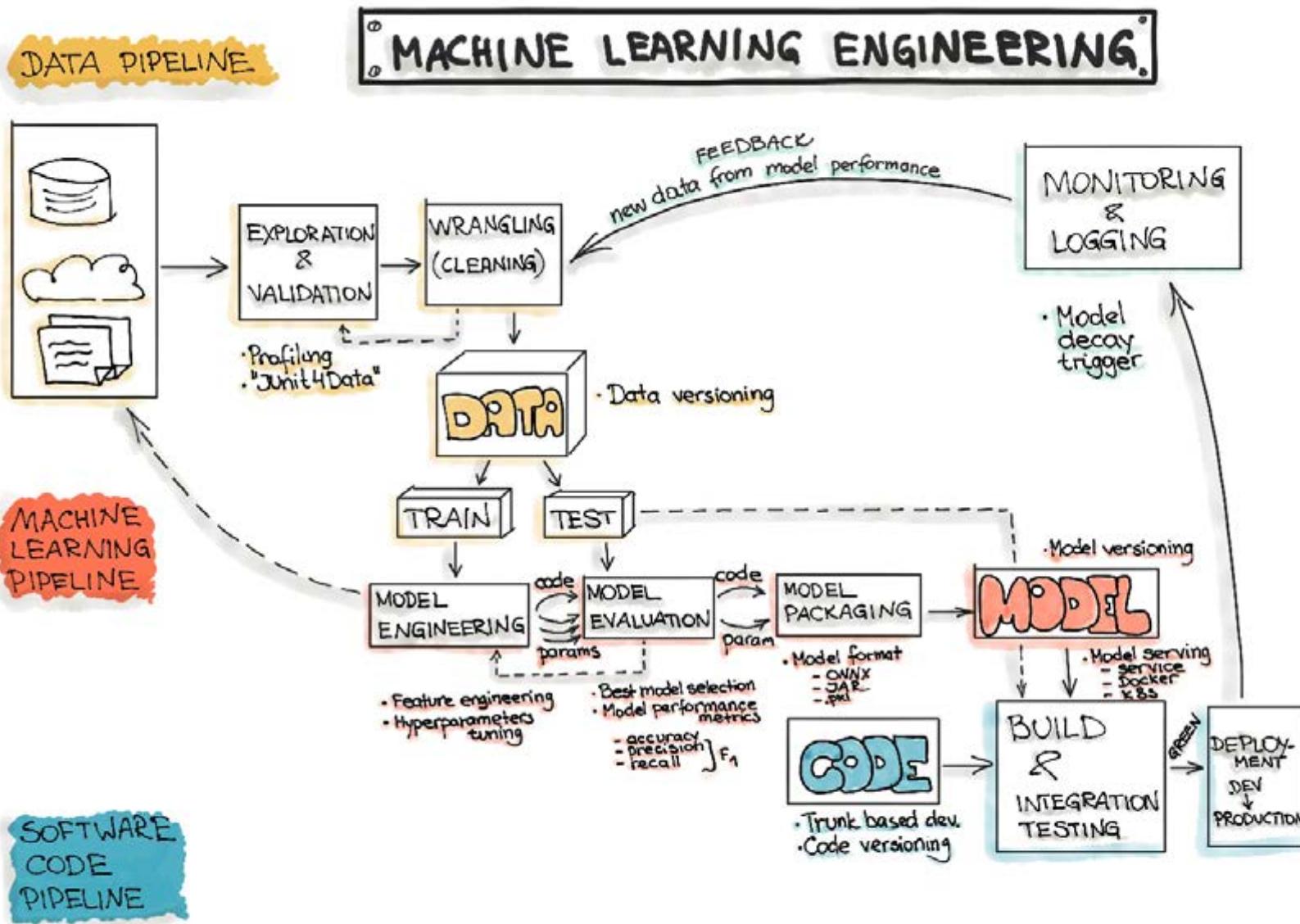
Pravin Y Pawar

Extracted from
An Overview of the End-to-End Machine Learning Workflow

Three levels of ML Software

- ML/AI is rapidly adopted by new applications and industries!
- Goal of a machine learning project is to build a statistical model by using collected data and applying machine learning algorithms
 - Yet building successful ML-based software projects is still difficult
- Every ML-based software needs to manage three main assets:
 - Data
 - Model
 - and Code
- 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.

Machine Learning Engineering



Data Engineering

- The initial step in any data science workflow is to **acquire and prepare the data to be analyzed**
 - Typically, data is being integrated from **various resources** and has **different formats**
- Data Preparation
 - according to Gartner “**an iterative and agile process for exploring, combining, cleaning and transforming raw data into curated datasets for data integration, data science, data discovery and analytics/business intelligence (BI) use cases**”.
 - is reported to be the most expensive with respect to resources and time
- Data preparation is a critical activity in the data science workflow
 - important to avoid the propagation of data errors to the next phase, data analysis
 - would result in the derivation of wrong insights from the data

Data Engineering(2)

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

- Data Ingestion
 - Collecting data by using various frameworks and formats, such as Spark, HDFS, CSV, etc
 - Might also include synthetic data generation or data enrichment.
- Data Exploration
 - Includes data profiling to obtain information about the content and structure of the data
 - The output of this step is a set of metadata, such as max, min, avg of values
- Data Validation
 - Operations are user-defined error detection functions, which scan the dataset in order to spot some errors
- Data Wrangling (Cleaning)
 - The process of re-formatting particular attributes and correcting errors in data, such as missing values imputation
- Data Labeling
 - The operation of the Data Engineering pipeline, where each data point is assigned to a specific category
- Data Splitting
 - Splitting the data into training, validation, and test datasets to be used during the core machine learning stages to produce the ML model

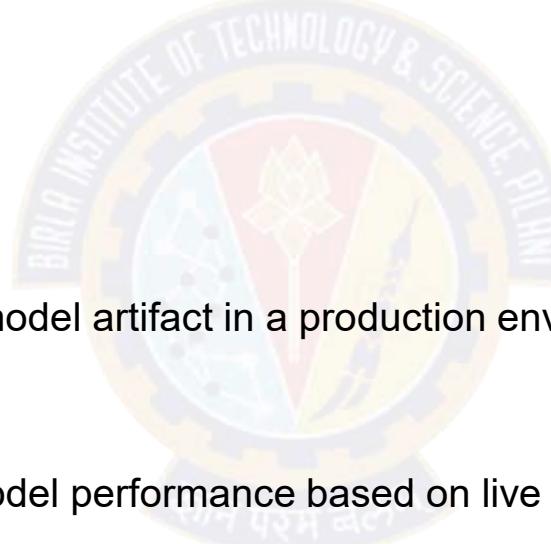
Model Engineering

The core of the ML workflow

- 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:
 - Model Training
 - The process of applying the machine learning algorithm on training data to train an ML model
 - includes feature engineering and the hyperparameter tuning for the model training activity
 - Model Evaluation
 - Validating the trained model to ensure it meets original codified objectives before serving the ML model in production to the end-user
 - Model Testing
 - Performing the final “Model Acceptance Test” by using the hold backtest dataset
 - 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

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





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

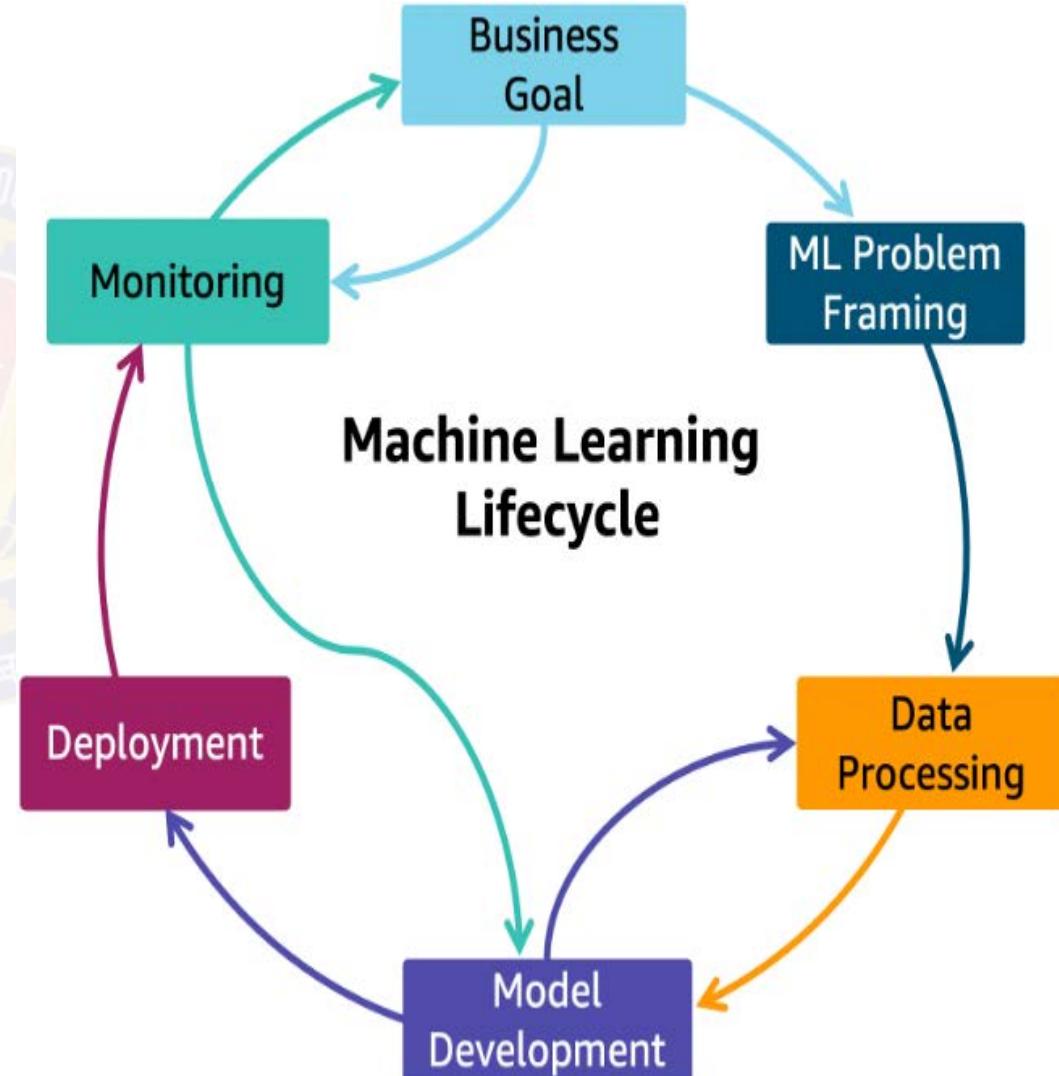
Machine Learning Lifecycle

Pravin Y Pawar

Extracted from AWS Well-Architected machine learning

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



Machine Learning Lifecycle(2)

Phases

- 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
- 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
- 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
- Model development
 - consists of model building, training, tuning, and evaluation
 - includes creating a CI/CD pipeline that automates the build, train and release to staging and production environments
- Deployment
 - After a model is trained, tuned, evaluated and validated, one can deploy the model into production
 - can then make predictions and inferences against the model
- Monitoring
 - ensures model is maintaining a desired level of performance through early detection and mitigation

ML lifecycle phase — Business goal

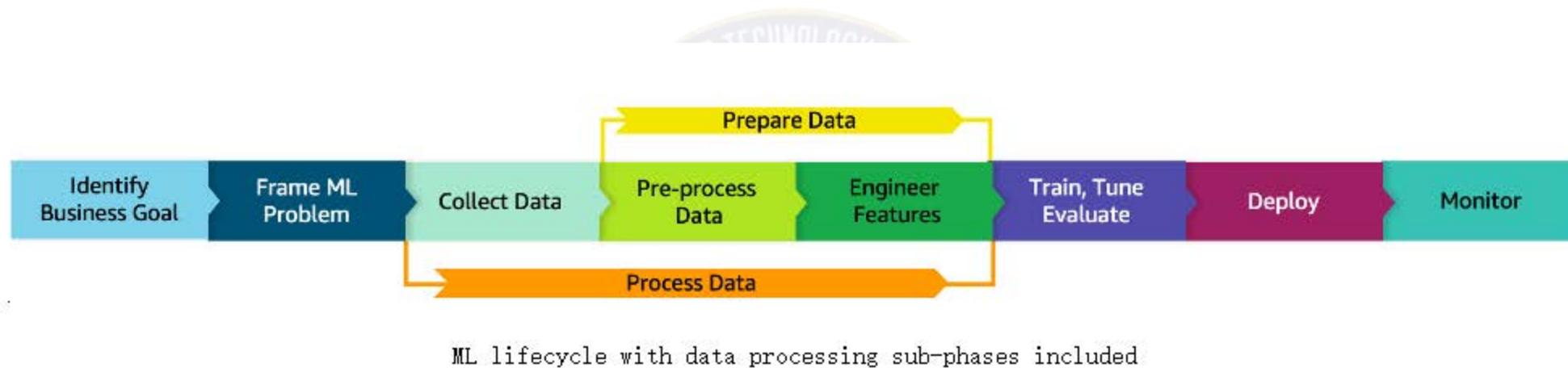
- The most important phase particularly challenging when considering ML solutions because ML is a constantly evolving technology!
- After success criteria is defined, evaluate organization's ability to move toward that target
 - The target should be **achievable** and provide a clear path to production
 - Involve
 - all relevant stakeholders from the beginning to align them to this target
 - 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

ML lifecycle phase — ML problem framing

- 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 must be optimized is a key step in ML
- 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

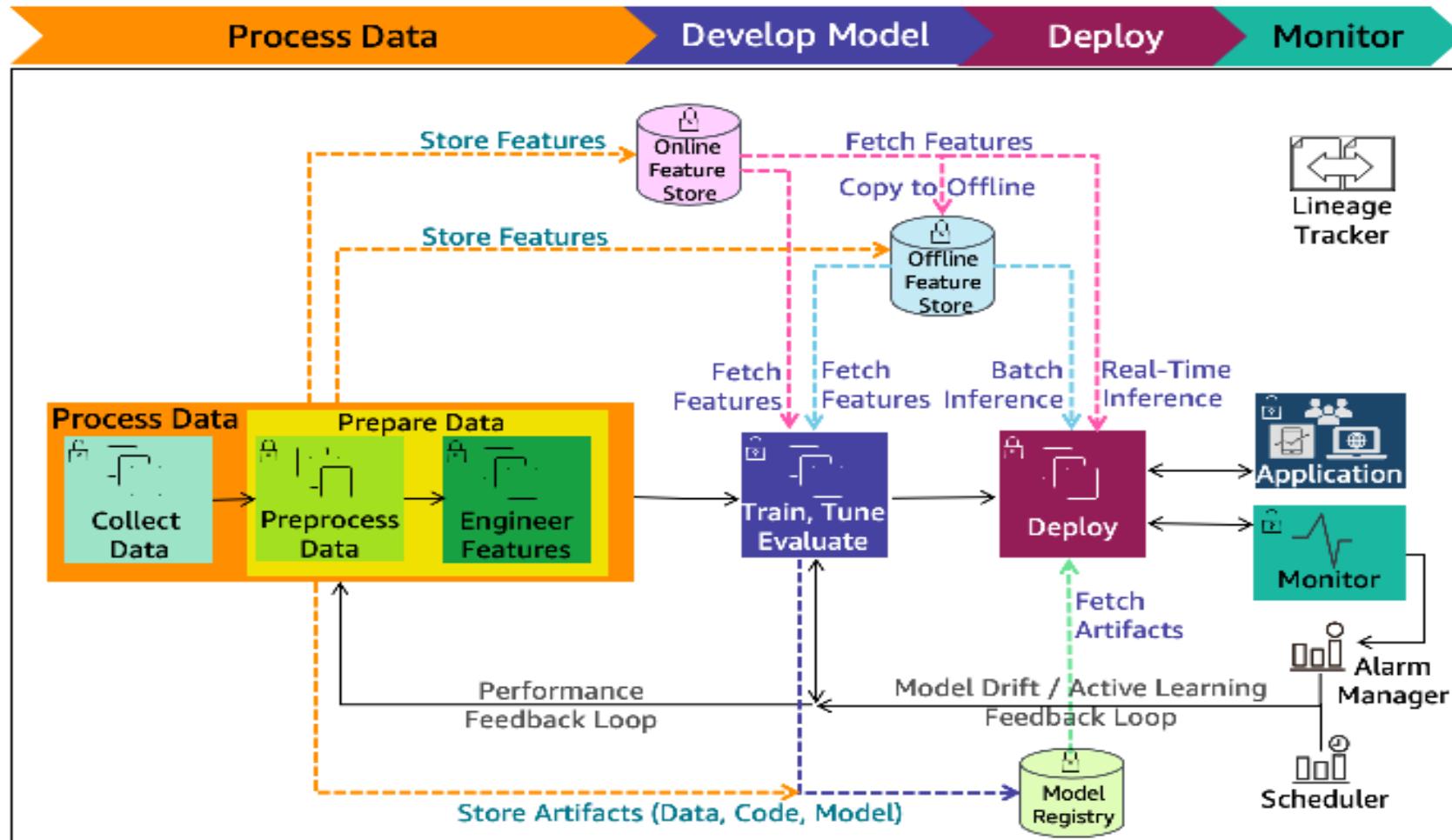
ML lifecycle architecture diagram

ML lifecycle with data processing sub-phases included



ML lifecycle architecture diagram(2)

ML lifecycle with detailed phases and expanded components



ML lifecycle with detailed phases and expanded components

ML lifecycle architecture diagram(3)

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

ML lifecycle architecture diagram(4)

Components

- 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

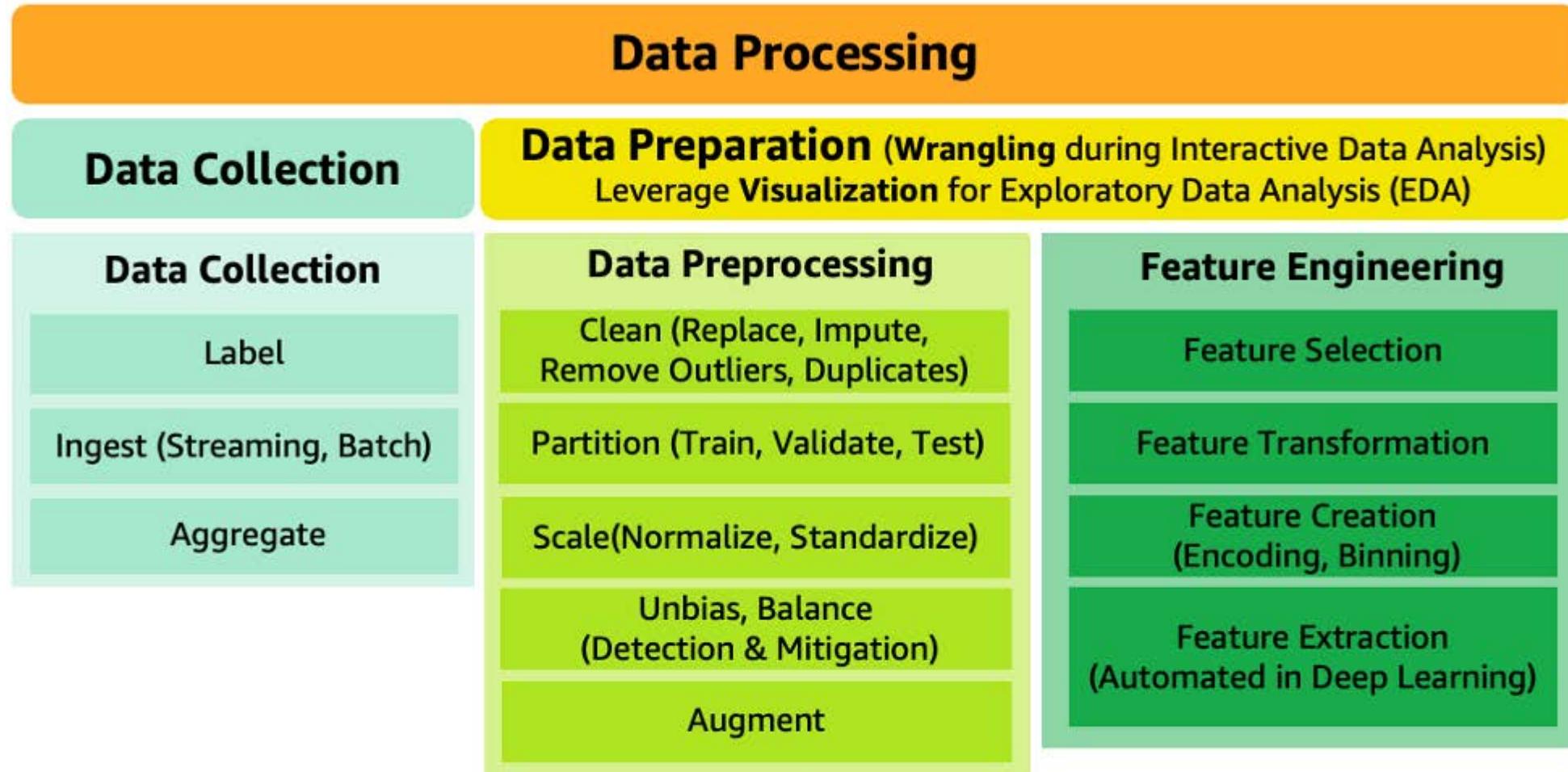
ML lifecycle phase - Data processing

Need

- 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 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.

ML lifecycle phase - Data processing(2)

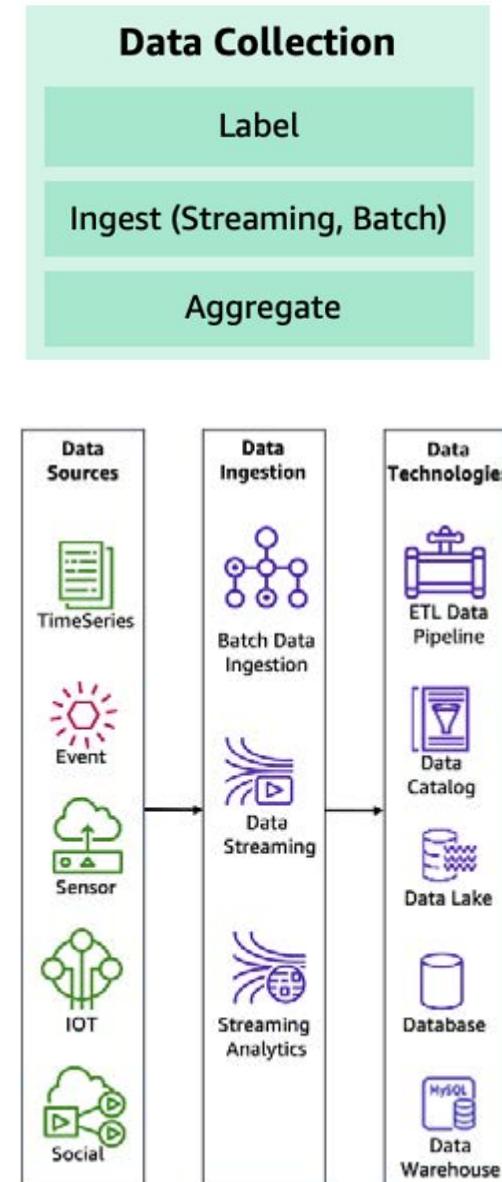
Data processing components



ML lifecycle phase - Data processing(3)

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 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 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



ML lifecycle phase - Data processing(4)

Data preparation

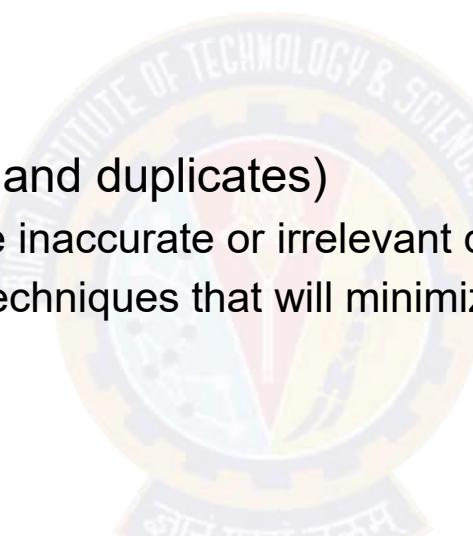
- ML models are only as good as the data that is used to train them
 - Ensure that suitable training data is available and is optimized for learning and generalization
- Data preparation includes
 - data preprocessing
 - feature engineering
- A key aspect to understanding data is to identify patterns 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 reduces the cost for interactive analysis



ML lifecycle phase - Data processing(5)

Data preprocessing

- Data preprocessing puts data into the right shape and quality for training
 - Many strategies: data cleaning, balancing, replacing, imputing, partitioning, scaling, augmenting and unbiasing
- 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



Data Preprocessing

Clean (Replace, Impute, Remove Outliers, Duplicates)

Partition (Train, Validate, Test)

Scale(Normalize, Standardize)

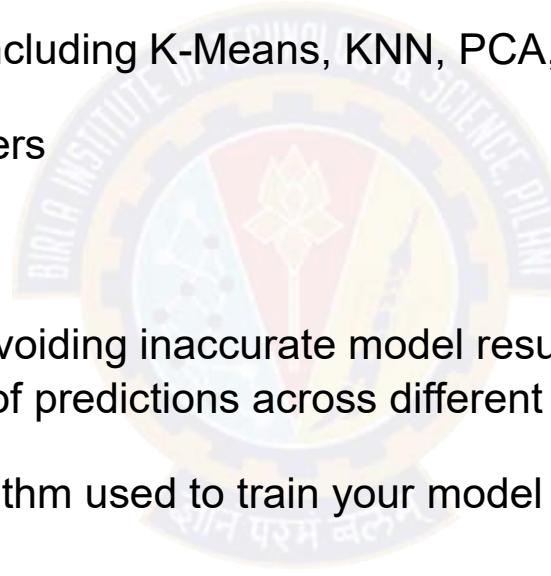
Unbias, Balance
(Detection & Mitigation)

Augment

ML lifecycle phase - Data processing(6)

Data preprocessing

- 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



Data Preprocessing

Clean (Replace, Impute, Remove Outliers, Duplicates)

Partition (Train, Validate, Test)

Scale(Normalize, Standardize)

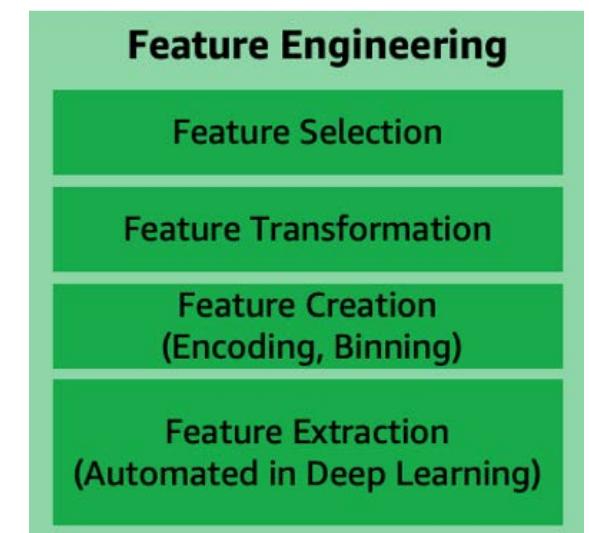
Unbias, Balance
(Detection & Mitigation)

Augment

ML lifecycle phase - Data processing(7)

Feature engineering

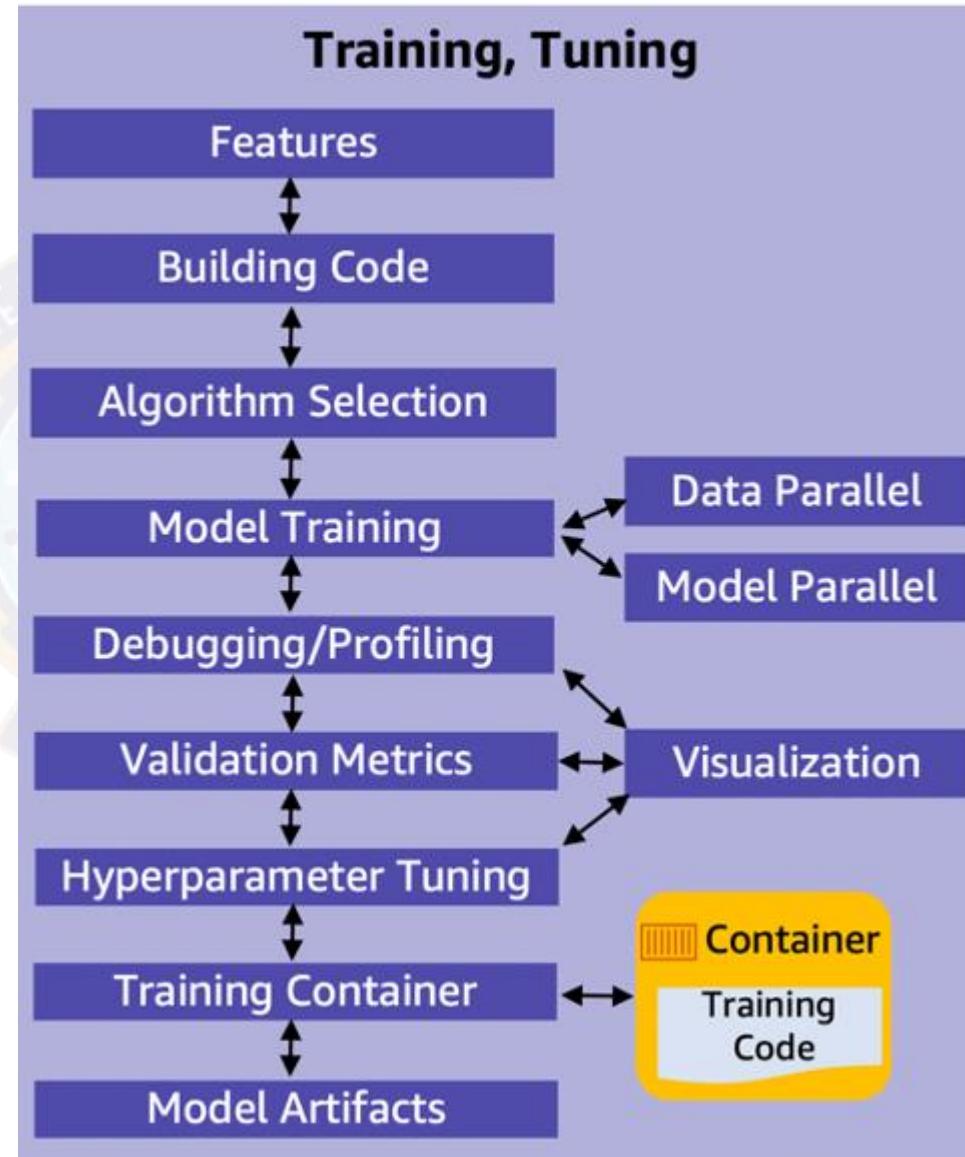
- Every unique attribute of the data is considered a feature
- Feature engineering is a process to select and transform variables when creating a predictive model using machine learning or statistical modeling
 - includes feature creation, feature transformation, feature extraction, and feature selection
 - With deep learning, the feature engineering is automated as part of the algorithm learning
- Feature 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 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



ML lifecycle phase – Model development

Model building, training, tuning

- In this phase, select a machine learning algorithm that is appropriate for 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
 - and set the hyperparameters to optimize the training process



ML lifecycle phase – Model development(2)

Model training, tuning activities

- Model training, tuning, and evaluation require prepared data and engineered features
- Features 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 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 in mini-batches evenly distributed across nodes
 - each node only trains the model on a fraction of the total dataset

ML lifecycle phase – Model development(3)

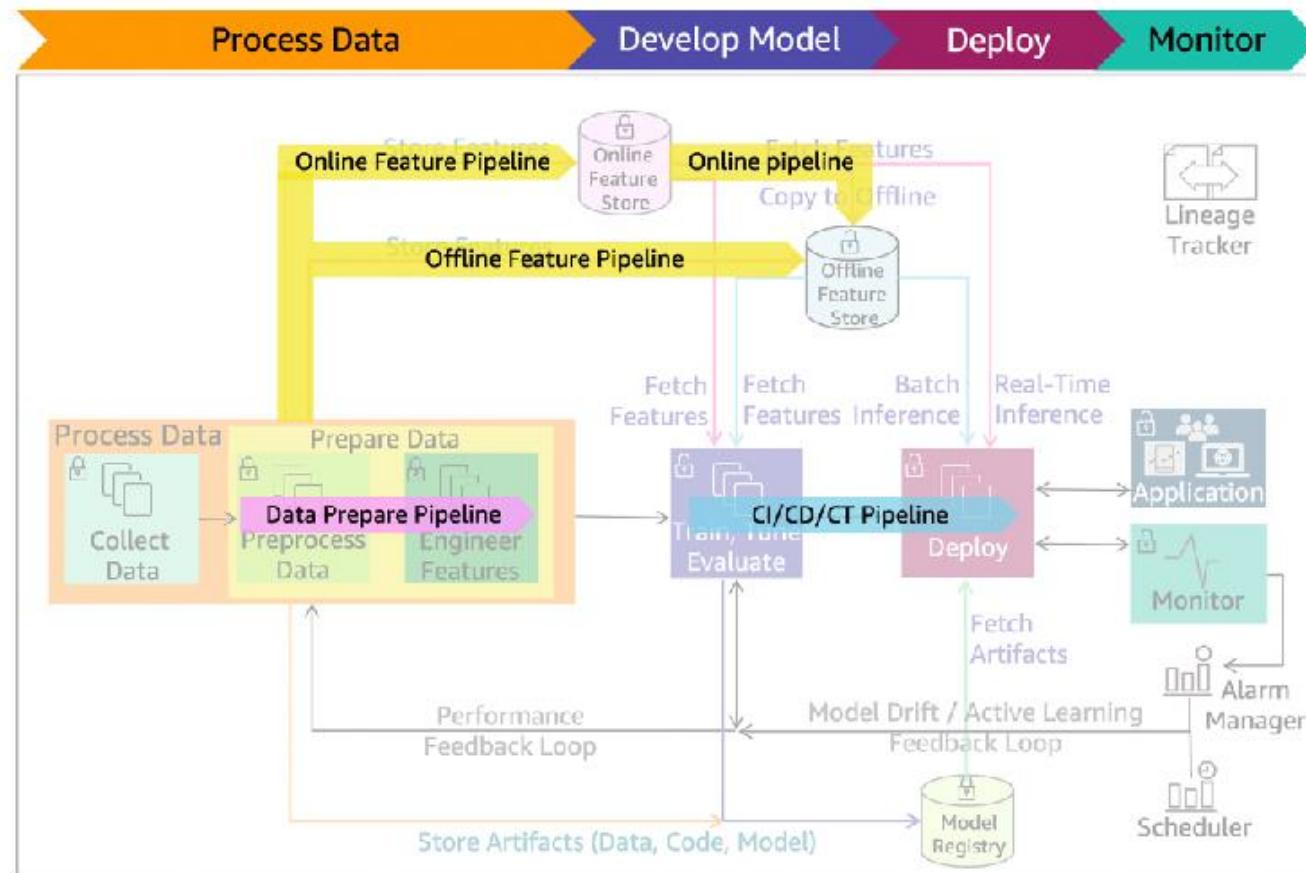
Model training, tuning activities

- 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 phase – Model development(4)

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



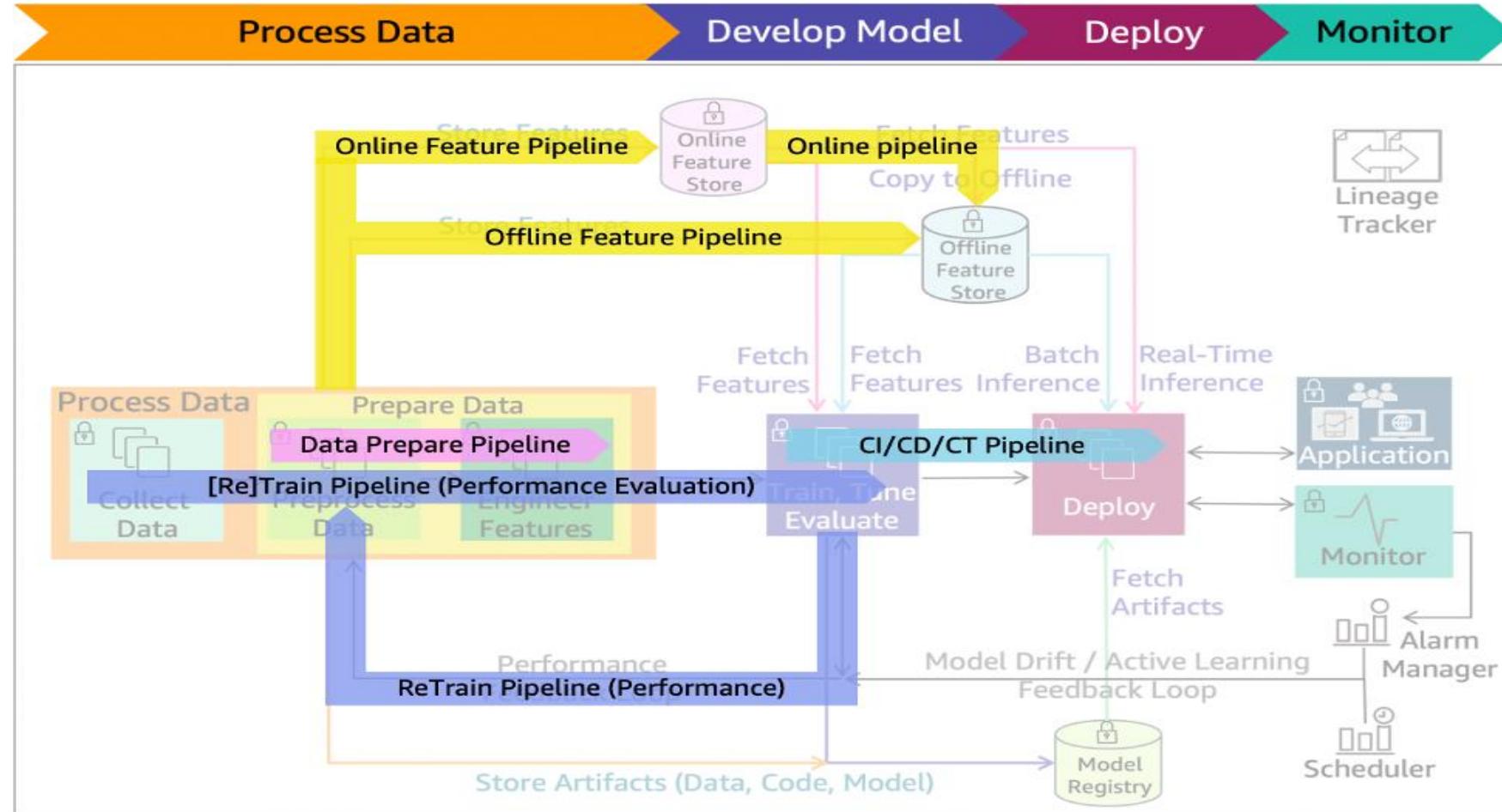
ML lifecycle phase – Model development(5)

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

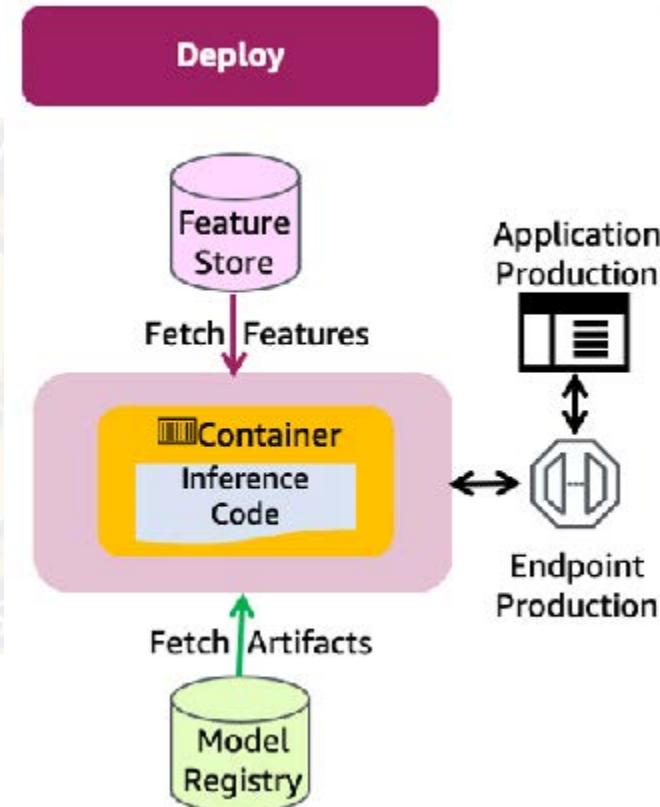
ML lifecycle phase – Model development(6)

ML lifecycle with performance evaluation pipeline added



ML lifecycle phase - Deployment

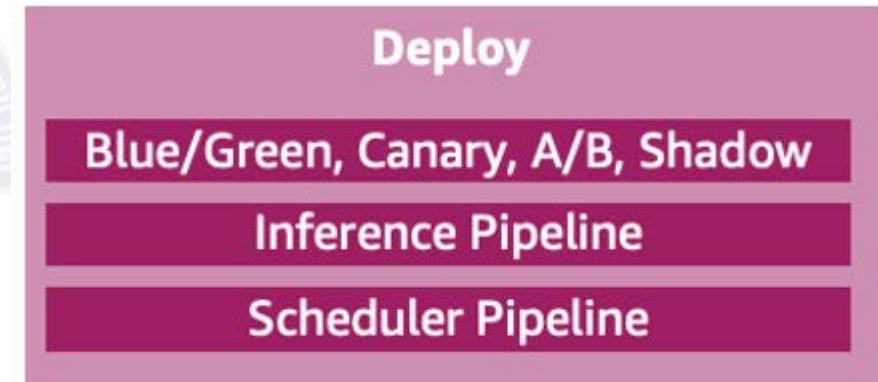
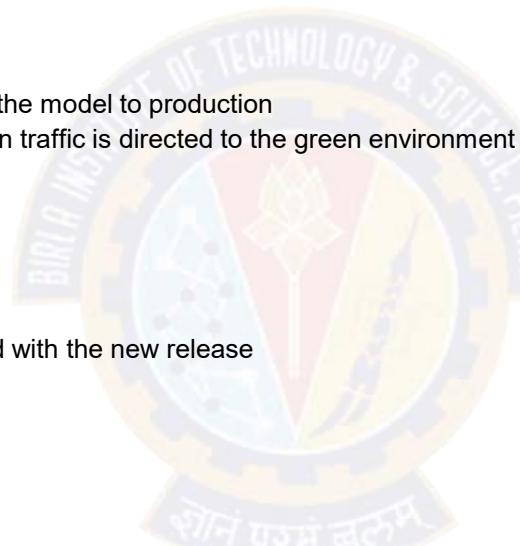
- After model have been trained, tuned, and evaluated, then can deploy the model into production
 - can make predictions and inferences against the deployed model
 - Use a manual governance process - ensures the model has fully been tested and evaluated before it's released to production
- Deploy production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



ML lifecycle phase – Deployment(2)

Deployment main components

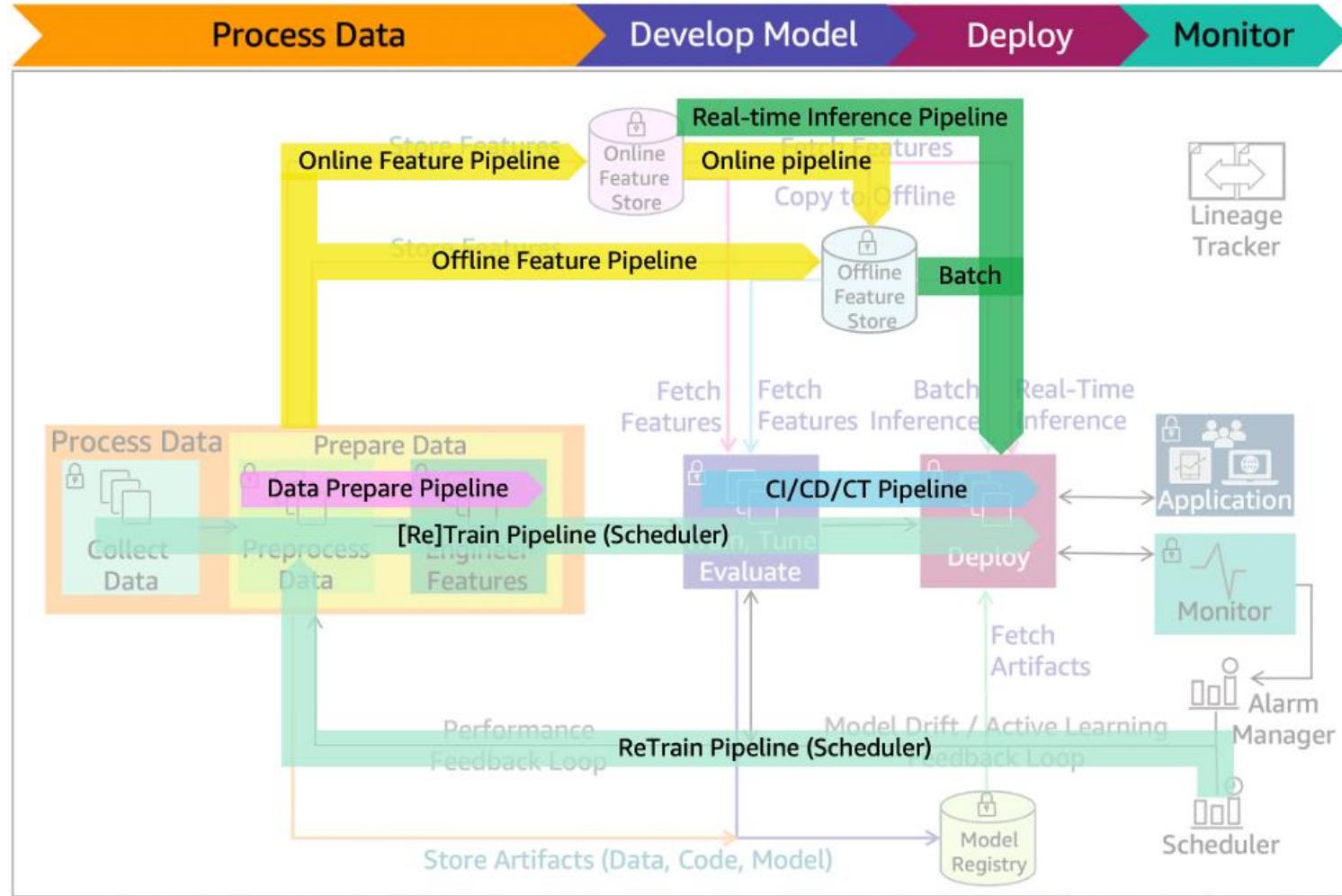
- Blue/green, canary, A/B, shadow deployment/testing
 - Deployment and testing strategies that reduce downtime and risks when releasing a new or updated version
- The blue/green deployment technique
 - provides two identical production environments
 - can use this technique when need to deploy a new 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 deploy 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
- 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
- Shadow deployment strategy
 - New version is 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 phase – Deployment(3)

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
 - 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



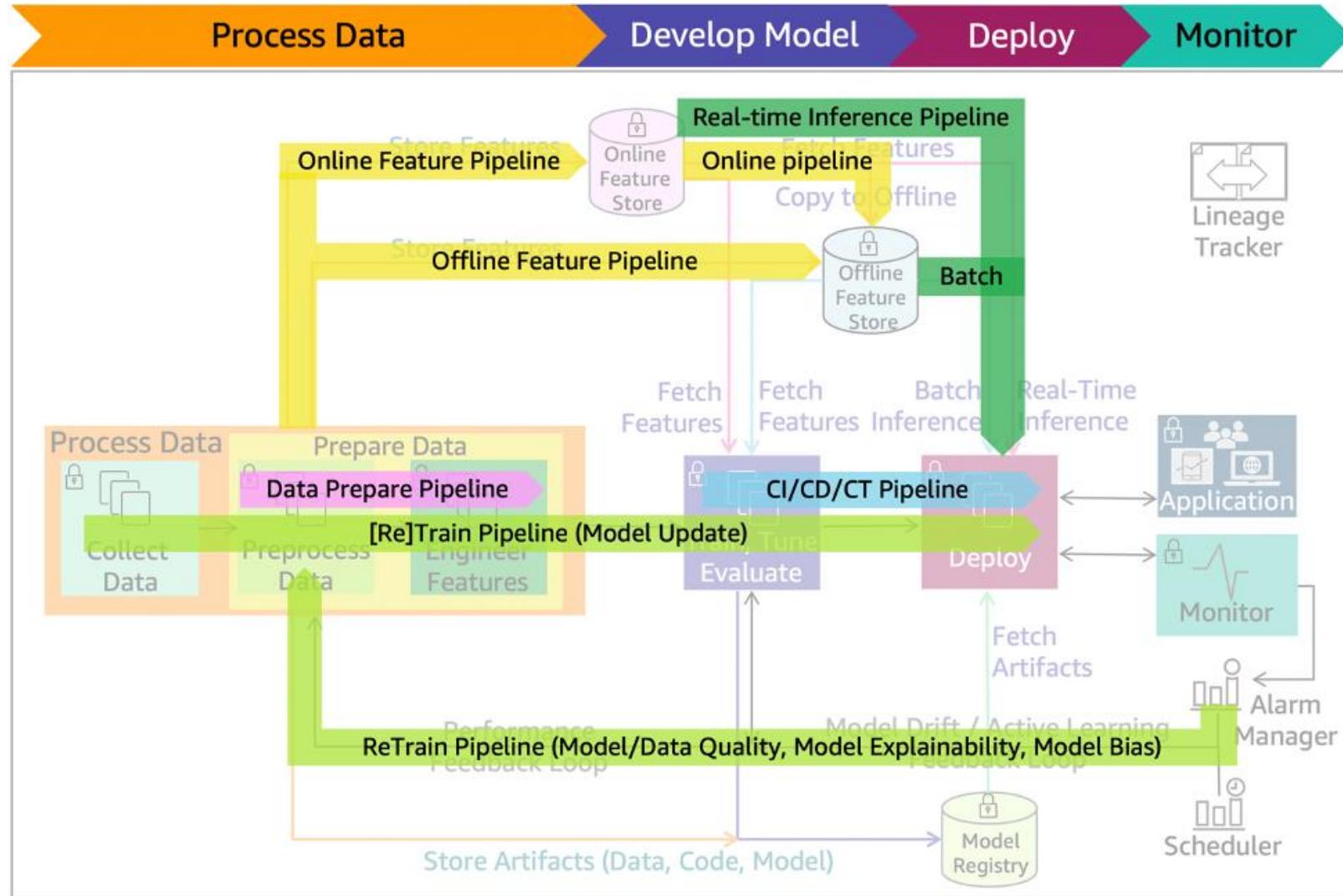
ML lifecycle phase – Monitoring

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

ML lifecycle phase – Monitoring(2)

ML lifecycle with model update re-train and batch/real-time Inference pipelines

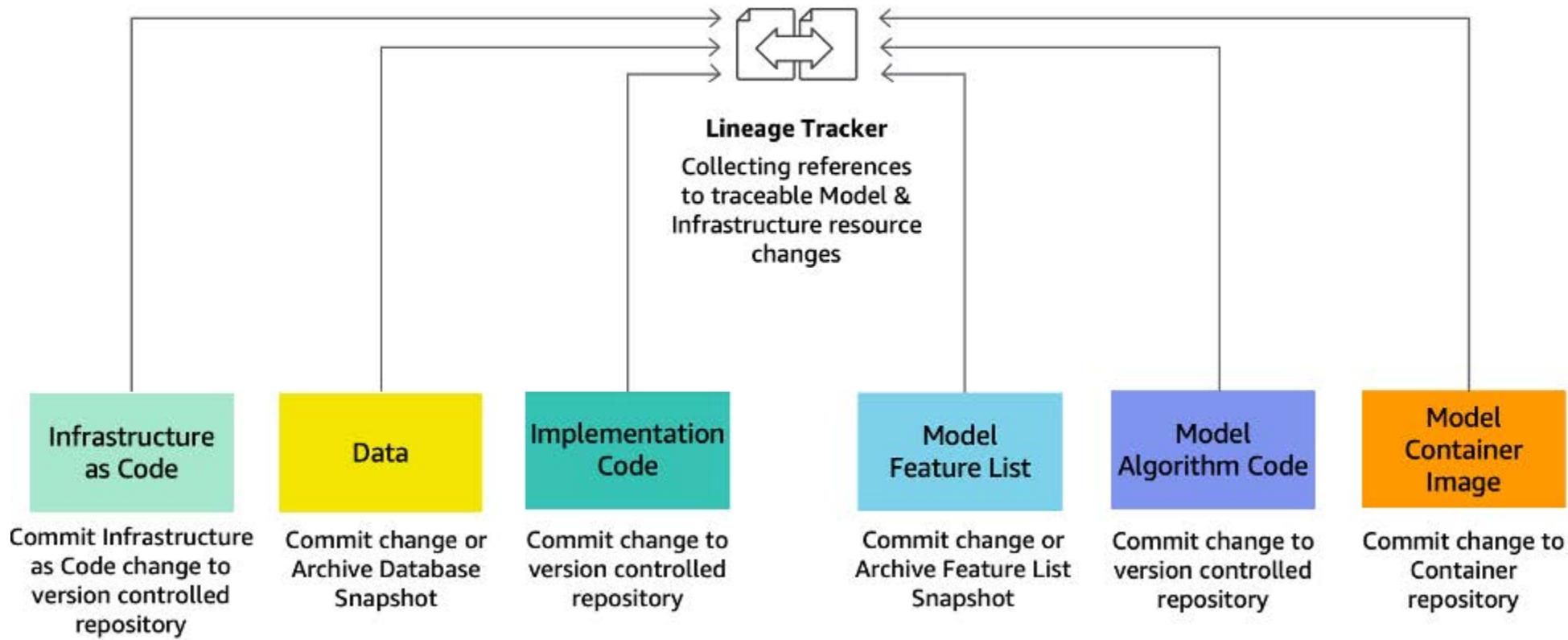


Additional - Lineage tracker

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

Lineage tracker



Lineage tracker

Components

- 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



Thank You!

In our next session:



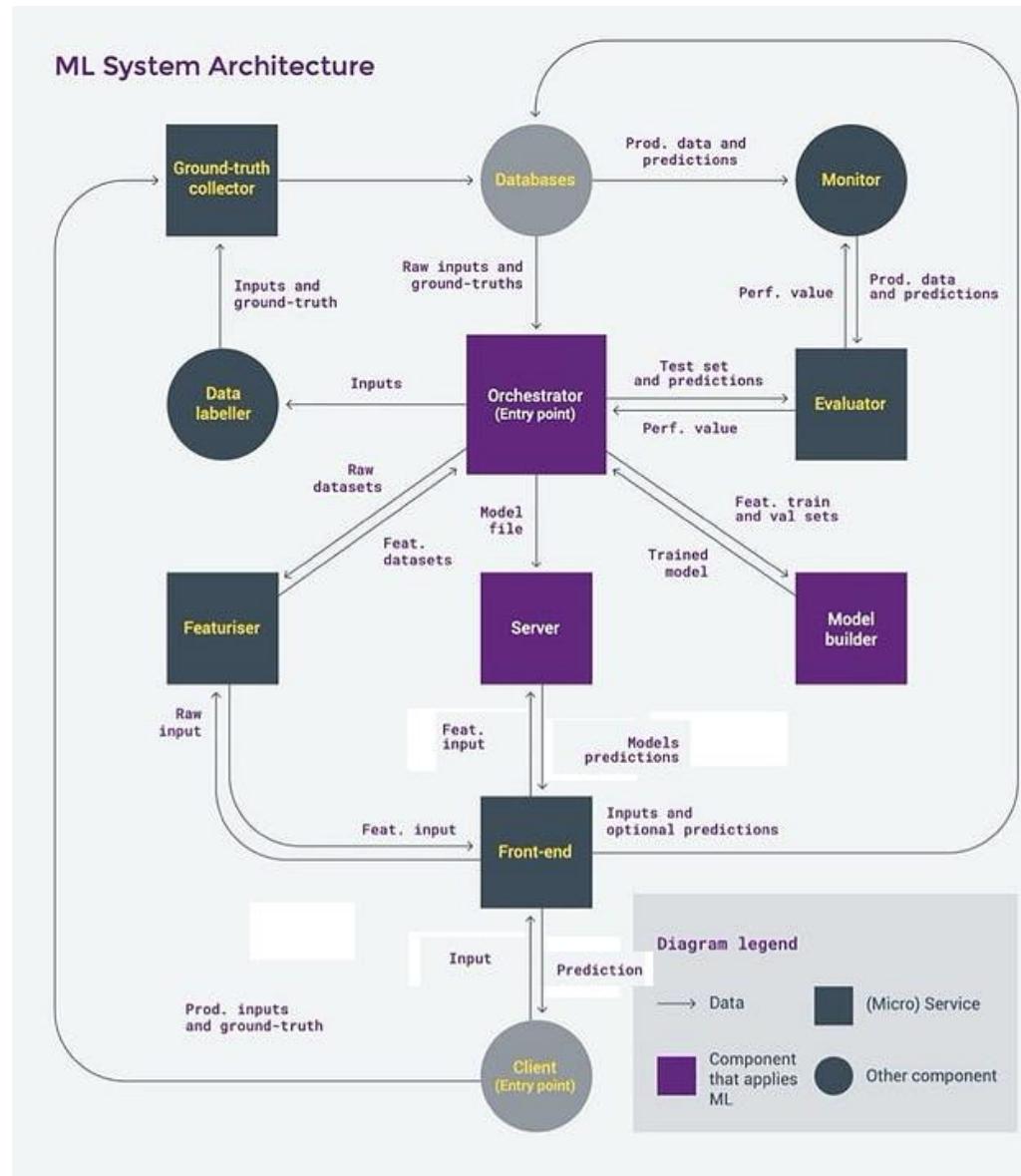
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML System Architecture

Pravin Y Pawar

Adapted from "[Architecture of a real-world Machine Learning system](#)" by Louis Dorard

ML System Architecture



Client-server architecture of a “supervised learning” system

'entry points'

the client requesting predictions, and the orchestrator creating/updating models

- Client represents the application
 - used by the end-user who will benefit from the ML system
 - can be the smartphone app used to order dinner
- Orchestrator would usually be a program called by a scheduler
 - so models could be updated periodically, e.g., every week
 - or called via an API (so that it could be part of a CI/CD pipeline)
 - In charge of evaluating models made by the model builder, on a test dataset that is kept secret
 - sends test predictions to the evaluator
- If a model is deemed good enough,
 - it is passed to the model server, which makes it available via an API
 - API can be directly exposed to the client software,
 - but domain-specific logic is often needed and implemented in a front-end



Monitoring

Monitor

- Assuming one or several (baseline) models would be available as APIs,
 - but would not be integrated into the final application yet,
 - need decide which model to integrate (and whether it's safe) by tracking performance on production data and visualizing it via a monitor
- In dinner-delivery example,
 - would let you compare a model's ETD with the actual time of delivery, on orders that were just delivered
- When a new model version becomes available,
 - client requests for predictions would be progressively directed to the new model's API, via the front-end
 - would be done for an increasing number of end-users while monitoring performance and checking that the new model is not "breaking" anything
- The owner of the ML system and the owner of the client application would be accessing the monitor's dashboard on a regular basis.

All the components

- Ground-truth Collector
- Data Labeler
- Evaluator
- Performance Monitor
- Featurizer
- Orchestrator
- Model Builder
- Model Server
- Front-end



#1: GROUND-TRUTH COLLECTOR

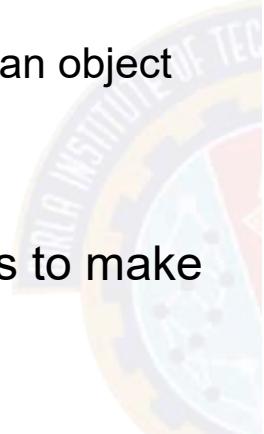
In the real world, it is key to be able to continuously acquire new data for the machine to learn from

- Ground-truth data
 - corresponds to what you want your ML models to predict, such as
 - the sale price of real-estate property,
 - a customer-related event (e.g. churn),
 - or a label to assign to input objects (e.g. ‘spam’ on incoming messages)
- Sometimes, you observe an input object, need to wait for a certain amount of time to observe the thing you wanted to predict about this object
 - wait for the property to get sold,
 - wait for the customer to renew or cancel their subscription,
 - wait for the user to interact with emails in their inbox
- May want the user to let you know when your ML system got the prediction wrong
 - If you want to give your user the ability to provide that kind of feedback, you’ll need a micro-service to send it to.



#2: DATA LABELLER

- Sometimes, have access to plenty of input data, but need to create the associated ground-truth data manually
 - case when building a spam detector, or an object detector from images
- Ready-made and open-source web apps to make data labeling easier
 - such as Label Studio
- Dedicated services for outsourcing the manual task of labeling data
 - for example, Figure Eight and Google's Data Labeling Service



Boeing Airbus

#3: EVALUATOR

- Before setting out to build any ML models with an initial dataset for the machine to learn from
 - it's important to define how to evaluate the planned ML system
- In addition to measuring prediction accuracy,
 - evaluation of short-term and long-term impact via application-specific performance metrics,
 - and system metrics such as lag and throughput, are desirable
- Two important objectives behind model evaluation:
 - comparing models
 - deciding whether it is safe to integrate a model into an application
- Evaluation can be performed on a predetermined set of test cases, for which it is known what the prediction should be (i.e. the ground truth)
- The error distribution can be examined, and errors can be aggregated into performance metrics
 - evaluator needs access to the test set's ground truth, so that when it gets predictions in input, it can compute prediction errors and return performance metrics

#4: PERFORMANCE MONITOR

- The next step towards deciding if a (baseline) model can be integrated into an application is
 - to use it on the inputs encountered in production (called ‘production data’), in a production-like setting,
 - and to monitor its performance through time
- Computing and monitoring performance metrics on production data requires
 - acquiring and storing production inputs, ground truths, and predictions in a database
- The performance monitor would consist of a program that
 - reads from that database,
 - calls the evaluator,
 - and updates a dashboard that shows how performance metrics evolve with time
- Need to check that models behave well through time and that they keep having a positive impact on the application in which they’re integrated
 - could also be augmented with data visualization widgets that show production data distributions,
 - so can make sure they are as expected, or can monitor drift and anomalies

#5: FEATURIZER

- When designing a prediction API, a decision needs to be made as to what the API should take as input
 - For example,
 - when making predictions about customers, should the input be the full feature representation of the customer, or just the customer id?
- Common that the full numerical representation would not be readily available
 - but it would have to be computed before it can be passed to a model
- For a customer input, some features would be already stored in a database
 - for example, date of birth
- Others would require computation-
 - For example, how the customer interacted with the product over a certain period of time
 - would be computed by querying and aggregating data that logged customer interactions with the product

#5: FEATURIZER(2)

- If, by nature, features do not change too often, they could be computed in batches
- Could have “hot” features that would change rapidly and would need to be computed in real-time
 - for instance, the average delivery time of a given restaurant over the last X minutes
- Calls for the creation of at least one featurization microservice
 - that would extract features for a batch of inputs, based on their ids
- May also need a real-time featurization microservice,
 - but this would be at an additional cost in the complexity of ML system
- Featurizers may query various databases and perform aggregations and treatments on the queried data
 - may have parameters (such as the number of minutes X), which may have an impact on the performance of models.

#6. ORCHESTRATOR

Workflow

- The orchestrator is at the core of the ML system and interacts with many other components
- Steps in its workflow/pipeline:
 - 1) Extract-Transform-Load and split (raw) data into training, validation, test sets
 - 2) Send training/validation/test sets for featurization (if any)
 - 3) Prepare featurized training/validation/test sets
 - 4) Send URLs of prepared train/validation sets, along with metric to optimize, to model builder
 - 5) Get optimal model, apply to test set, and send predictions to evaluator
 - 6) Get performance value and decide if it's OK to push the model to the server (for canary-testing on production data, for instance).
- Some more details on step #3 (“prepare featurized training/validation/test sets”):
 - Augment training data (for example, oversample/undersample, or rotate/flip/crop images)
 - Pre-process training/validation/test sets, with data sanitization (so that it can be safely used for modeling or predicting) and problem-specific preparation (for example, de-saturate and resize images).

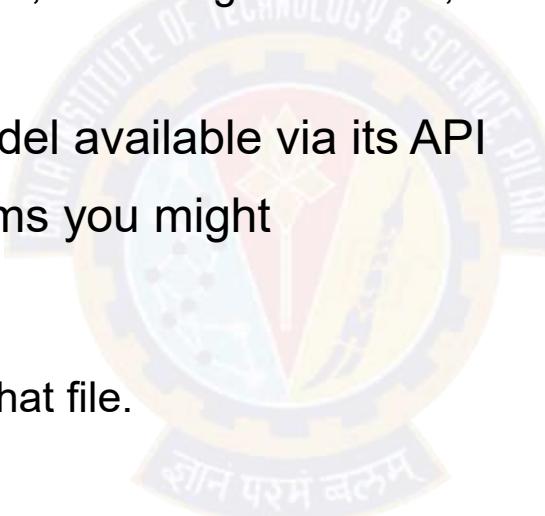
#6. ORCHESTRATOR

Ways to run the workflow

- The whole workflow could be executed manually
- Will have to be automated
 - If needs to update models frequently, or to tune hyperparameters of the featurizer and of the modeler jointly
- Workflow might be implemented as a simple script and run on a single thread, but computations could be more efficient by parallelizing runs.
 - End-to-end ML platforms allow doing that and can provide a single environment to define and run full ML pipelines
 - Google AI Platform, Databricks platform
- Alternatively, each step of the workflow might run on a different platform or in a different computing environment
 - One option is to execute these steps in different Docker containers
 - Kubernetes is one of the most popular open-source container orchestration systems among ML practitioners.
 - Apache Airflow is another open-source workflow management tool, originally developed by Airbnb

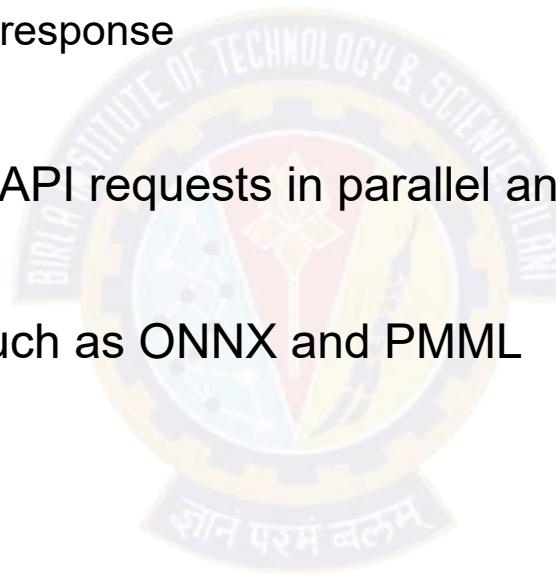
#7. MODEL BUILDER

- The model builder is in charge of providing an optimal model
 - trains various models on the training set
 - evaluates them on the validation set, with the given metric, in order to assess optimality
- BigML automatically makes the model available via its API
- With other ML development platforms you might
 - want to package the model,
 - save it as a file,
 - and have your model server load that file.
- If use a different ML development platform or no platform at all,
 - it's worth architecting system in a way where models are automatically created by a dedicated service,
 - which takes in a training set, a validation set, and a performance metric to optimize



#8. MODEL SERVER

- The role of a model server is to process API requests for predictions against a given model
 - loads a model representation saved in a file and applies to the inputs found in the API request
 - predictions are then returned in the API response
- Server should allow for serving multiple API requests in parallel and for model updates
- Different model representations exist, such as ONNX and PMML



#9. FRONT-END

- The front-end can serve multiple purposes:
 - simplify the model's output, for instance by turning a list of class probabilities into the most likely class
 - add to the model's output, for instance by using a black box model explainer and providing a prediction explanation
 - implement domain-specific logic, such as decisions based on predictions, or a fallback when receiving anomalous inputs
 - send production inputs and model predictions for storage in the production database
 - test new models, by also querying predictions from them (in addition to the “live” model) and storing them
 - allow the monitor to plot performance metrics over time, for these new candidate models



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

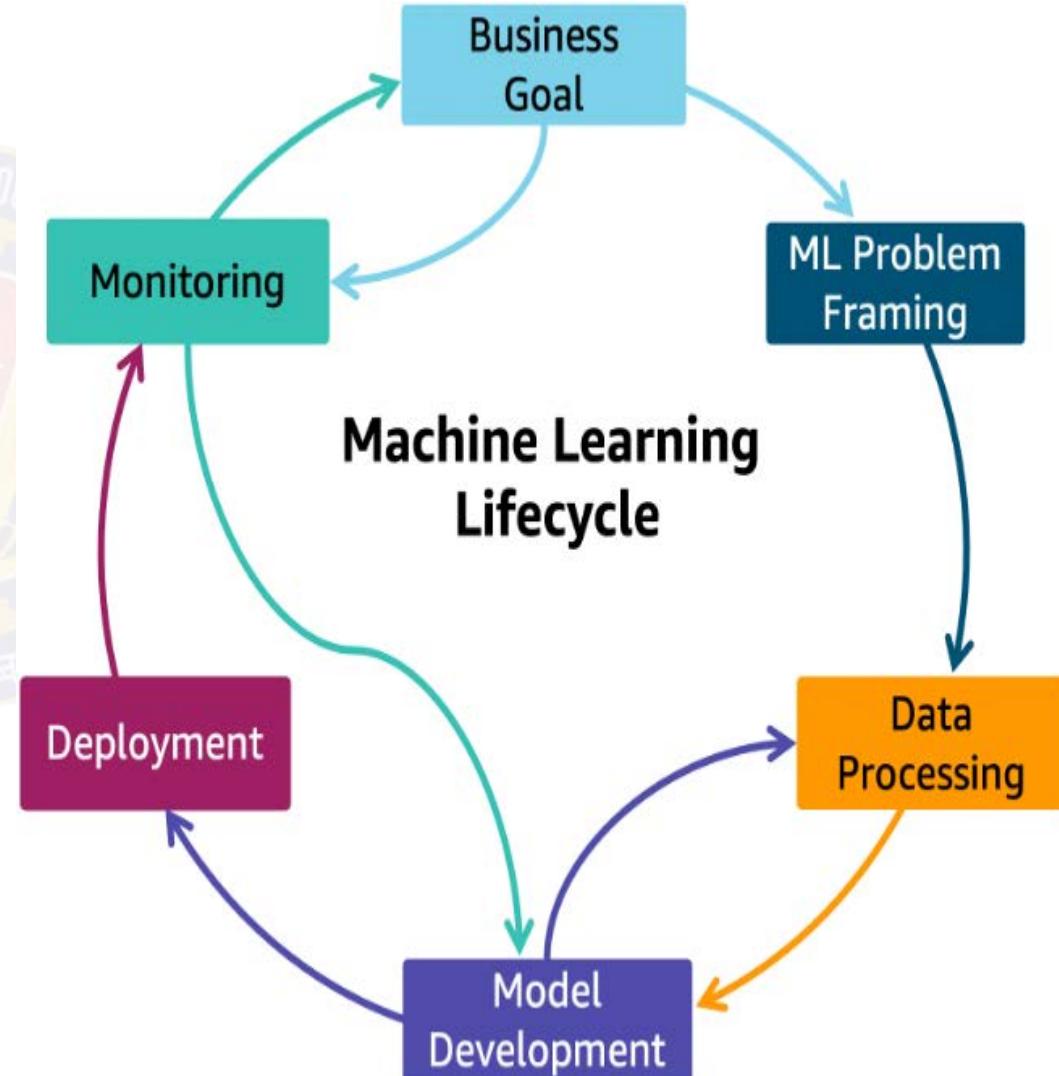
Machine Learning Lifecycle

Pravin Y Pawar

Extracted from AWS Well-Architected machine learning

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



Machine Learning Lifecycle(2)

Phases

- 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
- 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
- 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
- Model development
 - consists of model building, training, tuning, and evaluation
 - includes creating a CI/CD pipeline that automates the build, train and release to staging and production environments
- Deployment
 - After a model is trained, tuned, evaluated and validated, one can deploy the model into production
 - can then make predictions and inferences against the model
- Monitoring
 - ensures model is maintaining a desired level of performance through early detection and mitigation

ML lifecycle phase — Business goal

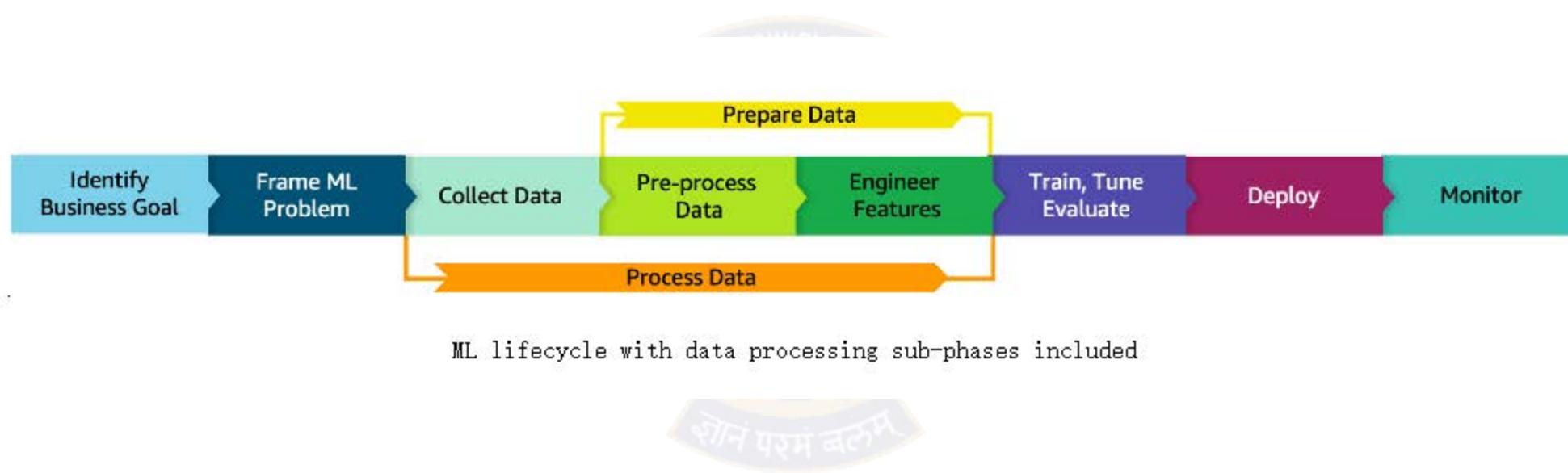
- The most important phase particularly challenging when considering ML solutions because ML is a constantly evolving technology!
- After success criteria is defined, evaluate organization's ability to move toward that target
 - The target should be **achievable** and provide a clear path to production
 - Involve
 - all relevant stakeholders from the beginning to align them to this target
 - 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

ML lifecycle phase — ML problem framing

- 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 must be optimized is a key step in ML
- 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

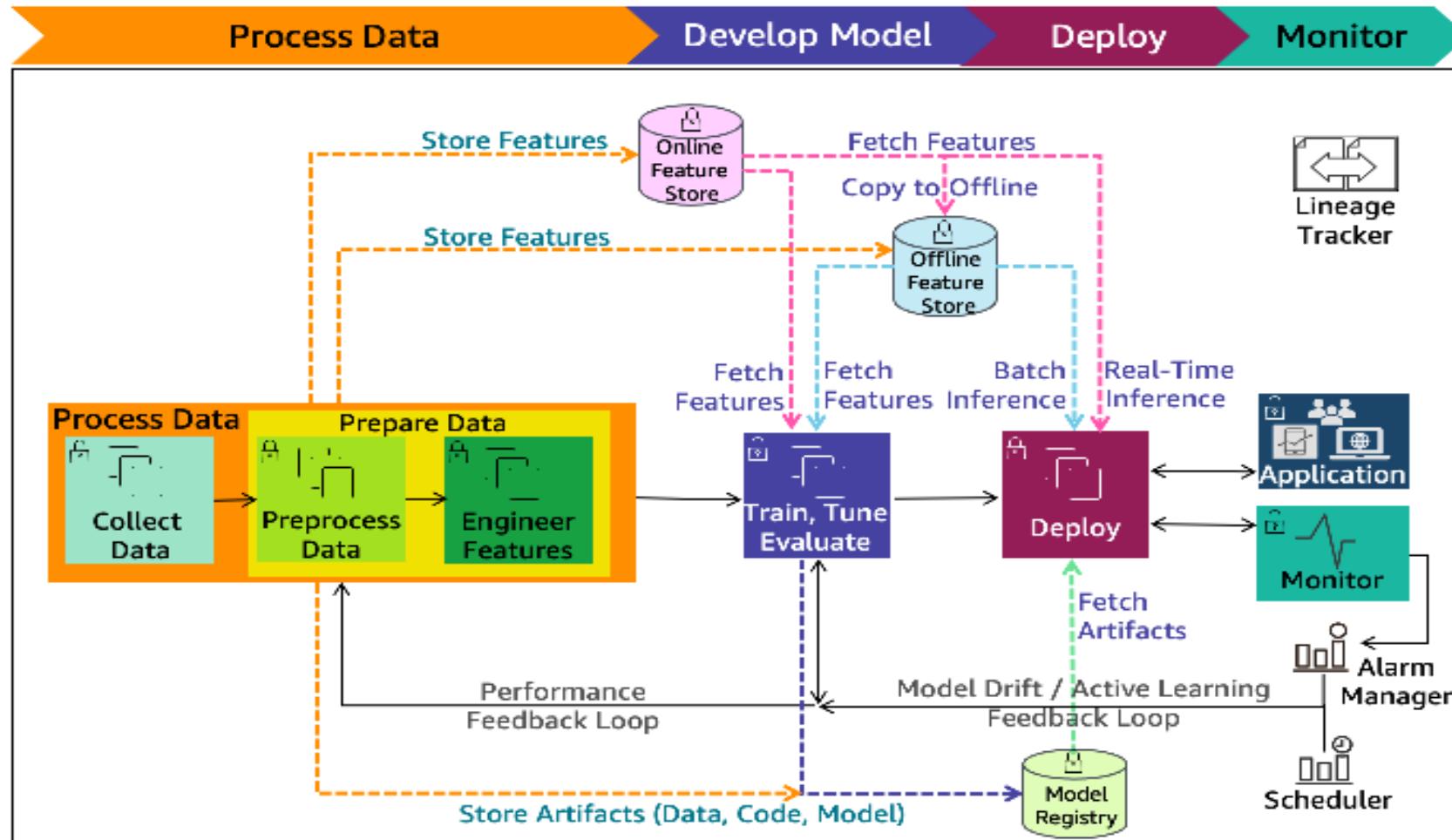
ML lifecycle architecture diagram

ML lifecycle with data processing sub-phases included



ML lifecycle architecture diagram(2)

ML lifecycle with detailed phases and expanded components



ML lifecycle with detailed phases and expanded components

ML lifecycle architecture diagram(3)

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

ML lifecycle architecture diagram(4)

Components

- 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

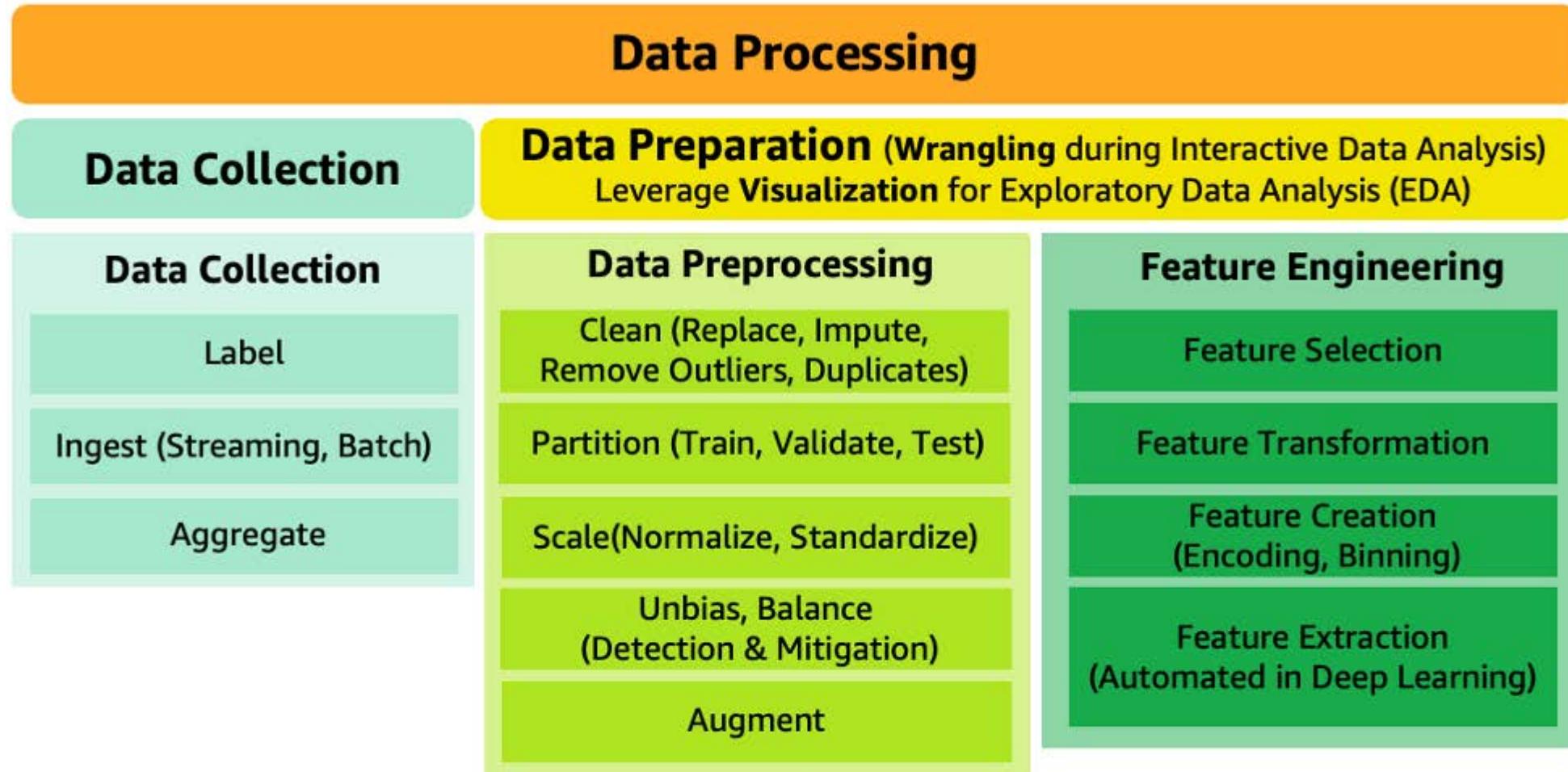
ML lifecycle phase - Data processing

Need

- 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 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.

ML lifecycle phase - Data processing(2)

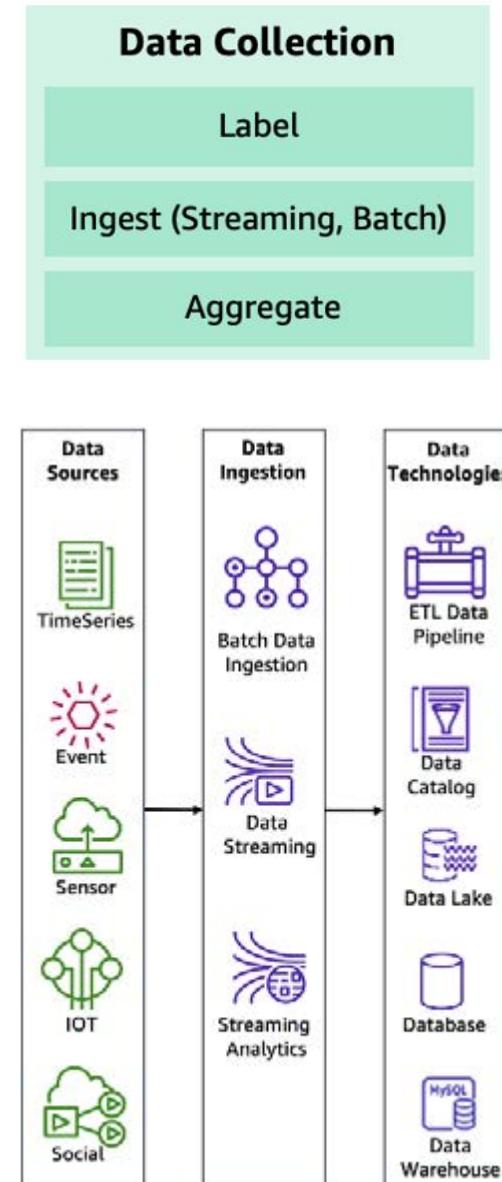
Data processing components



ML lifecycle phase - Data processing(3)

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 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 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



ML lifecycle phase - Data processing(4)

Data preparation

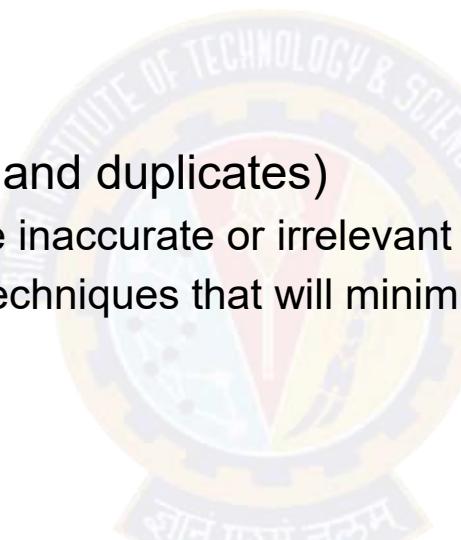
- ML models are only as good as the data that is used to train them
 - Ensure that suitable training data is available and is optimized for learning and generalization
- Data preparation includes
 - data preprocessing
 - feature engineering
- A key aspect to understanding data is to identify patterns 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 reduces the cost for interactive analysis



ML lifecycle phase - Data processing(5)

Data preprocessing

- Data preprocessing puts data into the right shape and quality for training
 - Many strategies: data cleaning, balancing, replacing, imputing, partitioning, scaling, augmenting and unbiasing
- 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



Data Preprocessing

Clean (Replace, Impute, Remove Outliers, Duplicates)

Partition (Train, Validate, Test)

Scale(Normalize, Standardize)

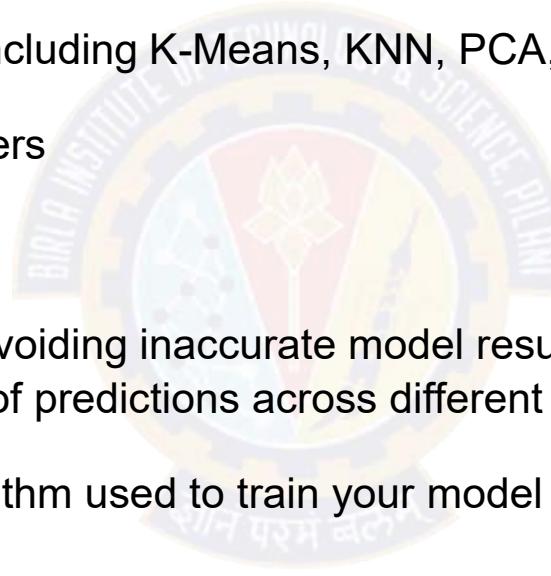
Unbias, Balance
(Detection & Mitigation)

Augment

ML lifecycle phase - Data processing(6)

Data preprocessing

- 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



Data Preprocessing

Clean (Replace, Impute, Remove Outliers, Duplicates)

Partition (Train, Validate, Test)

Scale(Normalize, Standardize)

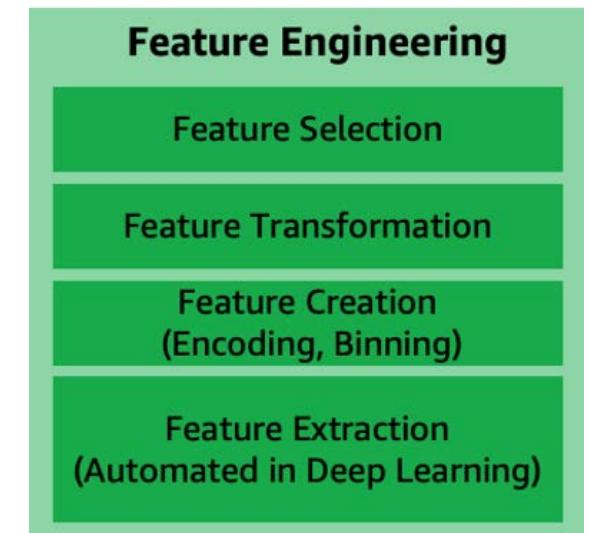
Unbias, Balance
(Detection & Mitigation)

Augment

ML lifecycle phase - Data processing(7)

Feature engineering

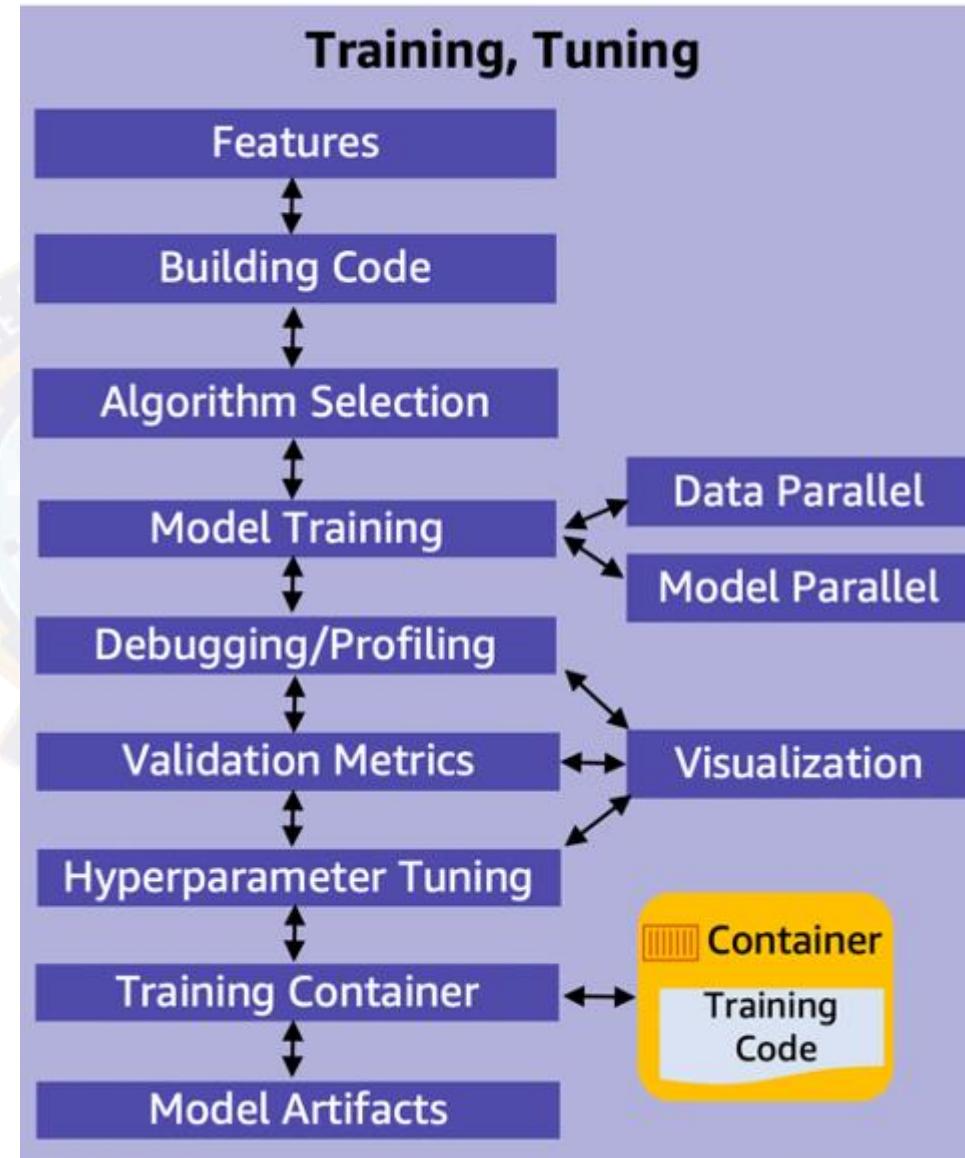
- Every unique attribute of the data is considered a feature
- Feature engineering is a process to select and transform variables when creating a predictive model using machine learning or statistical modeling
 - includes feature creation, feature transformation, feature extraction, and feature selection
 - With deep learning, the feature engineering is automated as part of the algorithm learning
- Feature 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 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



ML lifecycle phase – Model development

Model building, training, tuning

- In this phase, select a machine learning algorithm that is appropriate for 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
 - and set the hyperparameters to optimize the training process



ML lifecycle phase – Model development(2)

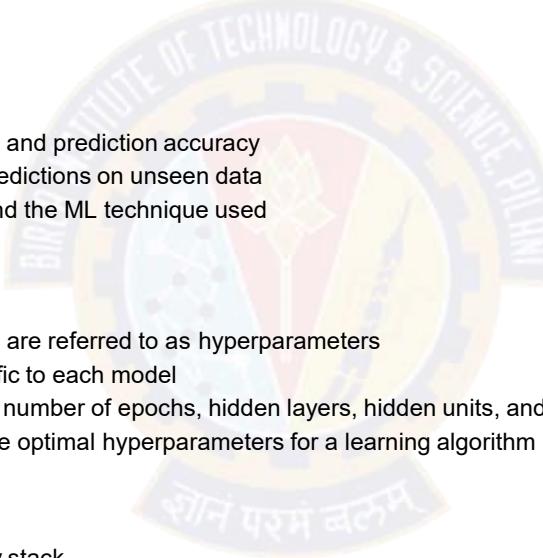
Model training, tuning activities

- Model training, tuning, and evaluation require prepared data and engineered features
- Features 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 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 in mini-batches evenly distributed across nodes
 - each node only trains the model on a fraction of the total dataset

ML lifecycle phase – Model development(3)

Model training, tuning activities

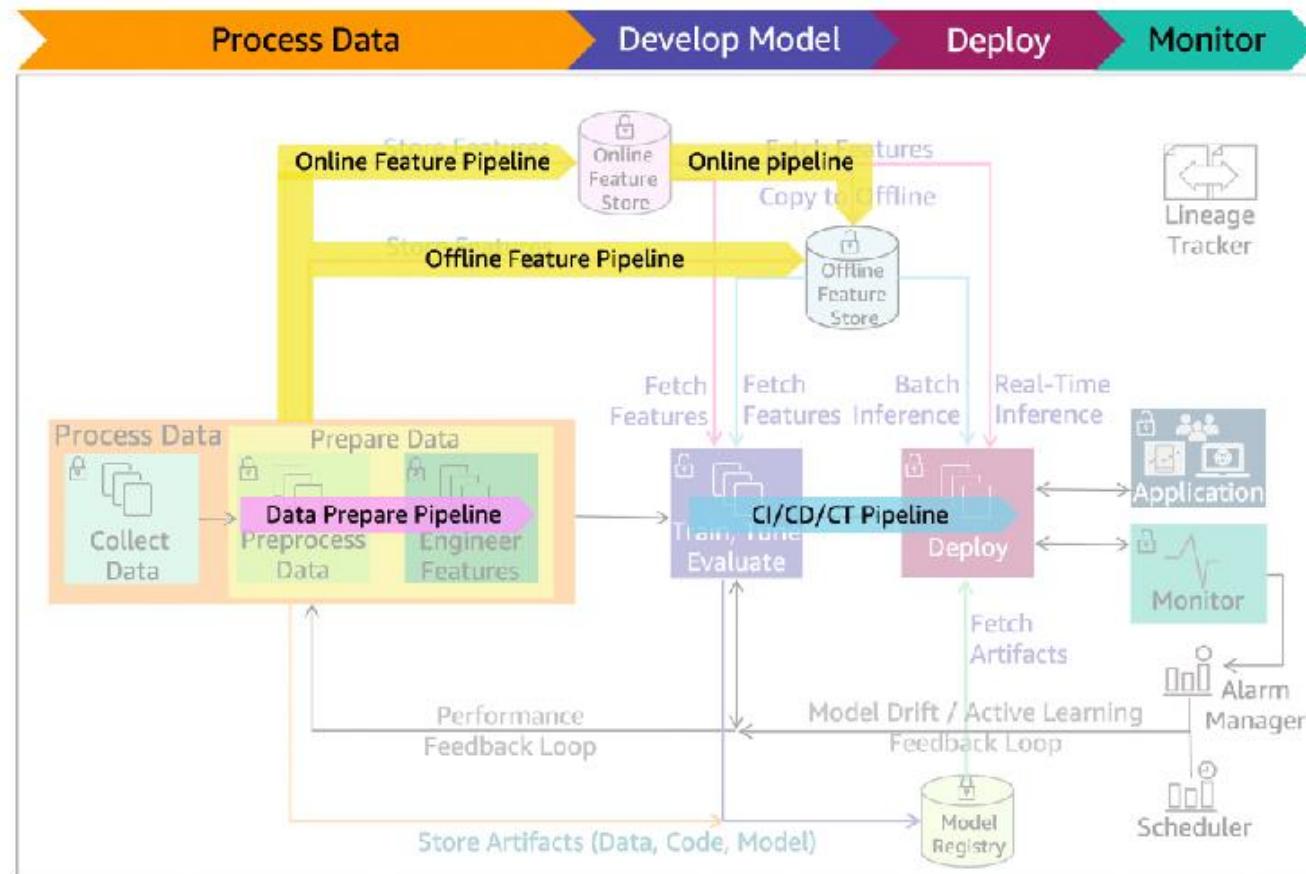
- Debugging/profiling
 - A machine learning training job can have problems including:
 - system bottlenecks, overfitting, saturated activation functions, and vanishing gradients. These problems can compromise model performance
 - A debugger provides visibility into the ML training process through monitoring, 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 phase – Model development(4)

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



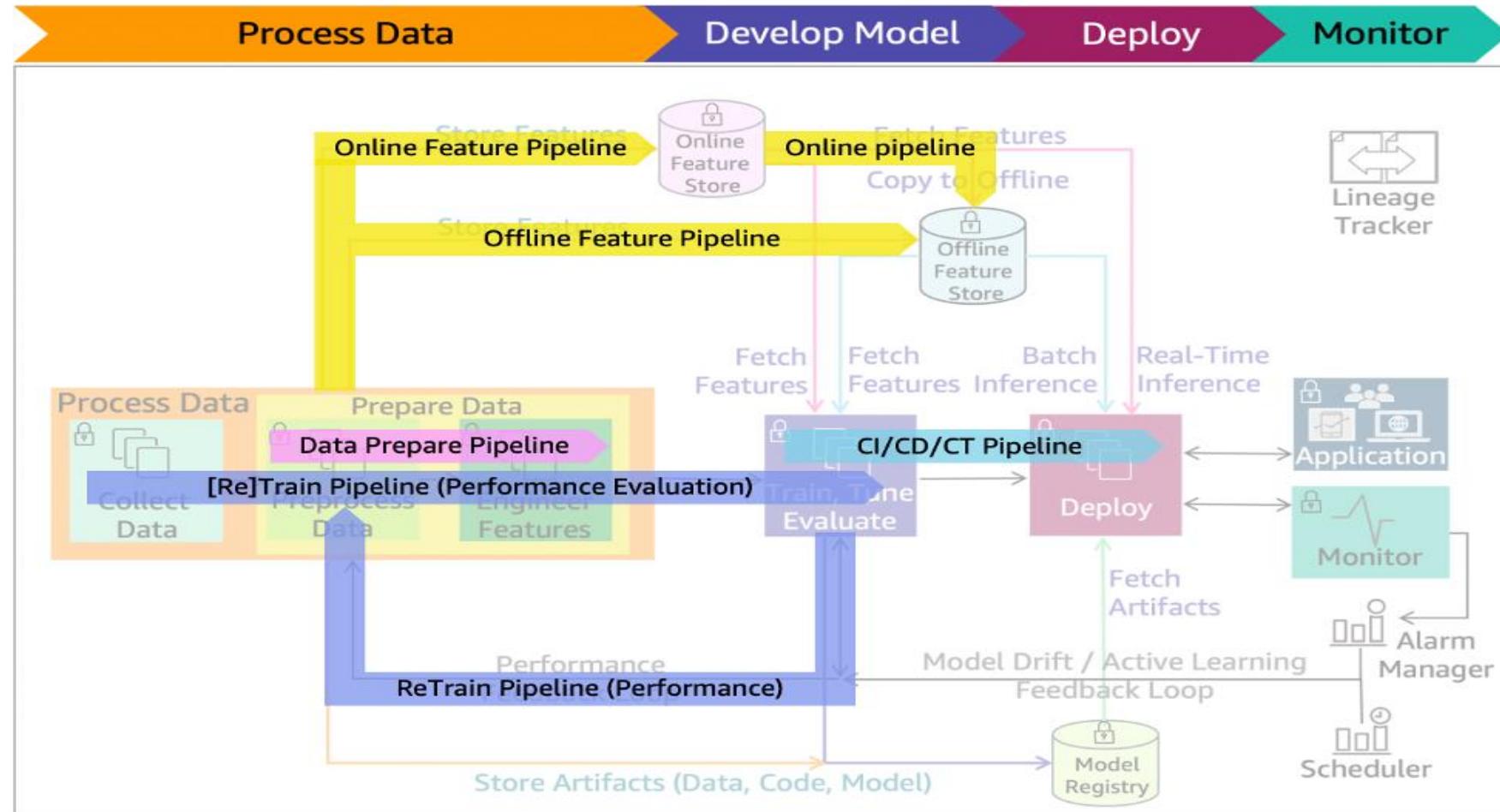
ML lifecycle phase – Model development(5)

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

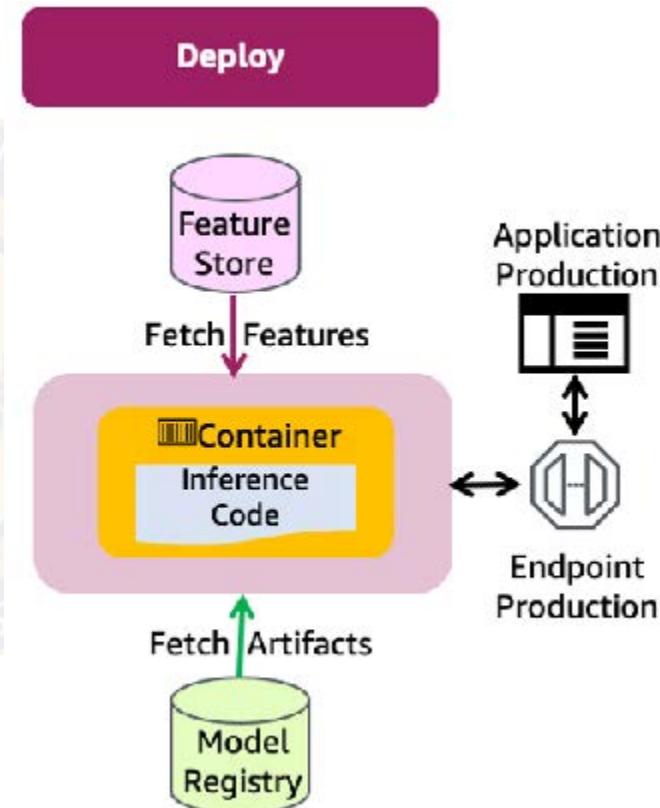
ML lifecycle phase – Model development(6)

ML lifecycle with performance evaluation pipeline added



ML lifecycle phase - Deployment

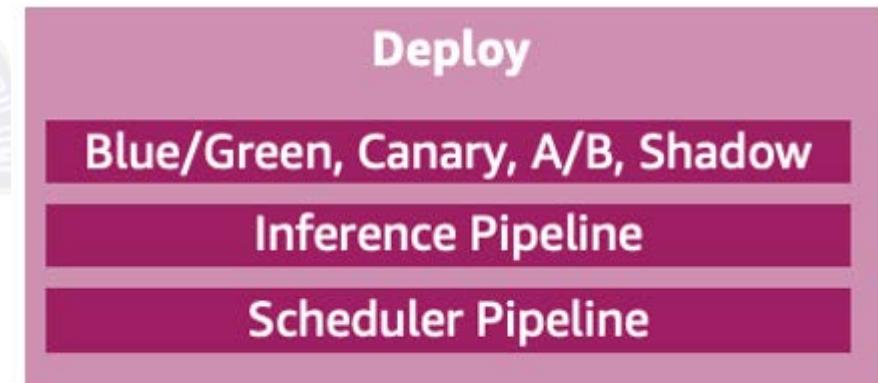
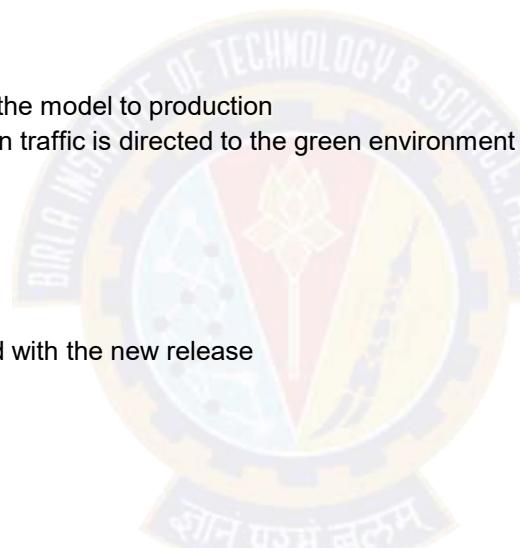
- After model have been trained, tuned, and evaluated, then can deploy the model into production
 - can make predictions and inferences against the deployed model
 - Use a manual governance process - ensures the model has fully been tested and evaluated before it's released to production
- Deploy production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



ML lifecycle phase – Deployment(2)

Deployment main components

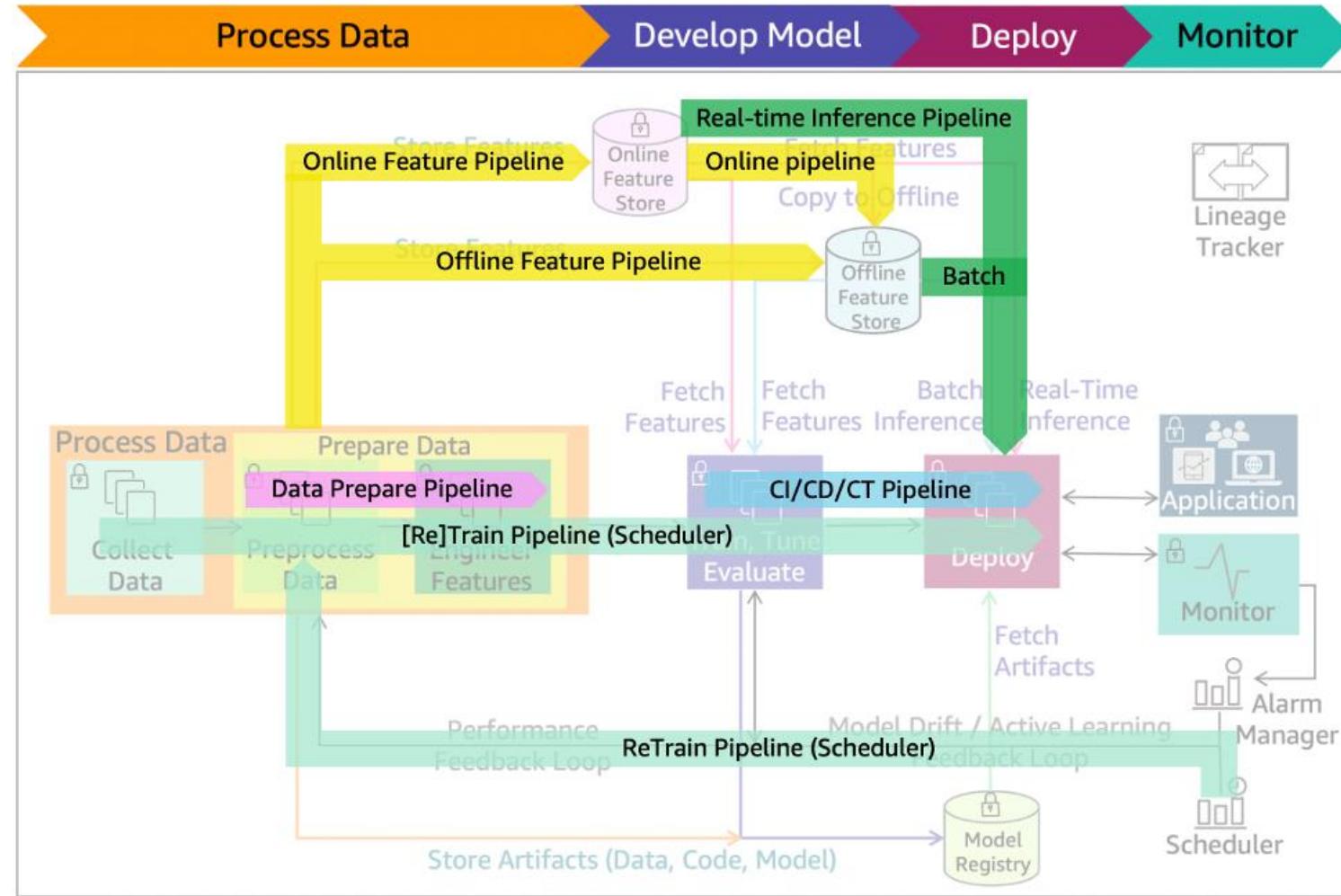
- Blue/green, canary, A/B, shadow deployment/testing
 - Deployment and testing strategies that reduce downtime and risks when releasing a new or updated version
- The blue/green deployment technique
 - provides two identical production environments
 - can use this technique when need to deploy a new 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 deploy 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
- 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
- Shadow deployment strategy
 - New version is 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 phase – Deployment(3)

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
 - 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



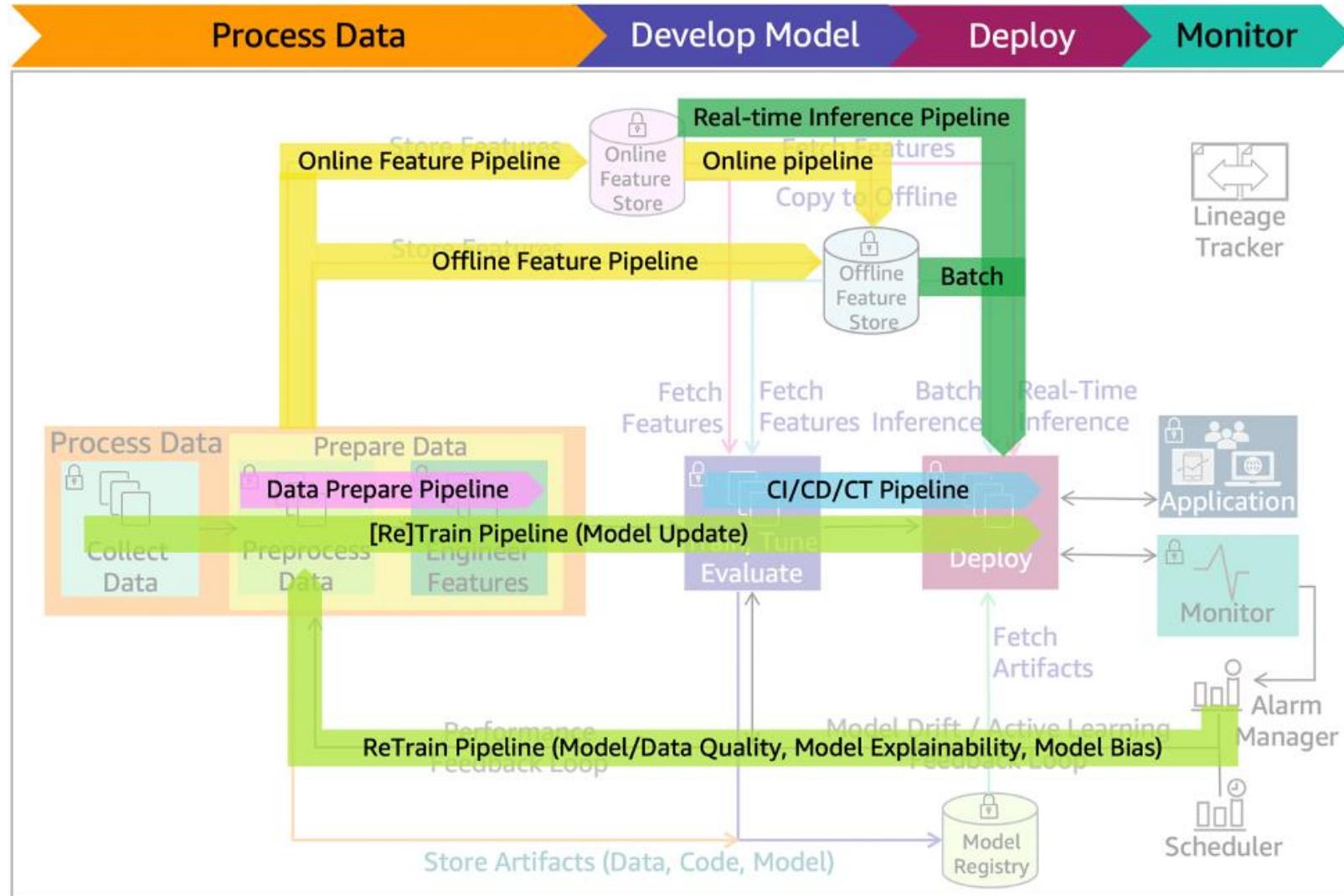
ML lifecycle phase – Monitoring

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

ML lifecycle phase – Monitoring(2)

ML lifecycle with model update re-train and batch/real-time Inference pipelines

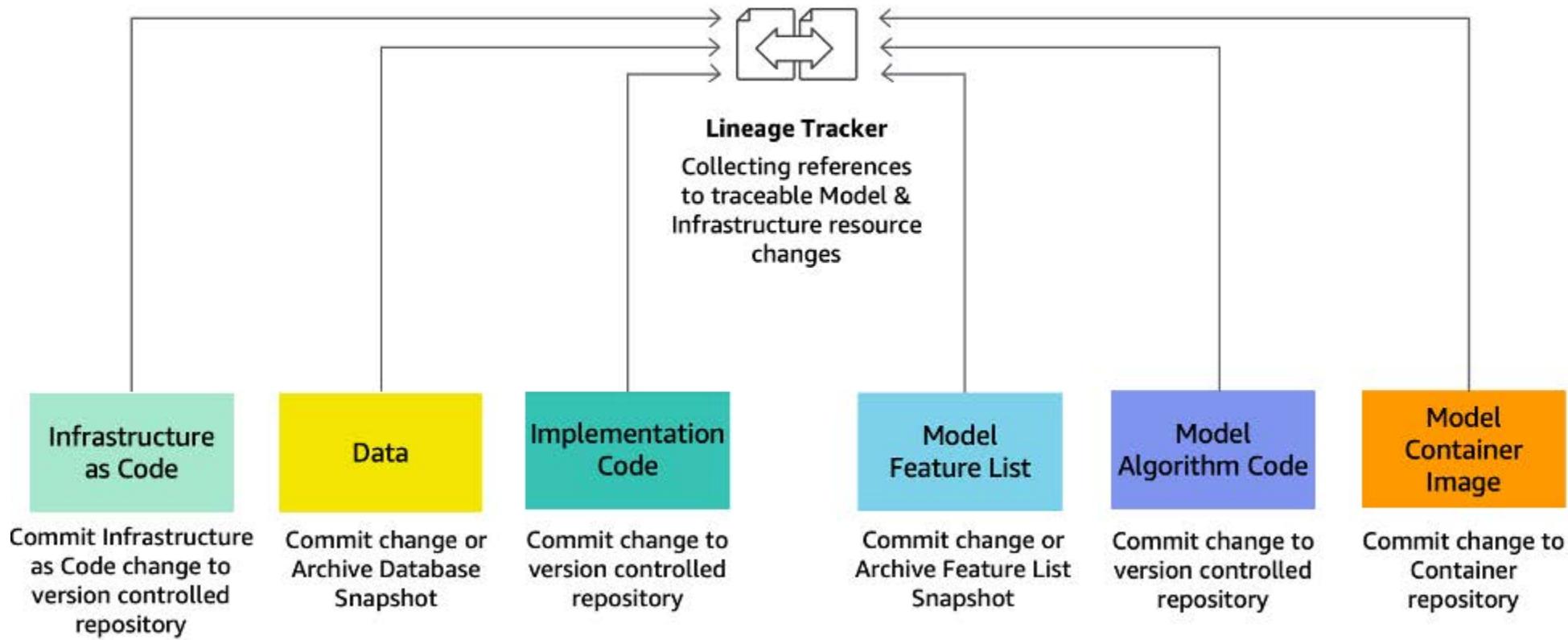


Additional - Lineage tracker

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

Lineage tracker



Lineage tracker

Components

- 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



Thank You!

In our next session:

Challenges in Deploying Machine Learning: a Survey of Case Studies

Andrei Paleyes
 Department of Computer Science
 University of Cambridge
 ap2169@cam.ac.uk

Raoul-Gabriel Urma
 Cambridge Spark
 raoul@cambridgespark.com

Neil D. Lawrence
 Department of Computer Science
 University of Cambridge
 ndl21@cam.ac.uk

Abstract

In recent years, machine learning has received increased interest both as an academic research field and as a solution for real-world business problems. However, the deployment of machine learning models in production systems can present a number of issues and concerns. This survey reviews published reports of deploying machine learning solutions in a variety of use cases, industries and applications and extracts practical considerations corresponding to stages of the machine learning deployment workflow. Our survey shows that practitioners face challenges at each stage of the deployment. The goal of this paper is to layout a research agenda to explore approaches addressing these challenges.

1 Introduction

Machine learning (ML) has evolved from purely an area of academic research to an applied field. In fact, according to a recent global survey conducted by McKinsey, machine learning is increasingly adopted in standard business processes with nearly 25 percent year-over-year growth [1] and with growing interest from the general public, business leaders [2] and governments [3].

This shift comes with challenges. Just as with any other field, there are significant differences between what works in academic setting and what is required by a real world system. Certain bottlenecks and invalidated assumptions should always be expected in the course of that process. As more solutions are developed and deployed, practitioners sometimes report their experience in various forms, including publications and blog posts. In this study, we undertake a survey of these reports to capture the current challenges in deploying machine learning in production¹. First, we provide an overview of the machine learning deployment workflow. Second, we review use case studies to extract problems and concerns practitioners have at each particular deployment stage. Third, we discuss cross-cutting aspects that affect every stage of the deployment workflow: ethical considerations, end users' trust and security. Finally, we conclude with a brief discussion of potential solutions to these issues and further work.

A decade ago such surveys were already conducted, albeit with a different purpose in mind. Machine learning was mainly a research discipline, and it was uncommon to see ML solution deployed for a business problem outside of “Big Tech” companies in the information technology industry. So

¹By *production* we understand the setting where a product or a service are made available for use by its intended audience.

the purpose of such a survey was to show that ML can be used to solve a variety of problems, and illustrate it with examples, as was done by Pěchouček and Mařík [4]. Nowadays the focus has changed: machine learning is commonly adopted in many industries, and the question becomes not “Where is it used?”, but rather “How difficult is it to use?”

A popular approach to assess the current state of machine learning deployment for businesses is a survey conducted among professionals. Such surveys are primarily undertaken by private companies, and encompass a variety of topics. Algorithmia’s report ([5], [6]) goes deep into deployment timeline, with the majority of companies reporting between 8 and 90 days to deploy a single model, and 18% taking even more time. A report by IDC [7] that surveyed 2,473 organizations and their experience with ML found that a significant portion of attempted deployments fail, quoting lack of expertise, bias in data and high costs as primary reasons. O’Reilly has conducted an interview study that focused on ML practitioners’ work experience and tools they use [8]. A broader interview-based reports have also been produced by dotscience [9] and dimensional research [10].

While there are a number of business reports on the topic, the challenges of the entire machine learning deployment pipeline are not covered nearly as widely in the academic literature. There is a burgeoning field of publications focusing on specific aspects of deployed ML, such as Bhatt et al. [11] which focuses on explainable ML or Amershi et al. [12] which discusses software engineering aspects of deploying ML. A large number of industry-specific surveys also exist, and we review some of these works in the appropriate sections below. However the only general purpose survey we found is Baier et al. [13], which combines literature review and interviews with industry partners. In contrast with that paper, which concentrates on experience of information technology sector, we aim at covering case studies from a wide variety of industries. We also discuss the most commonly reported challenges in the literature in much greater detail.

In our survey we consider three main types of papers:

- Case study papers that report experience from a single ML deployment project. Such works usually go deep into discussing each challenge the authors faced and how it was overcome.
- Review papers that describe applications of ML in a particular field or industry. These reviews normally give a summary of challenges that are most commonly encountered during the deployment of the ML solutions in the reviewed field.
- “Lessons learned” papers where authors reflect on their past experiences of deploying ML in production.

To ensure that this survey focuses on the current challenges, only papers published in the last five years are considered, with only few exceptions to this rule. We also refer other types of papers where it is appropriate, e.g. practical guidance reports, interview studies and regulations. We have not conducted any interviews ourselves as part of this study. Further work and extensions are discussed at the end of this paper.

The main contribution of our paper is to show that practitioners face challenges at each stage of the machine learning deployment workflow. This survey supports our goal to raise awareness in the academic community to the variety of problems that practitioners face when deploying machine learning, and start a discussion on what can be done to address these problems.

2 Machine Learning Deployment Workflow

For the purposes of this work we are using the ML deployment workflow definition suggested by Ashmore et al. [14], however it is possible to conduct a similar review with any other pipeline description. In this section we give a brief overview of the definition we are using.

According to Ashmore et. al. [14], the process of developing an ML-based solution in an industrial setting consists of four stages:

- **Data management**, which focuses on preparing data that is needed to build a machine learning model;
- **Model learning**, where model selection and training happens;
- **Model verification**, the main goal of which is to ensure model adheres to certain functional and performance requirements;

- **Model deployment**, which is about integration of the trained model into the software infrastructure that is necessary to run it. This stage also covers questions around model maintenance and updates.

Each of these stages is broken down further into smaller steps. It is important to highlight that the apparent sequence of this description is not necessarily the norm in a real-life scenario. It is perfectly normal for these stages to run in parallel to a certain degree and inform each other via feedback loops. Therefore this or any similar breakdown should be considered not as a timeline, but rather as a useful abstraction to simplify referring to concrete parts of the deployment pipeline.

For the remainder of this paper we discuss common issues practitioners face at each each stage and granular constituent steps. We also discuss cross-cutting aspects that can affect every stage of the deployment pipeline. Table \textcolor{red}{I} provides a summary of the issues and concerns we discuss. We do not claim that this list is exhaustive, nonetheless by providing illustrative examples for each step of the workflow we show how troublesome the whole deployment experience is today.

3 Data Management

Data is an integral part of any machine learning solution. Overall effectiveness of the solution depends on the training and test data as much as on the algorithm. The process of creating quality datasets is usually the very first stage in any production ML pipeline. Unsurprisingly, practitioners face a range of issues while working with data as previously reported by Polyzotis et al. \textcolor{blue}{[5]}. Consequently, this stage consumes time and energy that is often not anticipated beforehand. In this section, we describe issues concerning four steps within data management: data collection, data preprocessing, data augmentation and data analysis. Note that we consider storage infrastructure challenges, such as setting up databases and query engines, beyond the scope of this survey.

3.1 Data collection

Data collection involves activities that aim to discover and understand what data is available, as well as how to organize convenient storage for it. The task of discovering what data exists and where it is can be a challenge by itself, especially in large production environments. Finding data sources and understanding their structure is a major task, which may prevent data scientists from even getting started on the actual application development. As explained by Lin and Ryaboy \textcolor{blue}{[6]}, at Twitter this situation often happened due to the fact that the same entity (e.g. a Twitter user) is processed by multiple services. Internally Twitter consists of multiple services calling each other, and every service is responsible for a single operation. This approach, known in software engineering as “single responsibility principle” \textcolor{blue}{[7]}, results in an architecture that is very flexible in terms of scalability and modification. However, the flip side of this approach is that at a large scale it is impossible to keep track of what data related to the entity is being stored by which service, and in which form. Besides, some data may only exist in a form of logs, which by their nature are not easily parsed or queried. An even worse case is the situation when data is not stored anywhere, and in order to build a dataset one needs to generate synthetic service API calls. Such a dispersion of data creates major hurdles for data scientists, because without a clear idea of what data is available or can be obtained it is often impossible to understand what ML solutions can realistically achieve.

3.2 Data preprocessing

The preprocessing step normally involves a range of data cleaning activities: imputation of missing values, reduction of data into an ordered and simplified form, and mapping from raw form into a more convenient format. Methods for carrying out data manipulations like this is an area of research that goes beyond the scope of this study. We encourage readers to refer to review papers on the topic, such as Abedjan et al. \textcolor{blue}{[8]}, Patil and Kulkarni \textcolor{blue}{[9]}, Ridzuan et al. \textcolor{blue}{[20]}. An approach towards classifying readiness of data for ML tasks is given by Lawrence \textcolor{blue}{[21]}.

A lesser known but also important problem that can also be considered an object of the preprocessing step is data dispersion. It often turns out that there can be multiple relevant separate data sources which may have different schemas, different conventions, and their own way of storing and accessing the data. Joining this information into a single dataset suitable for machine learning can be a complicated task on its own right. An example of this is what developers of Firebird faced

Table 1: All considerations, issues and concerns explored in this study. Each is assigned to the stage and step of the deployment workflow where it is commonly encountered.

Deployment Stage	Deployment Step	Considerations, Issues and Concerns
Data management	Data collection	Data discovery
	Data preprocessing	Data dispersion Data cleaning
	Data augmentation	Labeling of large volumes of data Access to experts Lack of high-variance data
	Data analysis	Data profiling
Model learning	Model selection	Model complexity Resource-constrained environments Interpretability of the model
	Training	Computational cost Environmental impact
	Hyper-parameter selection	Resource-heavy techniques Hardware-aware optimization
Model verification	Requirement encoding	Performance metrics Business driven metrics
	Formal verification	Regulatory frameworks
	Test-based verification	Simulation-based testing
Model deployment	Integration	Operational support Reuse of code and models Software engineering anti-patterns Mixed team dynamics
	Monitoring	Feedback loops Outlier detection Custom design tooling
	Updating	Concept drift Continuous delivery
Cross-cutting aspects	Ethics	Country-level regulations Focus on technical solution only Aggravation of biases Authorship Decision making
	End users' trust	Involvement of end users User experience Explainability score
	Security	Data poisoning Model stealing Model inversion

[22]. Firebird is an advisory system in the Atlanta Fire Department, that helps identify priority targets for fire inspections. As a first step towards developing Firebird data was collected from 12 datasets including history of fire incidents, business licenses, households and more. These datasets were combined to give a single dataset covering all relevant aspects of each property monitored by the Fire Department. Authors particularly highlight data joining as a difficult problem. Given the fact that building location is the best way to identify that building, each dataset contained spatial data specifying building's address. Spatial information can be presented in different formats, and sometimes contain minor differences such as different spellings. All this needed to be cleaned and corrected, and turned out to be a massive effort that consumed a lot of time.

3.3 Data augmentation

There are multiple reasons why data might need to be augmented, and in practice one of the most problematic ones is the absence of labels. Real-world data is often unlabeled, thus labeling turns out to be a challenge in its own right. We discuss three possible factors for lack of labeled data: limited access to experts, absence of high-variance data, and sheer volume.

Labels assignment is difficult in environments that tend to generate large volumes of data, such as network traffic analysis. To illustrate a scale of this volume, a single 1-GB/s Ethernet interface can deliver up to 1.5 million packets per second. Even with a huge downsampling rate this is still a significant number, and each sampled packet needs to be traced in order to be labeled. This problem is described by Pacheco et al. [23], which surveys applications of machine learning to network traffic classification, with tasks such as protocol identification or attack detection. There are two main ways of acquiring data in this domain, and both are complicated for labeling purposes:

- Uncontrolled, collecting real traffic. This approach requires complex tracking flows belonging to a specific application. Due to this complexity very few works implement reliable ground truth assignment for real traffic.
- Controlled, emulating or generating traffic. Even in this case it has been shown that existing tools for label assignment introduce errors into collected ML datasets, going as high as almost 100% for certain applications. Moreover, these tools' performance degrades severely for encrypted traffic.

Access to experts can be another bottleneck for collecting high-quality labels. It is particularly true for areas where expertise mandated by the labeling process is significant, such as medical image analysis [24]. Normally multiple experts are asked to label a set of images, and then these labels are aggregated to ensure quality. This is rarely feasible for big datasets due to experts' availability. A possible option here is to use noisy label oracles or weaker annotations, however these approaches are by their definition a trade off that provides imprecise labels, which ultimately leads to a severe loss in quality of the model. Such losses are unacceptable in healthcare industry, where even the smallest deviation can cause catastrophic results (this is known as The Final Percent challenge).

Lack of access to high-variance data can be among the main challenges one faces when deploying machine learning solution from lab environment to real world. Dulac-Arnold et al. [25] explain that this is the case for Reinforcement Learning (RL). It is common practice in RL research to have access to separate environments for training and evaluation of an agent. However, in practice all data comes from the real system, and the agent can no longer have a separate exploration policy - this is simply unsafe. Therefore the data available becomes low-variance. While this approach ensures safety, it means that agent is not trained to recognize an unsafe situation and make right decision in it. A practical example of this issue is the goal specification for autonomous vehicles [26].

3.4 Data analysis

Data needs to be analyzed in order to uncover potential biases or unexpected distributions in it. Availability of high quality tools is essential for conducting any kind of data analysis. One area that practitioners find particularly challenging in that regard is visualization for data profiling [27]. Data profiling refers to all activities associated with troubleshooting data quality, such as missing values, inconsistent data types and verification of assumptions. Despite obvious relevance to the fields of databases and statistics, there are still too few tools that enable efficient execution of these data mining tasks. The need for such tools becomes apparent considering that, according to the

survey conducted by Microsoft [28], data scientists think about data issues as the main reason to doubt quality of the overall work.

4 Model Learning

Model learning is the stage of the deployment workflow that enjoys the most attention within the academic community. All modern research in machine learning methods contributes towards better selection and variety of models and approaches that can be employed at this stage. As an illustration of the scale of the field’s growth, the number of submissions to NeurIPS, primary conference on ML methods, has quadrupled in six years, going from 1678 submissions in 2014 to 6743 in 2019 [29]. Nevertheless, there is still plenty of practical considerations that affect the model learning stage. In this section, we discuss issues concerning three steps within model learning: model selection, training and hyper-parameter selection.

4.1 Model selection

In many practical cases the selection of a model is often decided by one key characteristic of a model: complexity. Despite areas such as deep learning and reinforcement learning gaining increasing levels of popularity with the research community, in practice simpler models are often chosen as we explain below. Such model include shallow network architectures, simple PCA-base approaches, decision trees and random forests.

Simple models can be used as a way to prove the concept of the proposed ML solution and get the end-to-end setup in place. This approach accelerates the time to get a deployed solution, allows the collection of important feedback and also helps avoid overcomplicated designs. This was the case reported by Haldar et al. [30]. In the process of applying machine learning to AirBnB search, the team started with a complex deep learning model. The team was quickly overwhelmed by its complexity and ended up consuming development cycles. After several failed deployment attempts the neural network architecture was drastically simplified: a single hidden layer NN with 32 fully connected ReLU activations. Even such a simple model had value, as it allowed the building of a whole pipeline of deploying ML models in production setting, while providing reasonably good performance². Over time the model evolved, with a second hidden layer being added, but it still remained fairly simple, never reaching the initially intended level of complexity.

Another advantage that less complex models can offer is their relatively modest hardware requirements. This becomes a key decision point in resource constrained environments, as shown by Wagstaff et al. [31]. They worked on deploying ML models to a range of scientific instruments onboard Europa Clipper spacecraft. Spacecraft design is always a trade-off between the total weight, robustness and the number of scientific tools onboard. Therefore computational resources are scarce and their usage has to be as small as possible. These requirements naturally favor the models that are light on computational demands. The team behind Europa Clipper used machine learning for three anomaly detection tasks, some models took time series data as input and some models took images, and on all three occasions simple threshold or PCA based techniques were implemented. They were specifically chosen because of their robust performance and low demand on computational power.

A further example of a resource-constrained environment is wireless cellular networks, where energy, memory consumption and data transmission are very limited. Most advanced techniques, such as deep learning, are not considered yet for practical deployment, despite being able to handle highly dimensional mobile network data [32].

The ability to interpret the output of a model into understandable business domain terms often plays a critical role in model selection, and can even outweigh performance considerations. For that reason decision trees (DT), which can be considered a fairly basic ML algorithm, are widely used in practice. For example, Hansson et al. [33] describe several cases in manufacturing that adopt DT due to its high interpretability.

Banking is yet another example of an industry where DT finds extensive use. As an illustrative example, it is used by Keramati et al. [34] where the primary goal of the ML application is to predict customer churn by understanding if-then rules. While it is easy to imagine more complicated

²We discuss more benefits of setting up the automated deployment pipeline in Section 6.3

models learning the eventual input-output relationship for this specific problem, interpretability is key requirement here because of the need to identify the features of churners. The authors found DT to be the best model to fulfill this requirement.

Nevertheless, deep learning (DL) is commonly used for practical background tasks that require analysis a large amount of previously acquired data. This notion is exemplified by the field of unmanned aerial vehicles (UAV) [35]. Image sensors are commonplace in UAVs due to their low cost, low weight, and low power consumption. Consequently, processing images acquired from sensors is the main way of exploiting excellent capabilities in processing and presentation of raw data that DL offers. But computational resource demands still remain the main blocker for deploying DL as an online processing instrument on board of UAVs.

4.2 Training

One of the biggest concern with model training is the economic cost associated with carrying the training stage due to the computational resources required. This is certainly true in the field of natural language processing (NLP), as illustrated by Sharir et al. [36]. The authors observe that while the cost of individual floating-point operations is decreasing, the overall cost of training NLP is only growing. They took one of the state-of-the-art models in the field, BERT [37], and found out that depending on the chosen model size full training procedure can cost anywhere between \$50k and \$1.6m, which is unaffordable for most research institutions and even companies. The authors observe that training dataset size, number of model parameters and number of operations utilized by the training procedure are all contributing towards the overall cost. Of particular importance here is the second factor: novel NLP models are already using billions of parameters, and this number is expected to increase further in the nearest future [38].

A related concern is raised by Strubell et al. [39] regarding the impact the training of ML models has on the environment. By consuming more and more computational resources, ML models training is driving up the energy consumption and greenhouse gas emissions. According to the estimates provided in the paper, one full training cycle utilizing neural architecture search emits the amount of CO₂ comparable to what four average cars emit in their whole lifetime. The authors stress how important it is for researchers to be aware of such impact of model training, and argue that community should give higher priority to computationally efficient hardware and algorithms.

4.3 Hyper-parameter selection

In addition to parameters that are learned during the training process, many ML models also define several hyper-parameters. Hyper-parameter optimization (HPO) is the process of choosing the optimal set of these hyper-parameters. Most HPO techniques involve multiple training cycles of the ML model. Besides, the size of HPO task grows exponentially with each new hyper-parameter, because it adds a new dimension to the search space. As discussed by Yang and Shami [40], these considerations make HPO techniques very expensive and resource-heavy in practice, especially for applications of deep learning. Even approaches like Hyperband [41] or Bayesian optimization [42], that are specifically designed to minimize the number of training cycles needed, are still not able to deal with certain problems due to the complexity of the models or the size of the datasets.

HPO often needs to take into account specific requirements imposed by the environment where the model will run. This is exemplified by Marculescu et al. [43] in the context of hardware-aware ML. In order to deploy models to embedded and mobile devices, one needs to be well aware of energy and memory constraints imposed by such devices. This creates a need for customized hardware-aware optimization techniques that co-optimize for the accuracy of the model and the hardware efficiency.

5 Model Verification

The goal of the model verification stage is multifaceted, because an ML model should generalize well to unseen inputs, demonstrate reasonable handling of edge cases and overall robustness, as well as satisfy all functional requirements. In this section, we discuss issues concerning three steps within model verification: requirement encoding, formal verification and test-based verification.

5.1 Requirement encoding

Defining requirements for a machine learning model is a crucial prerequisite of testing activities. It often turns out that an increase in model performance does not translate into a gain in business value, as Booking.com discovered after deploying 150 models into production [44]. Therefore more specific metrics need to be defined and measured, such as KPIs and other business driven measures. In the case of Booking.com such metrics included conversion, customer service tickets or cancellations. Cross-disciplinary effort is needed to even define such metrics, as understanding from modeling, engineering and business angles is required. Once defined, these metrics are used for monitoring of the production environment and for quality control of model updates.

Besides, simply measuring the accuracy of the ML model is not enough to understand its performance. Essentially, performance metrics should reflect audience priorities. For instance Sato et al. [45] recommend validating models for bias and fairness, while in the case described by Wagstaff et al. [31] controlling for consumption of spacecraft resources is crucial.

5.2 Formal Verification

The formal verification step verifies that the model functionality follows the requirements defined within the scope of the project. Such verification could include mathematical proofs of correctness or numerical estimates of output error bounds, but as Ashmore et. al. [14] point out this rarely happens in practice. More often quality standards are being formally set via extensive regulatory frameworks.

An example of where ML solutions have to adhere to regulations is the banking industry [46]. This requirement was developed in the aftermath of the global financial crisis, as the industry realized that there was a need for heightened scrutiny towards models. As a consequence an increased level of regulatory control is now being applied to the processes that define how the models are built, approved and maintained. For instance, official guidelines has been published by the UK's Prudential Regulation Authority [47] and European Central Bank [48]. These guidelines require model risk frameworks to be in place for all business decision-making solutions, and implementation of such frameworks requires developers to have extensive tests suites in order to understand behavior of their ML models. The formal verification step in that context means ensuring that the model meets all criteria set by the corresponding regulations.

Regulatory frameworks share similarities with country-wide policies, which we discuss in greater details in Section 7.1.

5.3 Test-based Verification

Test-based verification is intended for ensuring that the model generalizes well to the previously unseen data. While collecting validation dataset is usually not a problem, as it can be derived from splitting the training dataset, it may not be enough for production deployment.

In an ideal scenario testing is done in a real-life setting, where business driven metrics can be observed, as we discussed in Section 5.1. Full scale testing in real-world environment can be challenging for a variety of safety, security and scale reasons, and is often substituted with testing in simulation. That is the case for models for autonomous vehicles control [26]. Simulations are cheaper, faster to run, and provide flexibility to create situations rarely encountered in real life. Thanks to these advantages, simulations are becoming prevalent in this field. However, it is important to remember that simulation-based testing hinges on assumptions made by simulation developers, and therefore cannot be considered a full replacement for real-world testing. Even small variations between simulation and real world can have drastic effects on the system behavior, and therefore the authors conclude that validation of the model and simulation environment alone is not enough for autonomous vehicles. This point is emphasized further by the experiences from the field of reinforcement learning [25], where use of simulations is a de-facto standard for training agents.

In addition, the dataset itself also needs to be constantly validated to ensure data errors do not creep into the pipeline and do not affect the overall quality. Breck et al. [49] argue that one of the most common scenarios when issues in data can go unnoticed is the setup where data generation is decoupled from the ML pipeline. There could be multiple reasons for such issues to appear, including bugs in code, feedback loops, changes in data dependencies. Data errors can propagate

and manifest themselves at different stages of the pipeline, therefore it is imperative to catch them early by including data validation routines into the ML pipeline.

6 Model Deployment

Machine learning systems running in production are complex software systems that have to be maintained over time. This presents developers with another set of challenges, some of which are shared with running regular software services, and some are unique to ML.

There is a separate discipline in engineering, called DevOps, that focuses on techniques and tools required to successfully maintain and support existing production systems. Consequently, there is a necessity to apply DevOps principles to ML systems. However, even though some of the DevOps principles apply directly, there is also a number of challenges unique to productionizing machine learning. This is discussed in detail by Dang et al. [50] which uses the term AIOps for DevOps tasks for ML systems. Some of the challenges mentioned include lack of high quality telemetry data as well as no standard way to collect it, difficulty in acquiring labels which makes supervised learning approaches inapplicable³ and lack of agreed best practices around handling of machine learning models. In this section, we discuss issues concerning three steps within model deployment: integration, monitoring and updating.

6.1 Integration

The model integration step constitutes of two main activities: building the infrastructure to run the model and implementing the model itself in a form that can be consumed and supported. While the former is a topic that belongs almost entirely in systems engineering and therefore lies out of scope of this work, the latter is of interest for our study, as it exposes important aspects at the intersection of ML and software engineering. In fact, many concepts that are routinely used in software engineering are now being reinvented in the ML context.

Code reuse is a common topic in software engineering, and ML can benefit from adopting the same mindset. Reuse of data and models can directly translate into savings in terms of time, effort or infrastructure. An illustrative case is the approach Pinterest took towards learning image embeddings [51]. There are three models used in Pinterest internally which use similar embeddings, and initially they were maintained completely separately, in order to make it possible to iterate on the models individually. However, this created engineering challenges, as every effort in working with these embeddings had to be multiplied by three. Therefore the team decided to investigate the possibility of learning universal set of embeddings. It turned out to be possible, and this reuse ended up simplifying their deployment pipelines as well as improving performance on individual tasks.

A broad selection of engineering problems that machine learning practitioners now face is given in Sculley et al. [52]. Most of them are considered anti-patterns in engineering, but are currently widespread in machine learning software. Some of these issues, such as abstraction boundaries erosion and correction cascades, are caused by the fact that ML is used in cases where the software has to take explicit dependency on external data. Others, such as glue code or pipeline jungles, stem from the general tendency in the field to develop general-purpose software packages. Yet another source of problems discussed in the paper is the configuration debt, which is caused by the fact that ML systems, besides all configurations a regular software system may require, add a sizable number of ML-specific configuration settings that have to be set and maintained.

Researchers and software engineers often find themselves working together on the same project aiming to reach a business goal with a machine learning approach. On surface there seems to be a clear separation of responsibilities: researchers produce the model while engineers build infrastructure to run it. In reality, their areas of concern often overlap when considering the development process, model inputs and outputs and performance metrics. Contributors in both roles often work on the same code. Thus it is beneficial to loop researchers into the whole development journey, making sure they own the product code base along with the engineers, use the same version control and participate in code reviews. Despite obvious onboarding and slow-start challenges, this approach was seen to bring long term benefits in terms of speed and quality of product delivery [12].

³Please refer to Section 3.3 for detailed discussion about data labeling.

6.2 Monitoring

Monitoring is one of the issues associated with maintaining machine learning systems as reported by Sculley et al. [52]. The community is in the early stages of understanding what are the key metrics of data and models to monitor and how to alarm on them. Monitoring of evolving input data, prediction bias and overall performance of ML models is an open problem. Another maintenance issue highlighted by this paper that is specific to data-driven decision making is feedback loops. ML models in production can influence their own behavior over time via regular retraining. While making sure the model stays up to date, it is possible to create feedback loop where the input to the model is being adjusted to influence its behavior. This can be done intentionally, as well as happen inadvertently which is a unique challenge when running live ML systems.

Klaise et al. [53] point out the importance of outlier detection as a key instrument to flag model predictions that cannot be used in a production setting. The authors name two reasons for such predictions to occur: the inability of the models to generalize outside of the training dataset and also overconfident predictions on out-of-distribution instances due to poor calibration. Deployment of the outlier detector can be a challenge in its own right, because labeled outlier data is scarce, and the detector training often becomes a semi-supervised or even an unsupervised problem.

Additional insight on monitoring of ML systems can be found in Ackermann et al. [54]. This paper describes an early intervention system (EIS) for two police departments in the US. On the surface their monitoring objectives seem completely standard: data integrity checks, anomaly detection and performance metrics. One would expect to be able to use out-of-the-box tooling for these tasks. However, the authors explain that they had to build all these checks from scratch in order to maintain good model performance. For instance, the data integrity check meant verifying updates of a certain input table and checksums on historical records, performance metric was defined in terms of the number of changes in top k outputs, and anomalies were tracked on rank-order correlations over time. All of these monitoring tools required considerable investigation and implementation. This experience report highlights a common problem with currently available end-to-end ML platforms: the final ML solutions are usually so sensitive to problem's specifics that out-of-the-box tooling does not fit their needs well.

As a final remark we note that there is an overlap between choice of metrics for monitoring and validation. The latter topic is discussed in Section 5.1

6.3 Updating

Once the initial deployment of the model is completed, it is often necessary to be able to update the model later on in order to make sure it always reflects the most recent trends in data and the environment. There are multiple techniques for adapting models to a new data, including scheduled regular retraining and continual learning [55]. Nevertheless in production setting model updating is also affected by practical considerations.

A particularly important problem that directly impacts the quality and frequency of model update procedure is the concept drift. Concept drift in ML is understood as changes observed in joint distribution $p(X, y)$, where X is the model input and y is the model output. Undetected, this phenomenon can have major adverse effects on model performance, as is shown by Jameel et al. [56] for classification problems or by Celik and Vanschoren [57] in AutoML context. Concept drift can arise due to a wide variety of reasons. For example, the finance industry faced turbulent changes as the financial crisis of 2008 was unfolding, and if advanced detection techniques were employed it could have provided additional insights into the ongoing crisis, as explained by Masegosa et al. [58]. Changes in data can also be caused by inability to avoid fluctuations in the data collection procedure, as described in paper Langenkämper et al. [59] which studies the effects of slight changes in marine images capturing gear and location on deep learning models' performance. Data shifts can have noticeable consequences even when occurring at microscopic scale, as Zenisek et al. [60] show in their research on predictive maintenance for wear and tear of industrial machinery. Even though concept drift has been known for decades [61], these examples show that it remains a critical problem for applications of ML today.

On top of the question of when to retrain the model to keep it up to date, there is an infrastructural question on how to deliver the model artifact to the production environment. In software engineering such tasks are commonly solved with continuous delivery (CD), which is an approach for

accelerating development cycle by building an automated pipeline for building, testing and deploying software changes. CD for machine learning solutions is complicated because, unlike in regular software products where changes only happen in the code, ML solutions experience change along three axis: the code, the model and the data. An attempt to formulate CD for ML as a separate discipline can be seen in Sato et al. [45]. This work describes the pieces involved and the tools that can be used at each step of building the full pipeline. A direct illustration of benefits that a full CD pipeline can bring to the real-life ML solution can be found in Wider and Deger [62].

7 Cross-cutting aspects

In this section we describe three additional aspects that ML projects have to consider: ethics, end users' trust and security. These aspects can affect every stage of the deployment pipeline.

7.1 Ethics

Ethical considerations should always inform data collection activities. As stated in the report on ethical AI produced by the Alan Turing Institute [63], “it is essential to establish a continuous chain of human responsibility across the whole AI project delivery workflow”. If researchers and developers do not follow this recommendation, complications may come up due to a variety of reasons: breach of governmental regulations, unjustifiable outcomes, aggravation of existing issues, and more [63].

Various countries have produced regulations to protect personal data rights. The more sensitive is the information collected from the individual, the severer are the regulations governing its use. And of course industry that deals with some of the most sensitive information is healthcare. According to Han et al. [64], many countries have strict laws in place to protect data of patients, which makes adoption of ML in healthcare particularly difficult. Examples of such regulations include the General Data Protection Regulation in European Union [65] and ethical screening laws in a range of Asian countries [66]. On one hand there is no doubt that these rules are absolutely necessary to make sure people are comfortable with their data being used. On the other hand amount of reviews, software updates and cycles of data collection/annotation that is required makes it exceptionally hard to keep up with technical advances in ML, as Han et al. [64] explain following their experience deploying ML solutions in healthcare sector in Japan.

Companies should not be focusing solely on the technological side of their solutions, as DeepMind and Royal Free NHS Foundation Trust have discovered while working on Streams, an application for automatic review of test results for serious conditions [67]. Their initial collaboration was not specific enough on the use of patient data and on patient involvement overall, which triggered an investigation on their compliance with data protection regulations. The revised collaboration agreement was far more comprehensive and included patient and public engagement strategy in order to ensure data is being used ethically.

Since ML models use previously seen data to make decisions, they can worsen issues that already exist in data. This effect in the field of criminal justice is discussed in detail by O’Neil [68]. Models that calculate person’s criminal “risk score” are often marketed as a way to remove human bias. Nevertheless, they use seemingly neutral demographic information that often ends up serving as a proxy. As a result, people are disadvantaged on the basis of race or income.

Likewise, Soden et al. [69] mention aggravation of social inequalities through use of biased training datasets as one of the main concerns in applying ML to Disaster Risk Management (DRM) field. Furthermore, it is argued that ML causes privacy and security concerns in Fragility, Conflict and Violence settings through combination of previously distinct datasets. Reducing role of both experts and general public is also seen as an ethical issue by DRM professionals. Some of these issues are studied by the branch of machine learning known as Fairness in ML [70]. We discuss related cross-cutting security concerns in Section 7.3.

An interesting ethical aspect arises in usage of ML in the field of creative arts, discussed by Anantrasirichai and Bull [71]. When a trained model is used to create a piece of visual art, it is not entirely clear where the authorship of this piece resides. The questions of originality therefore requires a special attention. Closely related with this question is the growing concern of fake content being generated with ML, which can be easily used for the wrong purposes [72].

Another facet of ethical issues encountered by machine learning is the data based decision making. As machine learning tools become utilized in critical decision making processes ethical concerns with their usage grow. Illustrative note is made by Muthiah et al. [73]. Their system of predicting civil unrest, called EMBERS, is designed to be used as a forecasting and communication tool, however authors remark that it can also be potentially misused by governments, either due to misunderstanding of its role in society, or intentionally.

7.2 End users' trust

ML is often met cautiously by the end users [74], [3], [75]. On their own accord models provide minimal explanations, which makes it difficult to persuade end users in their utility [64]. In order to convince users to trust ML based solutions, time has to be invested to build that trust. In this section, we explore ways in which that is done in practice.

If an application has a well-defined accessible audience, getting these people involved early in the project is an efficient way to foster their confidence in the end product. This approach is very common in medicine, because the end product is often targeted at a well defined group of healthcare workers. One example is the project called Sepsis Watch [76]. In this project the goal was to build a model that estimates patient's risk of developing sepsis. It was not the first attempt at automating this prediction, and since previous attempts were considered failures, medical personnel were skeptical about eventual success of Sepsis Watch. To overcome this skepticism, the team prioritized building trust, by:

- Building strong communication channels;
- Sharing with stakeholders progress towards developing the goal instead of showing technical advances;
- Establishing mechanisms for public and external accountability;
- Engaging both front-line clinicians and enterprise-level decision makers at early stages of the project.

One of the key messages of this work is that model interpretability has limits as a trust-building tool, and other ways to achieve high credibility with the end users should be considered. While it may sound controversial, in fact it aligns with conclusions made by "Project explAIn" which found that relative importance of explanations of AI decisions varies by context [77].

Similar argument is made by Soden et al. [69], who explore an impact ML has on disaster risk management (DRM). Due to growing complexity of the ML solutions deployed, it is becoming increasingly hard for public to participate and consequently to trust the ML-based DRM services, such as flooding area estimates or prediction of damage from a hurricane. As a mitigation measure the authors recommend making development of these solutions as transparent as possible, by taking into account voice of residents in the areas portrayed by models as "at risk" and relying on open software and data whenever possible.

A reasonable approach towards building trustworthy products is investing time in specialised user interfaces with tailored user experience. Developers of Firebird [22], a system that helps identify priority targets for fire inspection in the city of Atlanta, USA, found that the best way to introduce ML solution as a replacement of the previously used pen-and-paper method was to develop a user interface that presented the results of modelling in a way that the end users (fire officers and inspectors in the Fire dept) found most useful and clear. Similar experience is reported by Ackermann et al. [54].

Authors of EMBERS [73], a system that forecasts population-level events (such as protest), in Latin America, noticed that their users have two modes of using the system: (a) high recall: obtain most events, and then filter them using other methods; (b) high precision: focus on specific area or specific hypothesis. To improve the user experience and thus increase their confidence in the product, user interface was improved to easily support both modes. This case study emphasizes the importance of context-aware personalization for ML systems' interfaces, one of the key observations delivered by Project explAIn [77].

Budd et al. [24] show that interface design directly impacts quality of applications built to collect annotations for unlabeled data. They discuss a range of projects that collected labels for medical

images, all of which benefited from well designed user interface. The authors conclude that end user interface plays a large part in overall success of the annotation applications.

Finally, the solutions based on models whose decisions can be explained are preferred when the target audience has experience and an understanding of ML. Bhatt et al. [11] analyzed explainability as a feature of machine learning models deployed within enterprises, and found that it is a must-have requirement for most of the stakeholders, including executives, ML engineers, regulators, and others. Moreover, their survey showed that explainability score is a desired model metric, along with measures of fairness and robustness.

7.3 Security

Machine Learning opens up new threat vectors across the whole ML deployment workflow as described by Kumar et al. [78]. Specialised adversarial attacks for ML can occur on the model itself, the data used for training but also the resulting predictions. The field of adversarial machine learning studies the effect of such attacks against ML models and how to protect against them [79, 80]. Recent work from Kumar et al. found that industry practitioners are not equipped to protect, detect and respond to attacks on their ML systems [81]. In this section, we describe the three most common attacks reported in practice which affects deployed ML models: data poisoning, model stealing and model inversion. We focus specifically on adversarial machine learning and consider other related general security concerns in deploying systems such as access control and code vulnerabilities beyond the scope of our work.

In data poisoning, the goal of the adversarial attack is to deliberately corrupt the integrity of the model during the training phase in order to manipulate the produced results. Poisoning attacks are particularly relevant in situations where the machine learning model is continuously updated with new incoming training data. Jagielski et al. reported that in a medical setting using a linear model, the introduction of specific malicious samples with a 8% poisoning rate in the training set resulted in incorrect dosage for half of the patients [82].

Data poisoning can also occur as a result of a coordinated collective effort that exploits feedback loops we have discussed in Section 6.2 as it happened with Microsoft’s Twitter bot Tay [83]. Tay was designed to improve its understanding of the language over time, but was quickly inundated with a large number of deliberately malevolent tweets. Within 16 hours of its release a troubling percentage of Tay’s messages were abusive or offensive, and the bot was taken down.

Another type of adversarial attack is reverse engineering a deployed model by querying its inputs (e.g. via a public prediction API) and monitoring the outputs. The adversarial queries are crafted to maximize the extraction of information about the model in order to train a substitute model. This type of attack is referred to as model stealing. In a nutshell, this attack results in loss of intellectual property which could be a key business advantage for the defender. Tramèr et al. [84] have shown that it is possible to replicate models deployed in production from ML services offered by Google, Amazon and Microsoft across a range of ML algorithms including logistic regression, decision trees, SVMs and neural networks. In their work, they report the number of queries ranging from 650 to 4013 to extract an equivalent model and in time ranging from 70s to 2088s.

A related attack is that of model inversion where the goal of the adversarial attack is recover parts of the private training set, thereby breaking its confidentiality. Fredrikson et al. have shown that they could recover training data by exploiting models that report confidence values along their predictions [85]. Veale et al. [86] emphasize the importance of protecting against model inversion attacks as a critical step towards compliance with data protection laws such as GDPR.

8 Discussion of potential solutions

This survey looked at case studies from a variety of industries: computer networks, manufacturing, space exploration, law enforcement, banking, and more. However, further growths of ML adoption can be severely hindered by poor deployment experience. To make the ML deployment scalable and accessible to every business that may benefit from it, it is important to understand the most critical pain points and to provide tools, services and best practices that address those points. We see this survey as an initial step in this direction: by recognizing the most common challenges currently being reported we hope to foster an active discussion within the academic community about what

possible solutions might be. We classify possible research avenues for solutions into two categories, which we discuss below.

8.1 Tools and services

The market for machine learning tools and services is experiencing rapid growth [87]. As a result, tools for individual deployment problems are continuously developed and released. For some of the problems we have highlighted, making use of the right specific tool is an entirely reasonable approach.

For example, this is most likely the case for operational maintenance of ML models. Many platforms on the market offer end-to-end experience for the user, taking care of such things as data storage, retraining and deployment. Examples include AWS SageMaker [88], Microsoft AzureML [89], Uber Michelangelo [90], TensorFlow TFX [91], MLflow [92] and more. A typical ML platform would include, among other features, data storage facility, model hosting with APIs for training and inference operations, a set of common metrics to monitor model health and an interface to accept custom changes from the user. By offering managed infrastructure and a range of out-of-the-box implementations for common tasks such platforms greatly reduce operational burden associated with maintaining the ML model in production.

Quality assurance also looks to be an area where better tools can be of much assistance. As mentioned in Section 3, data integrity plays a big part in quality control, and is an active branch of research [18]. Models themselves can also greatly benefit from development of a test suite to verify their behavior. This is normally done by trial and error, but more formal approaches are being developed, as is the case with Checklist methodology for NLP models [93].

As discussed in Section 3.3, obtaining labels is often a problem with real world data. Weak supervision has emerged as a separate field of ML which looks for ways to address this challenge. Consequently, a number of weak supervision libraries are now actively used within the community, and show promising results in industrial applications. Some of the most popular tools include Snorkel [94], Snuba [95] and cleanlab [96].

Using specific tools for solving individual problems is a straightforward approach. However practitioners need to be aware that by using a particular tool they introduce an additional dependency into their solution. While a single additional dependency seems manageable, their number can quickly grow and become a maintenance burden. Besides, as we mentioned above, new tools for ML are being released constantly, thus presenting practitioners with the dilemma of choosing the right tool by learning its strength and shortcomings.

8.2 Holistic approaches

Even though ML deployments require a certain amount of software development, ML projects are fundamentally different from traditional software engineering projects, and we argue they need a different approach. The main differences arise from unique activities like data discovery, dataset preparation, model training, deployment success measurement etc. Some of these activities cannot be defined precisely enough to have a reliable time estimate, some assume huge potential risks, and some make it difficult to measure the overall added value of the project. Therefore ML deployments do not lend themselves well to widespread approaches to software engineering management paradigms, and neither to common software architectural patterns.

Data Oriented Architectures (DOA, [97], [98]) is an example of an idea that suggests rethinking how things are normally approached in software development, and by doing so promises to solve quite a few issues we have discussed in this survey. Specifically, the idea behind DOA is to consider replacing micro-service architecture, widespread in current enterprise systems, with streaming-based architecture, thus making data flowing between elements of business logic more explicit and accessible. Micro-service architectures have been successful in supporting high scalability and embracing the single responsibility principle. However, they also make data flows hard to trace, and it is up to owners of every individual service to make sure inputs and outputs are being stored in a consistent form. DOA provides a solution to this problem by moving data to streams flowing between stateless execution nodes, thus making data available and traceable by design, therefore making simpler the tasks of data discovery, collection and labeling. In its essence DOA proposes to acknowledge

that modern systems are often data-driven, and therefore need to prioritize data in their architectural principles.

As noted above, ML projects normally do not fit well with commonly used management processes, such as Scrum or Waterfall. Therefore it makes sense to consider processes tailored specifically for ML. One such attempt is done by Lavin et. al. [99], who propose Technology Readiness Levels for ML (TRL4ML) framework. TRL4ML describes a process of producing robust ML systems that takes into account key differences between ML and traditional software engineering.

A very widespread practice in software engineering is to define a set of guidelines and best practices to help developers make decisions at various stages of the development process. These guidelines can cover a wide range of questions, from variable names to execution environment setup. Similarly, Zinkevich [100] compiled a collection of best practices for machine learning that are utilized in Google. While this cannot be viewed as a coherent paradigm towards doing ML deployment, this document gives practical advice on a variety of important aspects that draw from the real life experiences of engineers and researchers in the company.

Holistic approaches are created with ML application in mind, and therefore they have the potential to offer a significant ease of deploying ML. But it should be noted that all such approaches assume a big time investment, because they represent significant changes to current norms in project management and development. Therefore a careful assessment of risks versus benefits should be carried out before adopting any of them.

9 Further Work

Even though the set of challenges we reviewed covers every stage of the ML deployment workflow, we believe that it is far from being complete. Identifying other, especially non-technical, challenges is a natural direction of further work and could be augmented by conducting interviews with industry representatives about their experiences of deploying ML.

In this paper, we reviewed reports from a variety of industries, which shows the ubiquity and variety of challenges with deploying ML in production. An interesting extension can be the comparative analysis of industries. Quantitative and qualitative analysis of most commonly reported challenges may open interesting transferability opportunities, as approaches developed in one field may be applicable in the other.

Our work includes a brief discussion of existing tools in Section 8.1. However, the community would benefit from a comprehensive review of currently available tools and services, mapped to challenges reported in our study. This new work could be combined with our survey to enable practitioners to identify the problem they are facing and choose the most appropriate tool that addresses that problem.

10 Conclusion

In this survey, we showed that practitioners deal with challenges at every step of the ML deployment workflow due to practical considerations of deploying ML in production. We discussed challenges that arise during the data management, model learning, model verification and model deployment stages, as well as considerations that affect the whole deployment pipeline including ethics, end users' trust and security. We illustrated each stage with examples across different fields and industries by reviewing case studies, experience reports and the academic literature.

We argue that it is worth academic community's time and focus to think about these problems, rather than expect each applied field to figure out their own approaches. We believe that ML researchers can drive improvements to the ML deployment experience by exploring holistic approaches and taking into account practical considerations.

As an observation that follows from the process of collecting papers to review in this survey, we note the shortage of deployment experience reports in the academic literature. Valuable knowledge obtained by industry ML practitioners goes unpublished. We would like to encourage organisations to prioritize sharing such reports, as they provide valuable information for the wider community, but also as a way to self-reflect, collect feedback and improve on their own solutions.

We hope this survey will encourage discussions within the academic community about pragmatic approaches to deploying ML in production.

Acknowledgments and Disclosure of Funding

We would like to thank our reviewers for their thoughtful comments and suggestions. We are also grateful to Jessica Montgomery, Diana Robinson and Ferenc Huszar for discussions that helped shape this work.

References

- [1] Arif Cam, Michael Chui, and Bryce Hall. Global AI survey: AI proves its worth, but few scale impact. *McKinsey Analytics*, 2019.
- [2] Thomas H. Davenport and Rajeev Ronanki. Artificial intelligence for the real world. *Harvard business review*, 96(1):108–116, 2018.
- [3] Royal Society (Great Britain). *Machine Learning: The Power and Promise of Computers that Learn by Example: an Introduction*. Royal Society, 2017.
- [4] Michal Pěchouček and Vladimír Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous agents and multi-agent systems*, 17(3):397–431, 2008.
- [5] Kyle Wiggers. Algorithmia: 50% of companies spend between 8 and 90 days deploying a single AI model, 2019. Available at <https://venturebeat.com/2019/12/11/algorithmia-50-of-companies-spend-upwards-of-three-months-deploying-a-single-ai-model/>
- [6] Lawrence E. Hecht. Add it up: How long does a machine learning deployment take?, 2019. Available at <https://thenewstack.io/add-it-up-how-long-does-a-machine-learning-deployment-take/>
- [7] Kyle Wiggers. IDC: For 1 in 4 companies, half of all AI projects fail, 2019. Available at <https://venturebeat.com/2019/07/08/idc-for-1-in-4-companies-half-of-all-ai-projects-fail/>
- [8] Ben Lorica and Nathan Paco. *The State of Machine Learning Adoption in the Enterprise*. O'Reilly Media, 2018.
- [9] The state of development and operations of AI applications. *Dotsscience*, 2019. Available at https://dotsscience.com/assets/downloads/Dotsscience_Survey_Report_2019.pdf
- [10] Artificial intelligence and machine learning projects are obstructed by data issues. *dimensional research*, 2019. Available at <https://telecomreseller.com/wp-content/uploads/2019/05/EMBARGOED-UNTIL-800-AM-ET-0523-Dimen>
- [11] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José M. F. Moura, and Peter Eckersley. Explainable machine learning in deployment, 2019.
- [12] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.
- [13] Lucas Baier, Fabian Jöhren, and Stefan Seebacher. Challenges in the deployment and operation of machine learning in practice. 2019.
- [14] Rob Ashmore, Radu Calinescu, and Colin Paterson. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *arXiv preprint arXiv:1905.04223*, 2019.
- [15] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data lifecycle challenges in production machine learning: a survey. *ACM SIGMOD Record*, 47(2):17–28, 2018.
- [16] Jimmy Lin and Dmitriy Ryaboy. Scaling big data mining infrastructure: the Twitter experience. *Acm SIGKDD Explorations Newsletter*, 14(2):6–19, 2013.

- [17] Robert C. Martin. The single responsibility principle. *The principles, patterns, and practices of Agile Software Development*, pages 149–154, 2002.
- [18] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*, 9(12):993–1004, 2016.
- [19] Rajashree Y Patil and RV Kulkarni. A review of data cleaning algorithms for data warehouse systems. *International Journal of Computer Science and Information Technologies*, 3(5):5212–5214, 2012.
- [20] Fakhitah Ridzuan and Wan Mohd Nazmee Wan Zainon. A review on data cleansing methods for big data. *Procedia Computer Science*, 161:731–738, 2019.
- [21] Neil D Lawrence. Data readiness levels. *arXiv preprint arXiv:1705.02245*, 2017.
- [22] Michael Madaio, Shang-Tse Chen, Oliver L. Haimson, Wenwen Zhang, Xiang Cheng, Matthew Hinds-Aldrich, Duen Horng Chau, and Bistra Dilkina. Firebird: Predicting fire risk and prioritizing fire inspections in Atlanta. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 185–194, 2016.
- [23] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys Tutorials*, 21(2):1988–2014, 2019.
- [24] Samuel Budd, Emma C. Robinson, and Bernhard Kainz. A survey on active learning and human-in-the-loop deep learning for medical image analysis, 2019.
- [25] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning, 2019.
- [26] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control, 2019.
- [27] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, 2012.
- [28] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering*, 44(11):1024–1038, 2017.
- [29] Diego Charrez. NeurIPS 2019 stats, 2019. Available at <https://medium.com/@dcharrezt/neurips-2019-stats-c91346d31c8f>.
- [30] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and et al. Applying deep learning to AirBnB search. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Jul 2019.
- [31] Kiri L. Wagstaff, Gary Doran, Ashley Davies, Saadat Anwar, Srija Chakraborty, Marissa Cameron, Ingrid Daubar, and Cynthia Phillips. Enabling onboard detection of events of scientific interest for the Europa Clipper spacecraft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’19, page 2191–2201, New York, NY, USA, 2019. Association for Computing Machinery.
- [32] Ursula Challita, Henrik A. Ryden, and Hugo Tullberg. When machine learning meets wireless cellular networks: Deployment, challenges, and applications, 2019.
- [33] Karl Hansson, Siril Yella, Mark Dougherty, and Hasan Fleyeh. Machine learning algorithms in heavy process manufacturing. *American Journal of Intelligent Systems*, 6(1):1–13, 2016.
- [34] Abbas Keramati, Hajar Ghaneei, and Seyed Mohammad Mirmohammadi. Developing a prediction model for customer churn from electronic banking services using data mining. *Financial Innovation*, 2(1):10, 2016.
- [35] Adrian Carrio, Carlos Sampedro, Alejandro Rodriguez-Ramos, and Pascual Campoy. A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors*, 2017, 2017.
- [36] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training NLP models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

- [37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [38] Nathan Benaich and Ian Hogarth. State of AI report 2020. 2020. Available at www.stateof.ai
- [39] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*, 2019.
- [40] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.
- [41] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.
- [42] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.
- [43] Diana Marculescu, Dimitrios Stamoulis, and Ermao Cai. Hardware-aware machine learning: Modeling and optimization. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD ’18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [44] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. 150 successful machine learning models: 6 lessons learned at Booking.com. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1743–1751, 2019.
- [45] Danilo Sato, Arif Wider, and Christoph Windheuser. Continuous delivery for machine learning, 2019. Available at <https://martinfowler.com/articles/cd4ml.html>.
- [46] Ram Ananth, Seph Mard, and Peter Simon. Opening the “black box”: The path to deployment of AI models in banking, white paper. *DataRobot and REPLY AVANTAGE*, 2019.
- [47] Prudential Regulation Authority. Model risk management principles for stress testing, 2018.
- [48] ECB TRIM Guide. Guide for the targeted review of internal models (TRIM). *European Central Bank*, 2017.
- [49] Eric Breck, Marty Zinkevich, Neoklis Polyzotis, Steven Whang, and Sudip Roy. Data validation for machine learning. In *Proceedings of SysML*, 2019.
- [50] Yingnong Dang, Qingwei Lin, and Peng Huang. AIOps: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5. IEEE, 2019.
- [51] Andrew Zhai, Hao-Yu Wu, Eric Tzeng, Dong Huk Park, and Charles Rosenberg. Learning a unified embedding for visual search at Pinterest, 2019.
- [52] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pages 2503–2511, 2015.
- [53] Janis Klaise, Arnaud Van Looveren, Clive Cox, Giovanni Vacanti, and Alexandru Coca. Monitoring and explainability of models in production. *arXiv preprint arXiv:2007.06299*, 2020.
- [54] Klaus Ackermann, Joe Walsh, Adolfo De Unánue, Hareem Naveed, Andrea Navarrete Rivera, Sun-Joo Lee, Jason Bennett, Michael Defoe, Crystal Cody, Lauren Haynes, et al. Deploying machine learning models for public policy: A framework. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 15–22, 2018.
- [55] Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil Lawrence. Continual learning in practice. *arXiv preprint arXiv:1903.05202*, 2019.
- [56] Syed Muslim Jameel, Manzoor Hashmani, Hitham Alhussian, Mobashar Rehman, and Arif Budiman. A critical review on adverse effects of concept drift over machine learning classification models. *International Journal of Advanced Computer Science and Applications*, 11, 01 2020.

- [57] Bilge Celik and Joaquin Vanschoren. Adaptation strategies for automated machine learning on evolving data. *arXiv preprint arXiv:2006.06480*, 2020.
- [58] Andrés R Masegosa, Ana M Martínez, Darío Ramos-López, Helge Langseth, Thomas D Nielsen, and Antonio Salmerón. Analyzing concept drift: A case study in the financial sector. *Intelligent Data Analysis*, 24(3):665–688, 2020.
- [59] Daniel Langenkämper, Robin van Kevelaer, Autun Purser, and Tim W Nattkemper. Gear-induced concept drift in marine images and its effect on deep learning classification. *Frontiers in Marine Science*, 7:506, 2020.
- [60] Jan Zenisek, Florian Holzinger, and Michael Affenzeller. Machine learning based concept drift detection for predictive maintenance. *Computers & Industrial Engineering*, 137:106031, 2019.
- [61] Jeffrey C Schlimmer and Richard H Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.
- [62] Arif Wider and Christian Deger. Getting smart: Applying continuous delivery to data science to drive car sales, 2017. Available at <https://www.thoughtworks.com/insights/blog/getting-smart-applying-continuous-delivery-data-s>
- [63] David Leslie. Understanding artificial intelligence ethics and safety. *arXiv preprint arXiv:1906.05684*, 2019.
- [64] Changhee Han, Leonardo Rundo, Kohei Murao, Takafumi Nemoto, and Hideki Nakayama. Bridging the gap between AI and Healthcare sides: towards developing clinically relevant AI-powered diagnosis systems, 2020.
- [65] John Mark Michael Rumbold and Barbara Pierscionek. The effect of the general data protection regulation on medical research. *Journal of medical Internet research*, 19(2):e47, 2017.
- [66] Syed Mohamed Aljunid, Samrit Srithamrongsawat, Wen Chen, Seung Jin Bae, Raoh-Fang Pwu, Shunya Ikeda, and Ling Xu. Health-care data collecting, sharing, and using in Thailand, China mainland, South Korea, Taiwan, Japan, and Malaysia. *Value in Health*, 15(1):S132–S138, 2012.
- [67] Mustafa Suleyman and Dominic King. The Information Commissioner, the Royal Free, and what we've learned, 2017.
- [68] Cathy O’Neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Broadway Books, 2016.
- [69] Robert Soden, Dennis Wagenaar, Dave Luo, and Annegien Tijssen. Taking ethics, fairness, and bias seriously in machine learning for disaster risk management, 2019.
- [70] Solon Barocas, Moritz Hardt, and Arvind Narayanan. Fairness in machine learning. *NIPS Tutorial*, 1, 2017.
- [71] Nantheera Anantrasirichai and David Bull. Artificial intelligence in the creative industries: A review, 2020.
- [72] Yisroel Mirsky and Wenke Lee. The creation and detection of deepfakes: A survey. *arXiv preprint arXiv:2004.11138*, 2020.
- [73] Sathappan Muthiah, Patrick Butler, Rupinder Paul Khandpur, Parang Saraf, Nathan Self, Alla Rozovskaya, Liang Zhao, Jose Cadena, Chang-Tien Lu, Anil Vullikanti, et al. Embers at 4 years: Experiences operating an open source indicators forecasting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 205–214, 2016.
- [74] M-C Lai, M Brian, and M-F Mamzer. Perceptions of artificial intelligence in healthcare: findings from a qualitative survey study among actors in France. *Journal of Translational Medicine*, 18(1):1–13, 2020.
- [75] Ramprakash Ramamoorthy, P Satya Madhuri, and Malini Christina Raj. AI from labs to production - challenges and learnings. Santa Clara, CA, May 2019. USENIX Association.
- [76] Mark Sendak, Madeleine Clare Elish, Michael Gao, Joseph Futoma, William Ratliff, Marshall Nichols, Armando Bedoya, Suresh Balu, and Cara O’Brien. “The human body is a black box”: Supporting clinical decision-making with deep learning. In *Proceedings of the 2020*

- Conference on Fairness, Accountability, and Transparency*, FAT* ’20, page 99–109, New York, NY, USA, 2020. Association for Computing Machinery.
- [77] Information Commissioner’s Office. Project ExplAIn interim report, 2019. Available at <https://ico.org.uk/about-the-ico/research-and-reports/project-explain-interim-report/>.
 - [78] Ram Shankar Siva Kumar, David O’Brien, Kendra Albert, Salomé Viljöen, and Jeffrey Snover. Failure modes in machine learning systems. *arXiv preprint arXiv:1911.11034*, 2019.
 - [79] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.
 - [80] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.
 - [81] Ram Shankar Siva Kumar, Magnus Nystrom, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. Adversarial machine learning-industry perspectives. Available at SSRN 3532474, 2020.
 - [82] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. IEEE, 2018.
 - [83] Oscar Schwartz. In 2016 Microsoft’s racist chatbot revealed the dangers of online conversation. *IEEE Spectrum*, 11, 2019.
 - [84] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.
 - [85] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.
 - [86] Michael Veale, Reuben Binns, and Lilian Edwards. Algorithms that remember: model inversion attacks and data protection law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2133):20180083, 2018.
 - [87] Chip Huyen. What I learned from looking at 200 machine learning tools, 2020. Available at <https://huyenchip.com/2020/06/22/mlops.html>.
 - [88] Kumar Venkateswar. Using Amazon SageMaker to operationalize machine learning. 2019.
 - [89] AML Team. AzureML: Anatomy of a machine learning service. In *Conference on Predictive APIs and Apps*, pages 1–13, 2016.
 - [90] Jeremy Hermann and Mike Del Balso. Meet Michelangelo: Uber’s machine learning platform. URL <https://eng.uber.com/michelangelo>, 2017.
 - [91] Denis Baylor, Kevin Haas, Konstantinos Katsiapis, Sammy Leong, Rose Liu, Clemens Menwald, Hui Miao, Neoklis Polyzotis, Mitchell Trott, and Martin Zinkevich. Continuous training for production ML in the TensorFlow extended (TFX) platform. In *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*, pages 51–53, 2019.
 - [92] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.
 - [93] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Association for Computational Linguistics (ACL)*, 2020.
 - [94] Stephen H Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, et al. Snorkel DryBell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 International Conference on Management of Data*, pages 362–375, 2019.
 - [95] Paroma Varma and Christopher Ré. Snuba: Automating weak supervision to label training data. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 12, page 223. NIH Public Access, 2018.
 - [96] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. Confident learning: Estimating uncertainty in dataset labels, 2019.

- [97] Neil Lawrence. Modern data oriented programming, 2019. Available at <http://inverseprobability.com/talks/notes/modern-data-oriented-programming.html>.
- [98] Tom Borchert. Milan: An evolution of data-oriented programming, 2020. Available at <https://tborchertblog.wordpress.com/2020/02/13/28/>.
- [99] Alexander Lavin and Gregory Renard. Technology readiness levels for machine learning systems. *arXiv preprint arXiv:2006.12497*, 2020.
- [100] Martin Zinkevich. Rules of machine learning: Best practices for ML engineering. URL: <https://developers.google.com/machine-learning/guides/rules-of-ml>, 2017.



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Motivation and Drivers for MLOps

Pravin Y Pawar

Extracted from “mlops.org”
and
“Introducing MLOps” by Dataiku Team

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
 - i.e. classification, sentiment, recommendation, or clusters
 - The performance of each model is evaluated by using evaluation metrics,
 - such as precision & recall, or accuracy

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
	LEARNING (types of ML/AI)	PLANNING • determining the best sequence of steps for a goal	OPTIMISATION • identifying the most optimal parameters.	RECOMMENDATION • predicting user's preferences	REINFORCEMENT LEARNING • learning by experimenting • maximizing reward
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

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 the ML model deployment

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

Deployment Gap(2)

Responses from 582 survey respondents.

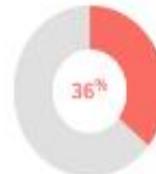
What percentage of your data scientists' time is spent deploying ML models?



36% of survey participants said their data scientists spend **a quarter** of their time deploying ML models



20% of survey participants said their data scientists spend **half to three-quarters** of their time deploying ML models



36% of survey participants said their data scientists spend **a quarter to half** of their time deploying ML models



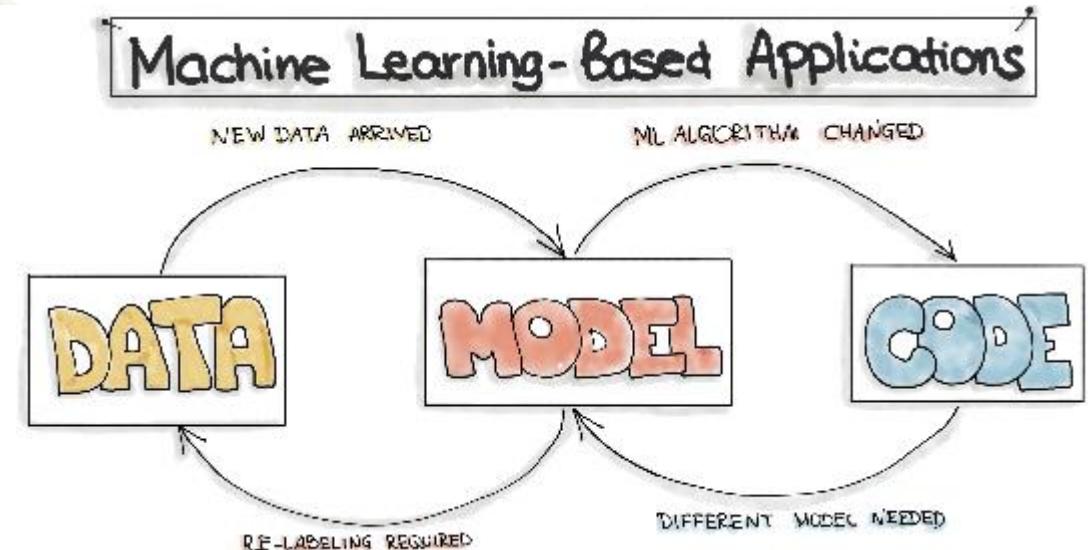
7% of survey participants said their data scientists spend **more than three-quarters** of their time deploying ML models

1% of respondents said they were unsure.

Figure Source

Scenarios of Change That Need to be Managed

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



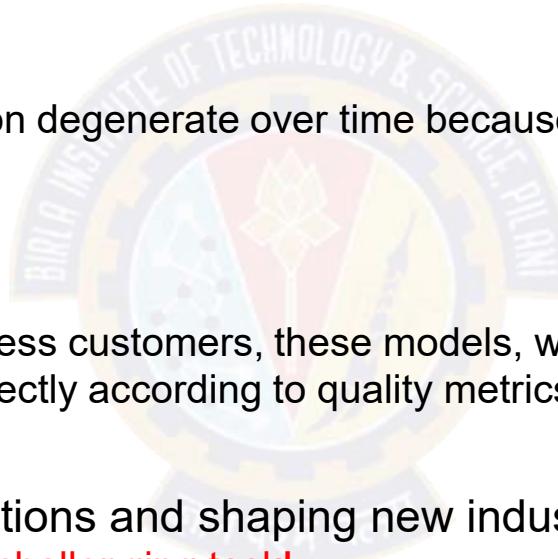
Three levels of change

Scenarios of possible changes in machine learning applications

- After deploying the ML model into a software system,
 - might recognize that as time goes by, the model starts to decay and to behave abnormally,
 - so would need new data to re-train ML model
- After examining the available data,
 - might recognize that it's difficult to get the data needed to solve the problem previously defined
 - so would need to re-formulate the problem
- In the ML project at some stages,
 - might go back in the process and either collect more data, or collect different data and re-label training data
 - which will trigger the re-training of the ML Model
- After serving the model to the end-users,
 - might recognize that the assumptions made for training the model are wrong,
 - so have to change model
- Sometimes the business objective might change while project development
 - need decide to change the machine learning algorithm to train the model

Issues influences the 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 degenerate 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!



MLOps Definition

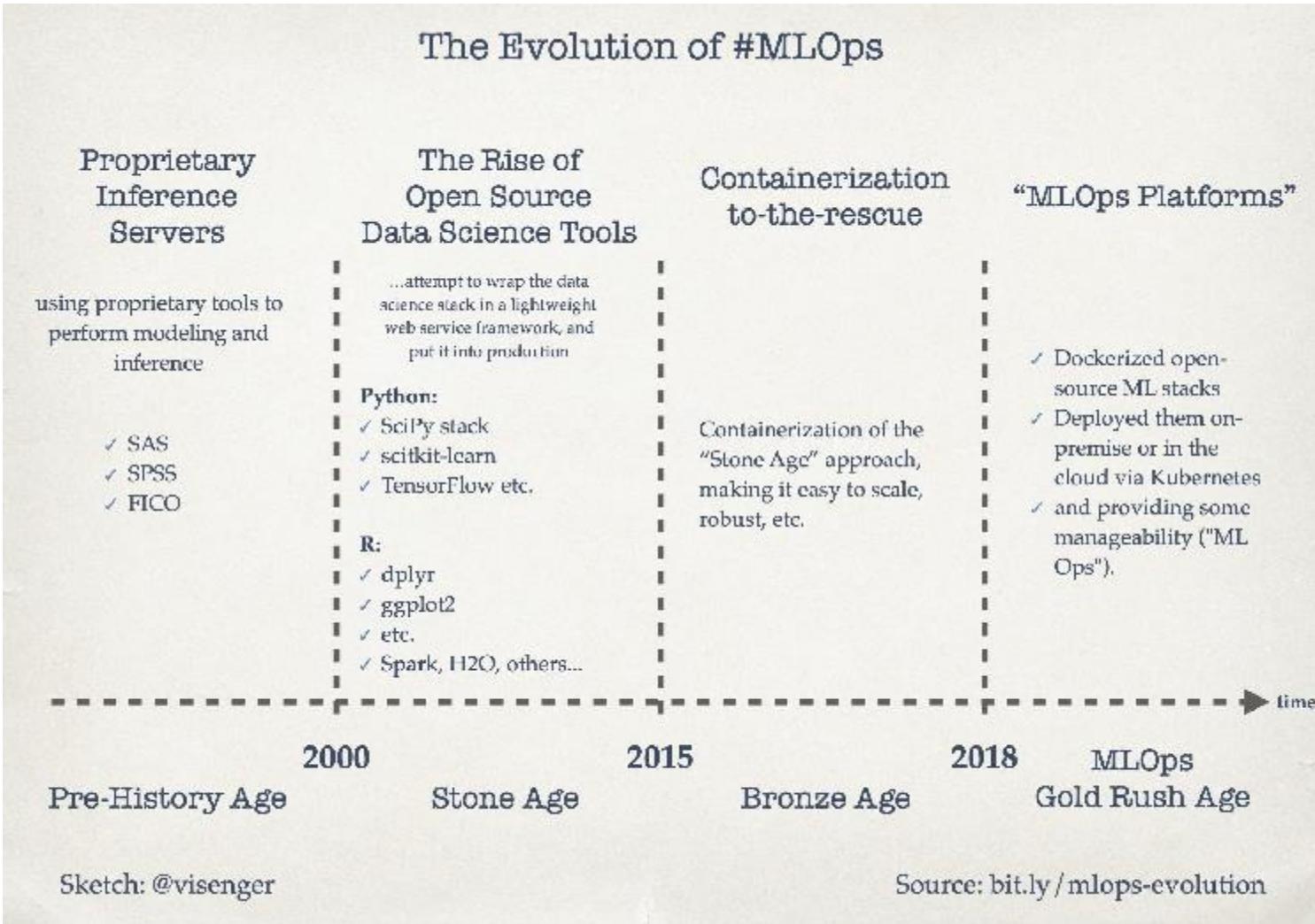
- Per MLOps SIG
 - The term MLOps is defined as “**the extension of the DevOps methodology to include Machine Learning and Data Science assets as first-class citizens within the DevOps ecology**”
- Alternatively, can use the definition of Machine Learning Engineering (MLE),
 - Involves use of scientific 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

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.

The Evolution of the MLOps





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

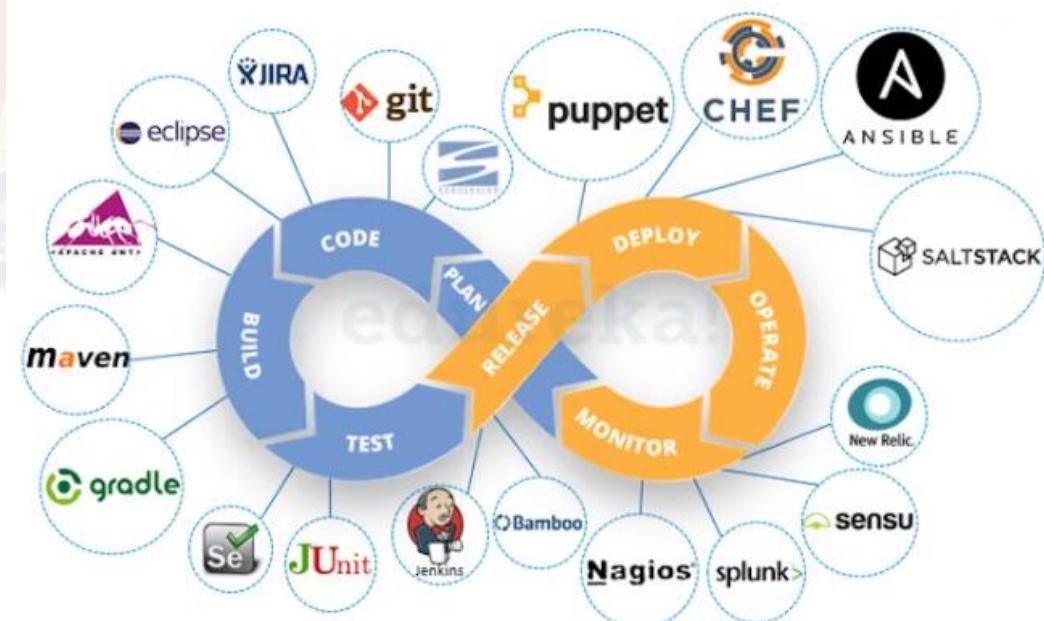
DevOps vs MLOps

Pravin Y Pawar

Extracted from
"Why is DevOps for Machine Learning so Different?"

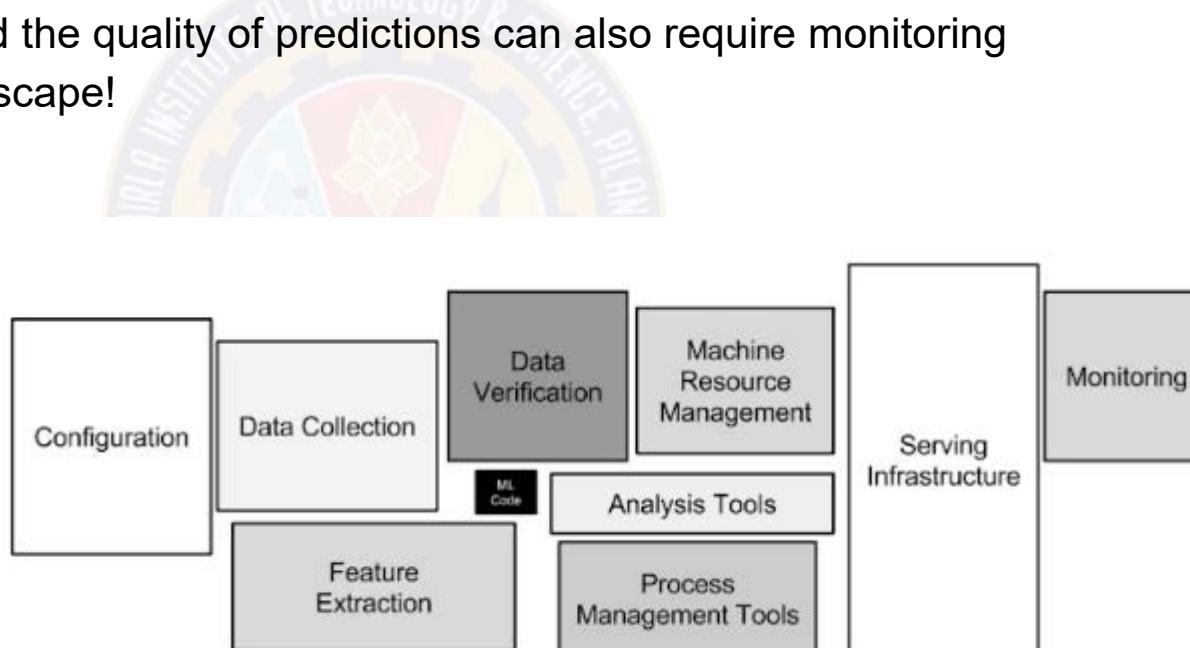
Current State of DevOps vs MLOps

- DevOps is a well-established set of practices to ensure smooth build-deploy-monitor cycles
 - based around CI/CD and infrastructure
 - space of tools includes git, Jenkins, Jira, docker, kubernetes etc.
- MLOps has not achieved the same level of maturity
 - as much as 87% of machine learning projects never go live



ML System is complex

- ML infrastructure is complex!
 - Workflows extend beyond production of artifacts to include data collection, prep and validation
 - Types of hardware resources involved can be specialized (e.g. GPUs) and require management
 - Data flowing through the model and the quality of predictions can also require monitoring
 - resulting in a complex MLOps landscape!



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
 - The simplest programming examples tend to be ‘hello world’ programs that simply codify that a program should output ‘hello world’
 - Further control structures can then be added to add more complex ways to perform actions in response to inputs
 - As more control structures are added, need more of the programming language
- Rule-based input-output pattern is easy to understand in relation to older terminal systems
 - where inputs are all via the keyboard and outputs are almost all text
 - the types of inputs and outputs can be very diverse and complex
- ML does not codify explicitly
 - Rules are indirectly set by capturing patterns from data
 - Makes ML more suitable for a more focused type of problem that can be treated numerically
 - For example, predicting salary from data points/features such as experience, education, location etc.

ML is different from traditional programming

As it needs to take into considerations the facts

- The training data and the code together drive fitting
- The closest thing to an executable is a trained/weighted model
 - These vary by ML toolkit (tensorflow, sc-kit learn, R, h2o, etc.) and model type
- Retraining can be necessary
- Data volumes can be large and training can take a long time
- The data scientist's working process is exploratory and visualizations can be an important part of it
- Leads to different workflows for traditional programming and ML development.

A traditional programming a workflow

- Prepare User Story
- Write code
- Submit Merge Request
- Tests run automatically
- Review and merge
- New version builds
- Built executable deployed to environment
- Further tests
- Promote to next environment
- More tests etc.
- PROD
- Monitor - stack traces or error codes

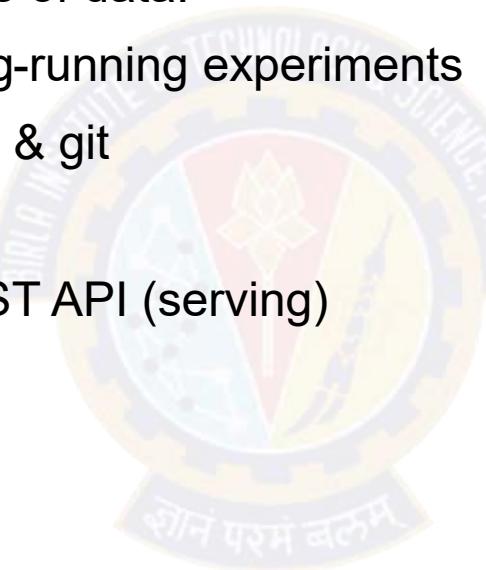


The trigger for a build is a code change in git

The packaging for an executable is normally docker.

A high-level MLOps workflow

- Data inputs and outputs. Preprocessed. Large.
- Data scientist tries stuff locally with a slice of data.
- Data scientist tries with more data as long-running experiments
- Collaboration - often in jupyter notebooks & git
- Model may be pickled/serialized
- Integrate into a running app e.g. add REST API (serving)
- Integration test with app
- Rollout and monitor performance metrics



Driver for a build might be a code change.
Or it might be new data!

The data likely won't be in GIT due to its size

ML is complex!

Training, Testing, serving, and monitoring

- Testing
 - Tests on ML are not likely to be a simple pass/fail since looking for quantifiable performance
 - might choose to express performance numerically with an error level
 - level can vary a lot by business context
- For example, consider a model that predicts a likelihood of a financial transaction being fraudulent
 - little risk in predicting good transactions as fraudulent so long as the customer is not impacted directly
 - but predicting bad transactions as good could be very high risk
- Learning / Training
 - The ML workflow can also differ depending on whether
 - model can learn while it is being used (online learning)
 - or if the training takes place separately from making live predictions (offline learning)
- Monitoring
 - The monitoring for performance metrics part can be particularly challenging and may involve business decisions.
- **The role of MLOps is to support the whole flow of training, serving, rollout and monitoring.**



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Peoples of MLOps

Pravin Y Pawar

Reference “Introducing MLOps”
By Treveil and Dataiku Team

People of MLOps

- Even though machine learning models are primarily built by data scientists
 - it's a misconception that only data scientists can benefit from robust MLOps processes and systems
- MLOps is an essential piece of enterprise AI strategy
 - affects everyone working on, or benefiting from, the machine learning model life cycle
 - covers the roles each of these people plays in the machine learning life cycle,
 - who they should ideally be connected and working together with under a topnotch MLOps program
- Important to note that this field is constantly evolving,
 - bringing with it many new job titles that may not be listed here
 - presenting new challenges (or overlaps) in MLOps responsibilities

Subject matter experts

Role in machine learning model life cycle	MLOps requirements
Provide business questions, goals, or KPIs around which ML models should be framed	Easy way to understand deployed model performance in business terms
Continually evaluate and ensure that model performance aligns with or resolves the initial need	Mechanism or feedback loop for flagging model results that don't align with business expectations

Data scientists

Role in machine learning model life cycle	MLOps requirements
Build models that address the business question or needs brought by subject matter experts	Automated model packaging and delivery for quick and easy (yet safe) deployment to production
Deliver operationalizable models so that they can be properly used in the production environment and with production data	Ability to develop tests to determine the quality of deployed models and to make continual improvements
Assess model quality (of both original and tests) in tandem with subject matter experts to ensure they answer initial business questions or needs	Visibility into the performance of all deployed models (including side-by-side for tests) from one central location
	Ability to investigate data pipelines of each model to make quick assessments and adjustments regardless of who originally built the model

Data engineers

Role in machine learning model life cycle	MLOps requirements
Optimize the retrieval and use of data to power ML models	Visibility into performance of all deployed models
	Ability to see the full details of individual data pipelines to address underlying data plumbing issues



Software engineers

Role in machine learning model life cycle	MLOps requirements
Integrate ML models in the company's applications and systems	Versioning and automatic tests
Ensure that ML models work seamlessly with other non-machine-learning-based applications	The ability to work in parallel on the same application



DevOps engineers

Role in machine learning model life cycle	MLOps requirements
Conduct and build operational systems and test for security, performance, availability	Seamless integration of MLOps into the larger DevOps strategy of the enterprise
Continuous Integration/Continuous Delivery (CI/CD) pipeline management	Seamless deployment pipeline



Model risk managers/ auditors

Role in machine learning model life cycle	MLOps requirements
Minimize overall risk to the company as a result of ML models in production	Robust, likely automated, reporting tools on all models (currently or ever in production), including data lineage
Ensure compliance with internal and external requirements before pushing ML models to production	



Machine learning architects

Role in machine learning model life cycle	MLOps requirements
Ensure a scalable and flexible environment for ML model pipelines, from design to development and monitoring	High-level overview of models and their resources consumed
Introduce new technologies when appropriate that improve ML model performance in production	Ability to drill down into data pipelines to assess and adjust infrastructure needs



ML Engineer

Google's definition

- A Professional Machine Learning Engineer
 - designs, builds, and productionizes ML models to solve business challenges
 - using Google Cloud technologies and knowledge of proven ML models and techniques
- Considers responsible AI throughout the ML development process
- Collaborates closely with other job roles to ensure long-term success of models
- Designs and creates scalable solutions for optimal performance
 - through an understanding of training, retraining, deploying, scheduling, monitoring, and improving models
- Should be proficient in all aspects of model architecture, data pipeline interaction, and metrics interpretation
- Needs familiarity with foundational concepts of application development, infrastructure management, data engineering, and data governance

[Google Certification](#)

MLOps Engineer

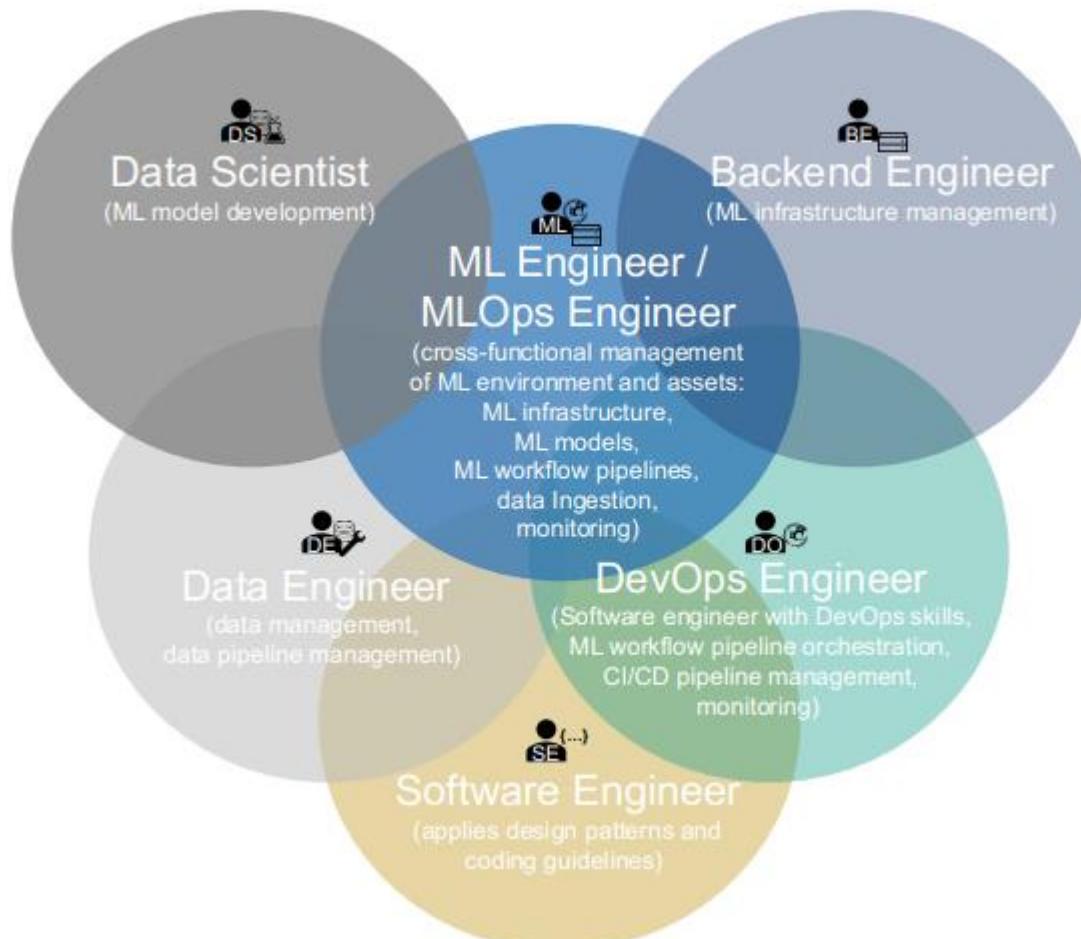
More focused on post-production

- Key responsibilities:
 - Design and implement cloud solutions, build MLOps on cloud (AWS, Azure, or GCP)
 - Build CI/CD pipelines orchestration by GitLab CI, GitHub Actions, Circle CI, Airflow or similar tools
 - Data science model review, run the code refactoring and optimization, containerization, deployment, versioning, and monitoring of its quality
 - Data science models testing, validation and tests automation
 - Communicate with a team of data scientists, data engineers and architect, document the processes
- Required Qualifications:
 - Ability to design and implement cloud solutions and ability to build MLOps pipelines on cloud solutions (AWS, MS Azure or GCP)
 - Experience with MLOps Frameworks like Kubeflow, MLFlow, DataRobot, Airflow etc., experience with Docker and Kubernetes, OpenShift
 - Programming languages like Python, Go, Ruby or Bash, good understanding of Linux, knowledge of frameworks such as scikit-learn, Keras, PyTorch, Tensorflow, etc.
 - Ability to understand tools used by data scientist and experience with software development and test automation

[Accenture Job Post](#)

In summary

Roles and their intersections contributing to the MLOps paradigm



[source](#)



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

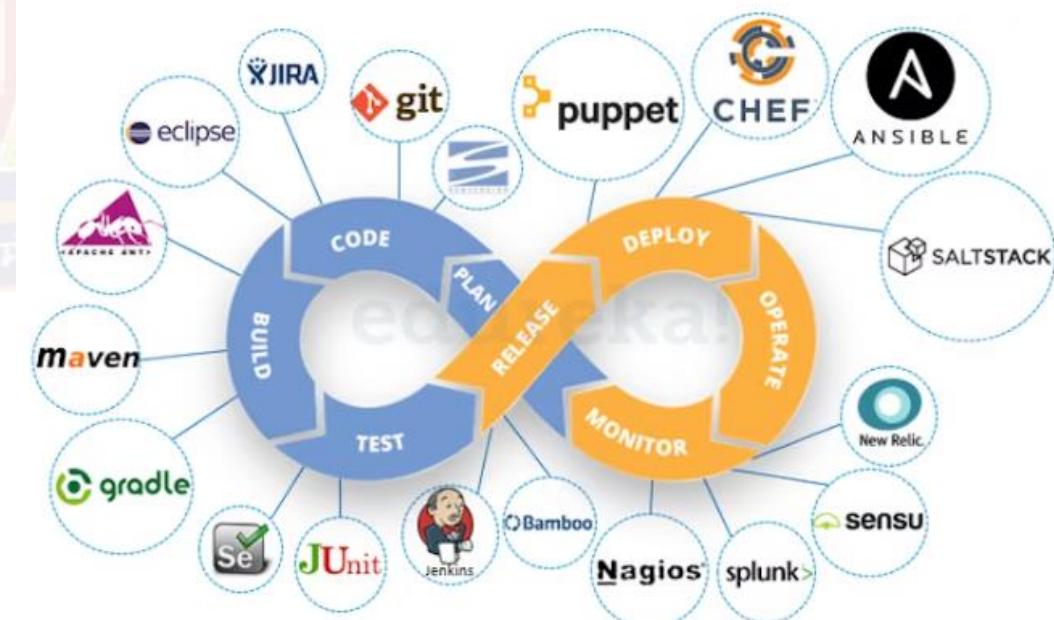
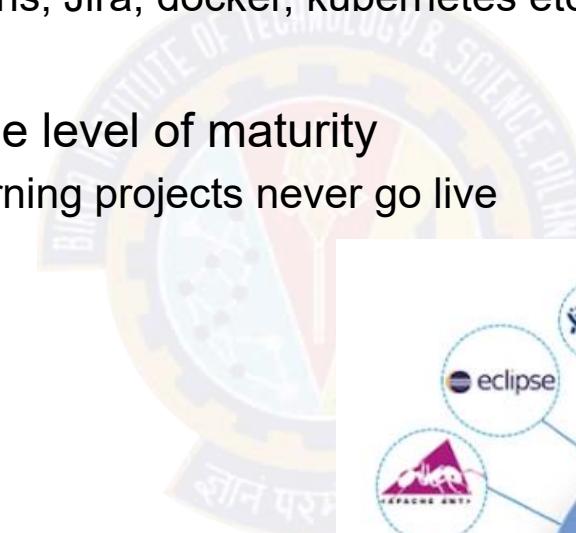
DevOps vs MLOps

Pravin Y Pawar

Extracted from
"Why is DevOps for Machine Learning so Different?"

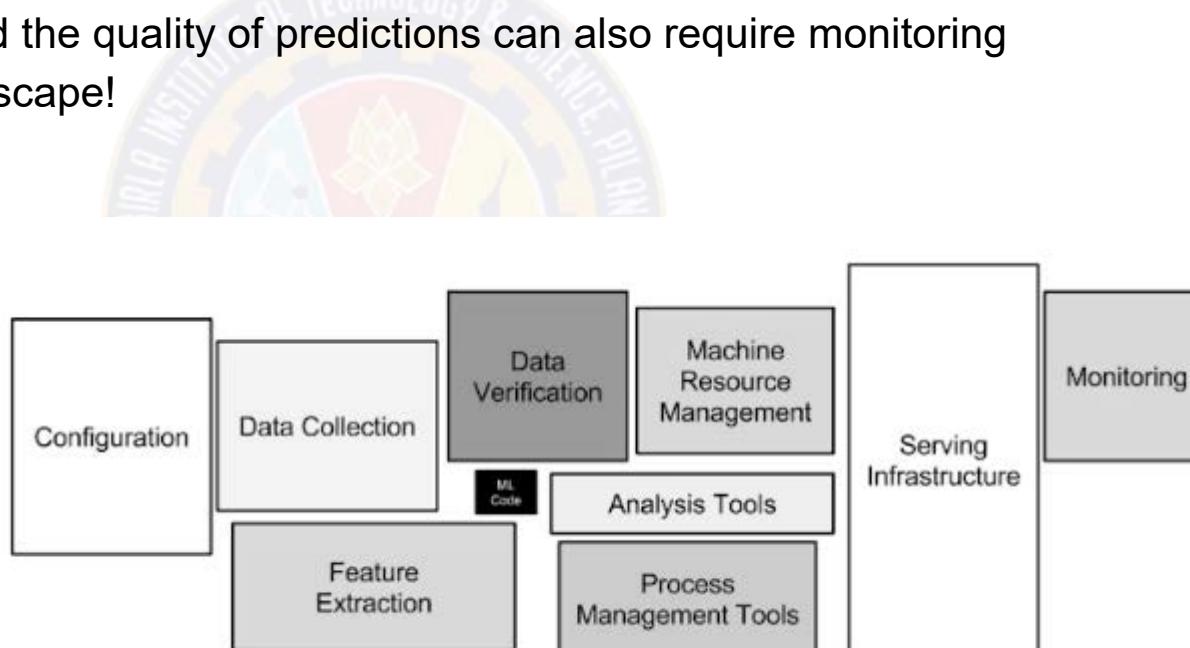
Current State of DevOps vs MLOps

- DevOps is a well-established set of practices to ensure smooth build-deploy-monitor cycles
 - based around CI/CD and infrastructure
 - space of tools includes git, Jenkins, Jira, docker, kubernetes etc.
- MLOps has not achieved the same level of maturity
 - as much as 87% of machine learning projects never go live



ML System is complex

- ML infrastructure is complex!
 - Workflows extend beyond production of artifacts to include data collection, prep and validation
 - Types of hardware resources involved can be specialized (e.g. GPUs) and require management
 - Data flowing through the model and the quality of predictions can also require monitoring
 - resulting in a complex MLOps landscape!



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
 - The simplest programming examples tend to be ‘hello world’ programs that simply codify that a program should output ‘hello world’
 - Further control structures can then be added to add more complex ways to perform actions in response to inputs
 - As more control structures are added, need more of the programming language
- Rule-based input-output pattern is easy to understand in relation to older terminal systems
 - where inputs are all via the keyboard and outputs are almost all text
 - the types of inputs and outputs can be very diverse and complex
- ML does not codify explicitly
 - Rules are indirectly set by capturing patterns from data
 - Makes ML more suitable for a more focused type of problem that can be treated numerically
 - For example, predicting salary from data points/features such as experience, education, location etc.

ML is different from traditional programming

As it needs to take into considerations the facts

- The training data and the code together drive fitting
 - The closest thing to an executable is a trained/weighted model
 - These vary by ML toolkit (tensorflow, sc-kit learn, R, h2o, etc.) and model type
 - Retraining can be necessary
 - Data volumes can be large and training can take a long time
 - The data scientist's working process is exploratory and visualizations can be an important part of it
-
- Leads to different workflows for traditional programming and ML development.

A traditional programming a workflow

- Prepare User Story
- Write code
- Submit Merge Request
- Tests run automatically
- Review and merge
- New version builds
- Built executable deployed to environment
- Further tests
- Promote to next environment
- More tests etc.
- PROD
- Monitor - stack traces or error codes



The trigger for a build is a code change in git

The packaging for an executable is normally docker.

A high-level MLOps workflow

- Data inputs and outputs. Preprocessed. Large.
- Data scientist tries stuff locally with a slice of data.
- Data scientist tries with more data as long-running experiments
- Collaboration - often in jupyter notebooks & git
- Model may be pickled/serialized
- Integrate into a running app e.g. add REST API (serving)
- Integration test with app
- Rollout and monitor performance metrics



Driver for a build might be a code change.
Or it might be new data!

The data likely won't be in GIT due to its size

ML is complex!

Training, Testing, serving, and monitoring

- Testing
 - Tests on ML are not likely to be a simple pass/fail since looking for quantifiable performance
 - might choose to express performance numerically with an error level
 - level can vary a lot by business context
- For example, consider a model that predicts a likelihood of a financial transaction being fraudulent
 - little risk in predicting good transactions as fraudulent so long as the customer is not impacted directly
 - but predicting bad transactions as good could be very high risk
- Learning / Training
 - The ML workflow can also differ depending on whether
 - model can learn while it is being used (online learning)
 - or if the training takes place separately from making live predictions (offline learning)
- Monitoring
 - The monitoring for performance metrics part can be particularly challenging and may involve business decisions.
- **The role of MLOps is to support the whole flow of training, serving, rollout and monitoring.**



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Peoples of MLOps

Pravin Y Pawar

Reference “Introducing MLOps”
By Treveil and Dataiku Team

People of MLOps

- Even though machine learning models are primarily built by data scientists
 - it's a misconception that only data scientists can benefit from robust MLOps processes and systems
- MLOps is an essential piece of enterprise AI strategy
 - affects everyone working on, or benefiting from, the machine learning model life cycle
 - covers the roles each of these people plays in the machine learning life cycle,
 - who they should ideally be connected and working together with under a topnotch MLOps program
- Important to note that this field is constantly evolving,
 - bringing with it many new job titles that may not be listed here
 - presenting new challenges (or overlaps) in MLOps responsibilities

Subject matter experts

Role in machine learning model life cycle	MLOps requirements
Provide business questions, goals, or KPIs around which ML models should be framed	Easy way to understand deployed model performance in business terms
Continually evaluate and ensure that model performance aligns with or resolves the initial need	Mechanism or feedback loop for flagging model results that don't align with business expectations

Data scientists

Role in machine learning model life cycle	MLOps requirements
Build models that address the business question or needs brought by subject matter experts	Automated model packaging and delivery for quick and easy (yet safe) deployment to production
Deliver operationalizable models so that they can be properly used in the production environment and with production data	Ability to develop tests to determine the quality of deployed models and to make continual improvements
Assess model quality (of both original and tests) in tandem with subject matter experts to ensure they answer initial business questions or needs	Visibility into the performance of all deployed models (including side-by-side for tests) from one central location
	Ability to investigate data pipelines of each model to make quick assessments and adjustments regardless of who originally built the model

Data engineers

Role in machine learning model life cycle	MLOps requirements
Optimize the retrieval and use of data to power ML models	Visibility into performance of all deployed models
	Ability to see the full details of individual data pipelines to address underlying data plumbing issues



Software engineers

Role in machine learning model life cycle	MLOps requirements
Integrate ML models in the company's applications and systems	Versioning and automatic tests
Ensure that ML models work seamlessly with other non-machine-learning-based applications	The ability to work in parallel on the same application



DevOps engineers

Role in machine learning model life cycle	MLOps requirements
Conduct and build operational systems and test for security, performance, availability	Seamless integration of MLOps into the larger DevOps strategy of the enterprise
Continuous Integration/Continuous Delivery (CI/CD) pipeline management	Seamless deployment pipeline



Model risk managers/ auditors

Role in machine learning model life cycle	MLOps requirements
Minimize overall risk to the company as a result of ML models in production	Robust, likely automated, reporting tools on all models (currently or ever in production), including data lineage
Ensure compliance with internal and external requirements before pushing ML models to production	



Machine learning architects

Role in machine learning model life cycle	MLOps requirements
Ensure a scalable and flexible environment for ML model pipelines, from design to development and monitoring	High-level overview of models and their resources consumed
Introduce new technologies when appropriate that improve ML model performance in production	Ability to drill down into data pipelines to assess and adjust infrastructure needs



ML Engineer

Google's definition

- A Professional Machine Learning Engineer
 - designs, builds, and productionizes ML models to solve business challenges
 - using Google Cloud technologies and knowledge of proven ML models and techniques
- Considers responsible AI throughout the ML development process
- Collaborates closely with other job roles to ensure long-term success of models
- Designs and creates scalable solutions for optimal performance
 - through an understanding of training, retraining, deploying, scheduling, monitoring, and improving models
- Should be proficient in all aspects of model architecture, data pipeline interaction, and metrics interpretation
- Needs familiarity with foundational concepts of application development, infrastructure management, data engineering, and data governance

[Google Certification](#)

MLOps Engineer

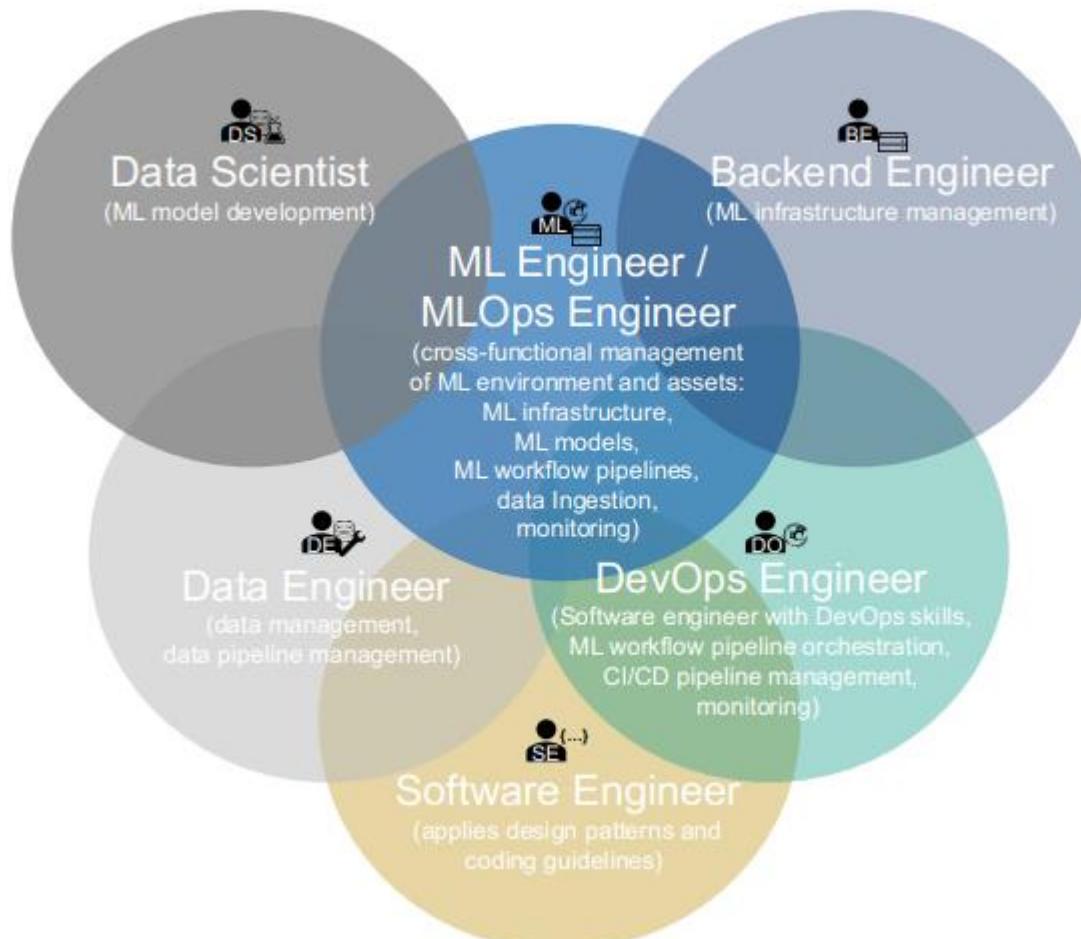
More focused on post-production

- Key responsibilities:
 - Design and implement cloud solutions, build MLOps on cloud (AWS, Azure, or GCP)
 - Build CI/CD pipelines orchestration by GitLab CI, GitHub Actions, Circle CI, Airflow or similar tools
 - Data science model review, run the code refactoring and optimization, containerization, deployment, versioning, and monitoring of its quality
 - Data science models testing, validation and tests automation
 - Communicate with a team of data scientists, data engineers and architect, document the processes
- Required Qualifications:
 - Ability to design and implement cloud solutions and ability to build MLOps pipelines on cloud solutions (AWS, MS Azure or GCP)
 - Experience with MLOps Frameworks like Kubeflow, MLFlow, DataRobot, Airflow etc., experience with Docker and Kubernetes, OpenShift
 - Programming languages like Python, Go, Ruby or Bash, good understanding of Linux, knowledge of frameworks such as scikit-learn, Keras, PyTorch, Tensorflow, etc.
 - Ability to understand tools used by data scientist and experience with software development and test automation

[Accenture Job Post](#)

In summary

Roles and their intersections contributing to the MLOps paradigm



[source](#)



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

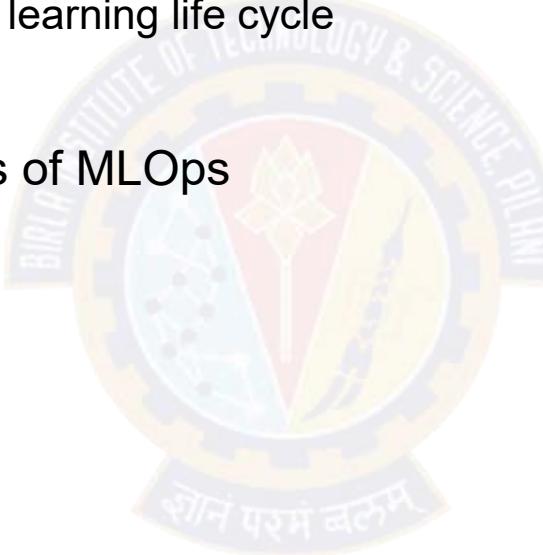
Key MLOps features

Pravin Y Pawar

Adapted from “Introducing MLOps”
by Treveil and Dataiku team

MLOps Component

- 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
 - and governance



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

Model development(2)

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 hyper parameters 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

Model development(3)

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

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,
 - the 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!**



Productionalization and Deployment(2)

Model Deployment Types and Contents

- Good 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
 - Model is packaged into an application, which is then published
 - A common example is an application that provides batch-scoring of requests
- What to-be-deployed models consist of depends, of course, on the technology chosen,
 - but typically they comprise 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
 - aims to improve model portability between systems and simplify deployment
 - 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.

Productionalization and Deployment(3)

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
 - 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

Monitoring

Different concerns

- Once a model is deployed to production, it is crucial that it continue to perform well over time
 - But good performance means different things to different people, in particular to the DevOps team, to data scientists, and to the business
- DevOps Concerns
 - The concerns of the DevOps team are very familiar and include questions like:
 - Is the model getting the job done quickly enough?
 - Is it using a sensible amount of memory and processing time?
 - But overall, the existing expertise in DevOps teams for monitoring and managing resources can be readily applied to ML models
- Data Scientist Concerns
 - The data scientist is interested in monitoring ML models for a new, more challenging reason:
 - they can degrade over time, since ML models are effectively models of the data they were trained on
 - not a problem faced by traditional software, but it is inherent to machine learning
 - At some point, the performance will become unacceptable, and model retraining becomes necessary
- Business Concerns
 - The business has a holistic outlook on monitoring, and some of its concerns might include questions like:
 - Is the model delivering value to the enterprise?
 - Do the benefits of the model outweigh the cost of developing and deploying it? (And how can we measure this?)
 - The KPIs identified for the original business objective are one part of this process
 - Where possible, these should be monitored automatically, but this is rarely trivial

Iteration and Life Cycle

Iteration

- Developing and deploying improved versions of a model is an essential part of the MLOps life cycle
 - one of the more challenging
- Various reasons to develop a new model version,
 - one of which is model performance degradation due to model drift
 - sometimes there is a need to reflect refined business objectives and KPIs
 - other times, it's just that the data scientists have come up with a better way to design the model
- Iteration
 - In some fast-moving business environments, new training data becomes available every day
 - Daily retraining and redeployment of the model are often automated to ensure that the model reflects recent experience as closely as possible.
 - Retraining an existing model with the latest training data is the simplest scenario for iterating a new model version
 - With a new model version built, the next step is to compare the metrics with the current live model version
 - requires evaluating both models on the same development dataset, whether it be the previous or latest version
 - If metrics and checks suggest a wide variation between the models, automated scripts should not be redeployed, and manual intervention should be sought
 - Even in the “simple” automated retraining scenario with new training data,
 - there is a need for multiple development datasets based on scoring data reconciliation (with ground truth when it becomes available), data cleaning and validation, the previous model version, and a set of carefully considered checks
 - Retraining in other scenarios is likely to be even more complicated,
 - rendering automated redeployment unlikely

Iteration and Life Cycle(2)

The Feedback Loop

- In large enterprises, DevOps best practices typically dictate that
 - the live model scoring environment and the model retraining environment are distinct
 - as a result, the evaluation of a new model version on the retraining environment is likely to be compromised
- One approach to mitigating this uncertainty is shadow testing,
 - where the new model version is deployed into the live environment alongside the existing model
 - All live scoring is handled by the incumbent model version,
 - but each new request is then scored again by the new model version and the results logged, but not returned to the requestor
 - Once sufficient requests have been scored by both versions, the results can be compared statistically
 - Shadow scoring also gives more visibility to the SMEs on the future versions of the model and may thus allow for a smoother transition
- In A/B testing, both models are deployed into the live environment,
 - but input requests are split between the two models
 - Each request is processed by one or the other model, not both
 - Results from the two models are logged for analysis (but never for the same request)
 - Drawing statistically meaningful conclusions from an A/B test requires careful planning of the test

Governance

- Governance is the set of controls placed on a business to ensure that it delivers on its responsibilities
 - to all stakeholders, from shareholders and employees to the public and national governments
 - include financial, legal, and ethical obligations
 - Underpinning all three of these is the fundamental principle of fairness
- Legal obligations are the easiest to understand!
 - Businesses were constrained by regulations long before the advent of machine learning
 - Many regulations target specific industries - finance, pharma etc
- Recently, governments across the world have imposed regulations
 - to protect the public from the impact of the use of personal data by businesses
 - The 2016 EU General Data Protection Regulation (GDPR)
 - The 2018 California Consumer Privacy Act(CCPA)
- Governments are now starting to turn their regulatory eye to ML specifically,
 - hoping to mitigate the negative impact of its use
 - what is clear is that businesses will have to take heed of yet more regulation when applying ML

Governance(2)

Data and Process governance

- With increasing public activism on the subject, businesses are engaging with ideas of Responsible AI,
 - the ethical, transparent, and accountable application of AI technology
- Applying good governance to MLOPs is challenging!
 - The processes are complex, the technology is opaque, and the dependence on data is fundamental
- Governance initiatives in MLOps broadly fall into one of two categories:
- Data governance
 - A framework for ensuring appropriate use and management of data
- Process governance
 - The use of well-defined processes to ensure all governance considerations have been addressed at the correct point in the life cycle of the model and that a full and accurate record has been kept



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

MLOps: Continuous delivery and automation pipelines in machine learning

Pravin Y Pawar

Adapted from
MLOps: Continuous delivery and automation
pipelines in machine learning
Cloud Architecture Center | Google Cloud

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**

Real Challenge

Continuously deploy and operate

- Data scientists can implement and train an ML model with predictive performance on an offline holdout dataset
 - 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 that **advocating for automation and monitoring at all steps of ML system construction**, including
 - integration,
 - testing,
 - releasing,
 - deployment and infrastructure management

Elements of ML System

Vast and Complex

- 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
- 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,
 - need to **apply DevOps principles to ML systems (MLOps)**

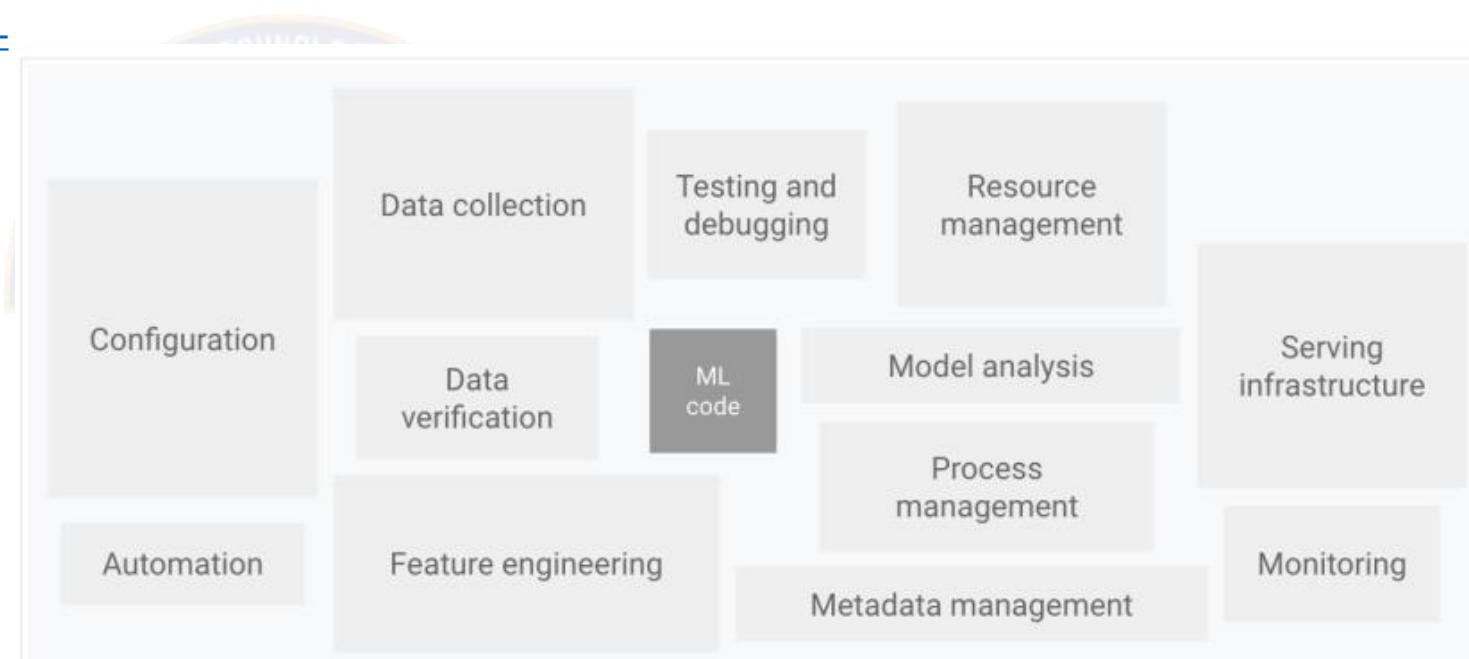


Figure 1. Elements for ML systems. Adapted from [Hidden Technical Debt in Machine Learning Systems](#) (<https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf>).

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 \(CI\)](#)
 - [Continuous Delivery \(CD\)](#)
- Continuous Integration (CI)
 - practice of merging all developers' working copies to a shared mainline several times a day
 - the practice of frequently integrating one's new or changed code with the existing code repository
 - should occur frequently enough that no intervening window remains between commit and build
 - such that no errors can arise without developers noticing them and correcting them immediately
- Continuous Delivery (CD)
 - a software engineering approach in which teams produce software in short cycles,
 - ensuring that the software can be reliably released at any time and, following a pipeline through a "production-like environment", without doing so manually
 - aims at building, testing, and releasing software with greater speed and frequency
 - helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production
 - A straightforward and repeatable deployment process is important for continuous delivery

DevOps versus MLOps(2)

ML systems differ from other software systems - Phases

- Team skills
 - In an ML project, the team usually includes **data scientists or ML researchers**,
 - who focus on exploratory data analysis, model development, and experimentation
 - might **not be experienced** software engineers who can **build production-class services**
- Development
 - ML is **experimental** in nature - need to try different features, algorithms, modeling techniques, and parameter configurations
 - Challenge is **tracking what worked and what didn't**, and maintaining **reproducibility** while maximizing code **reusability**
- Testing
 - More **involved** than testing other software systems
 - In addition to typical unit and integration tests, need **data validation, trained model quality evaluation, and model validation**
- Deployment
 - In ML systems, deployment isn't as simple as deploying an offline-trained ML model as a prediction service
 - **Required to deploy a multi-step pipeline to automatically retrain and deploy model**
 - pipeline adds **complexity** and requires **to automate steps** that are manually done before deployment by data scientists to train and validate new models
- Production
 - ML models can have **reduced performance** not only due to **suboptimal coding**, but also due to constantly **evolving data profiles**
 - Models can **decay** in more ways than conventional software systems - need to consider this degradation
 - **Need to track summary statistics of data and monitor the online performance of model**
 - to send notifications or roll back when values deviate from expectations

DevOps versus MLOps(3)

ML systems differ from other software systems - Process

- An **ML system is a software system**
 - similar practices apply to help guarantee reliably to build and operate ML systems at scale
- ML and other software systems are similar wrt
 - **continuous integration** of source control, unit testing, integration testing
 - **continuous delivery** of the software module or the package
- However, in ML, there are a few **notable differences**:
 - CI is no longer only about **testing and validating code and components**
 - but also **testing and validating data, data schemas, and models**
 - CD is no longer about a **single software package or a service**,
 - but a **system (an ML training pipeline)** that should **automatically deploy another service (model prediction service)**
 - CT is a new property, **unique to ML systems**
 - concerned with **automatically retraining and serving the models**

Data science steps for ML

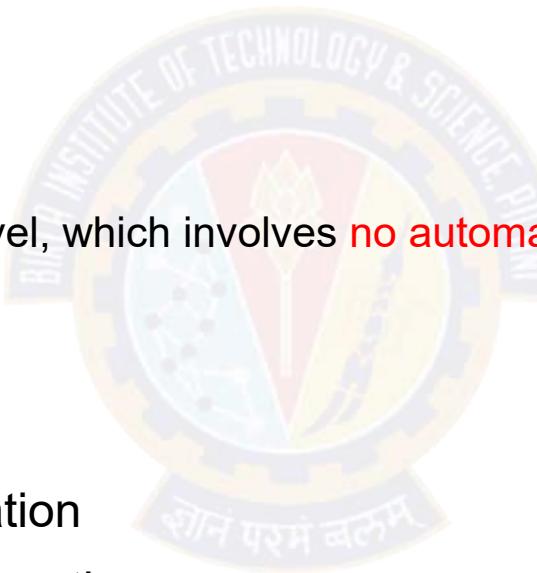
Steps can be completed manually or can be completed by an automatic pipeline

- Data extraction
 - select and integrate the relevant data from various data sources for the ML task.
- Data analysis
 - perform exploratory data analysis
- Data preparation
 - prepare date for the ML task - involves data cleaning, data transformations splitting and feature engineering
- Model training
 - implement different algorithms with the prepared data to train various ML models
- Model evaluation
 - evaluate on a holdout test set to assess the model quality
- Model validation
 - confirm whether model is adequate for deployment — predictive performance is better than a certain baseline
- Model serving
 - deploy to a target environment to serve predictions
- Model monitoring
 - monitor models predictive performance to potentially invoke a new iteration in the ML process

Maturity of the ML process

MLOps levels

- The **level of automation of steps** defines the **maturity** of the ML process
 - reflects the velocity of training new models given new data or training new models given new implementations
- Three levels of MLOps
 - starting from the most common level, which involves **no automation**, up to automating both ML and **CI/CD pipelines**
- MLOps level 0: Manual process
- MLOps level 1: ML pipeline automation
- MLOps level 2: CI/CD pipeline automation



MLOps level 0: Manual process

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

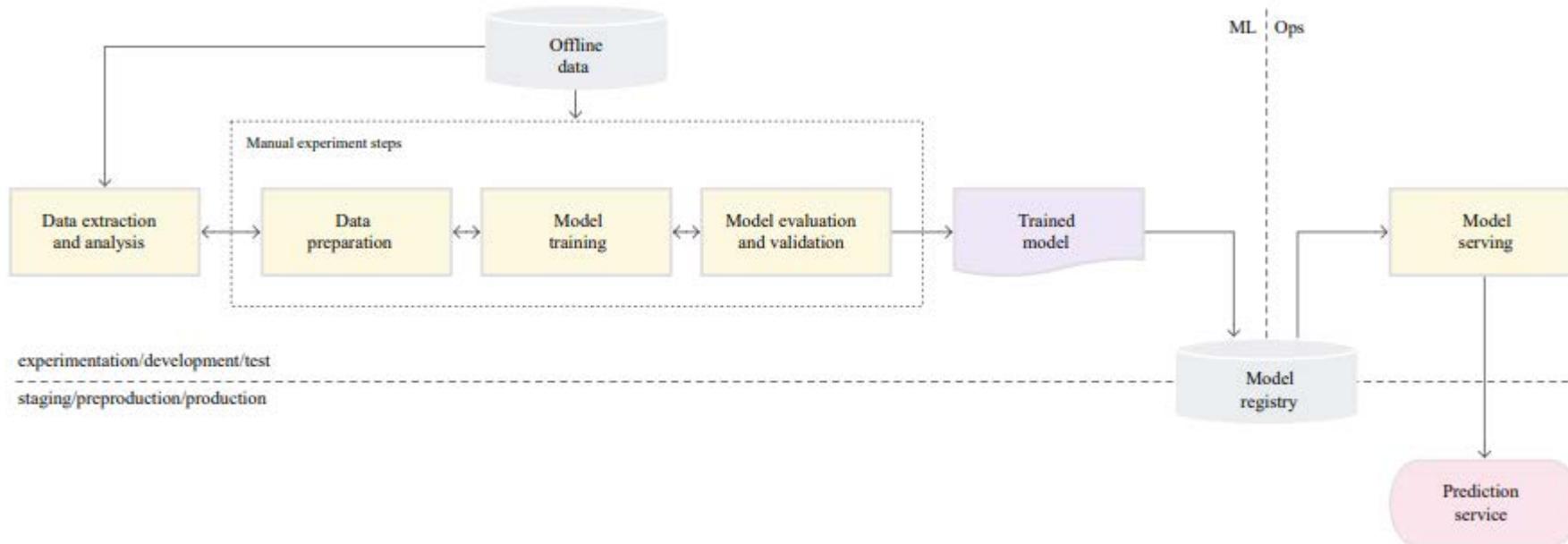


Figure 2. Manual ML steps to serve the model as a prediction service.

MLOps level 0: Manual process(2)

Characteristics

- Manual, script-driven, and interactive process
 - Every step is **manual**, including data analysis, data preparation, model training, and validation
 - Requires **manual execution** of each step, and **manual transition** from one step to another
 - usually driven by **experimental code** that is written and executed in **notebooks** by data scientist interactively, until a workable model is produced
- Disconnection between ML and operations
 - Separates **data scientists who create the model** and **engineers who serve the model as a prediction service**
 - Data scientists hand over a trained model as an artifact to the engineering team to deploy on their API infrastructure
 - can include putting the trained model in a storage location, checking the model object into a code repository, or uploading it to a models registry
 - Engineers who deploy the model need to make the required features available in production for low-latency serving
 - **can lead to training-serving skew**
- Infrequent release iterations
 - Assumes that data science team manages **a few models** that **don't change frequently**

MLOps level 0: Manual process(3)

Characteristics(2)

- No CI
 - Because **few implementation changes** are assumed, CI is **ignored**
 - Usually, testing the code is part of the notebooks or script execution
 - Scripts and notebooks that implement the experiment steps are source controlled
 - produce artifacts such as **trained models**, **evaluation metrics**, and **visualizations**
- No CD
 - Because there **aren't frequent model version deployments**, CD isn't considered
- Deployment refers to the prediction service
 - Concerned only with **deploying the trained model as a prediction service**
 - (for example, a microservice with a REST API)
- Lack of active performance monitoring
 - Doesn't track or log the **model predictions and actions**, which are required in order to **detect model performance degradation and other model behavioral drifts**

MLOps level 0: Manual process(4)

Challenges

- Common in many **businesses** that are **beginning** to apply ML to their use cases
 - manual, data-scientist driven process might be sufficient when **models are rarely changed or trained**
 - in practice, models **often break** when they are **deployed** in the real world
- To address **these challenges** and to maintain model's accuracy in production, **requires** to:
- Actively monitor the **quality of model** in production
 - to detect performance degradation and model staleness
 - acts as a **cue to a new experimentation iteration** and (manual) **retraining** of the model on new data
- Frequently **retrain** production models
 - need to retrain model with the **most recent data** - to **capture the evolving and emerging patterns**
- Continuously **experiment with new implementations** to produce the model
 - To harness the latest ideas and advances in **technology**
 - To try out new implementations such as **feature engineering, model architecture, and hyper parameters**
- To address the challenges of this manual process, **MLOps practices for CI/CD and CT** are helpful!

MLOps level 1: ML pipeline automation

Aim to perform continuous training of the model by automating the ML pipeline

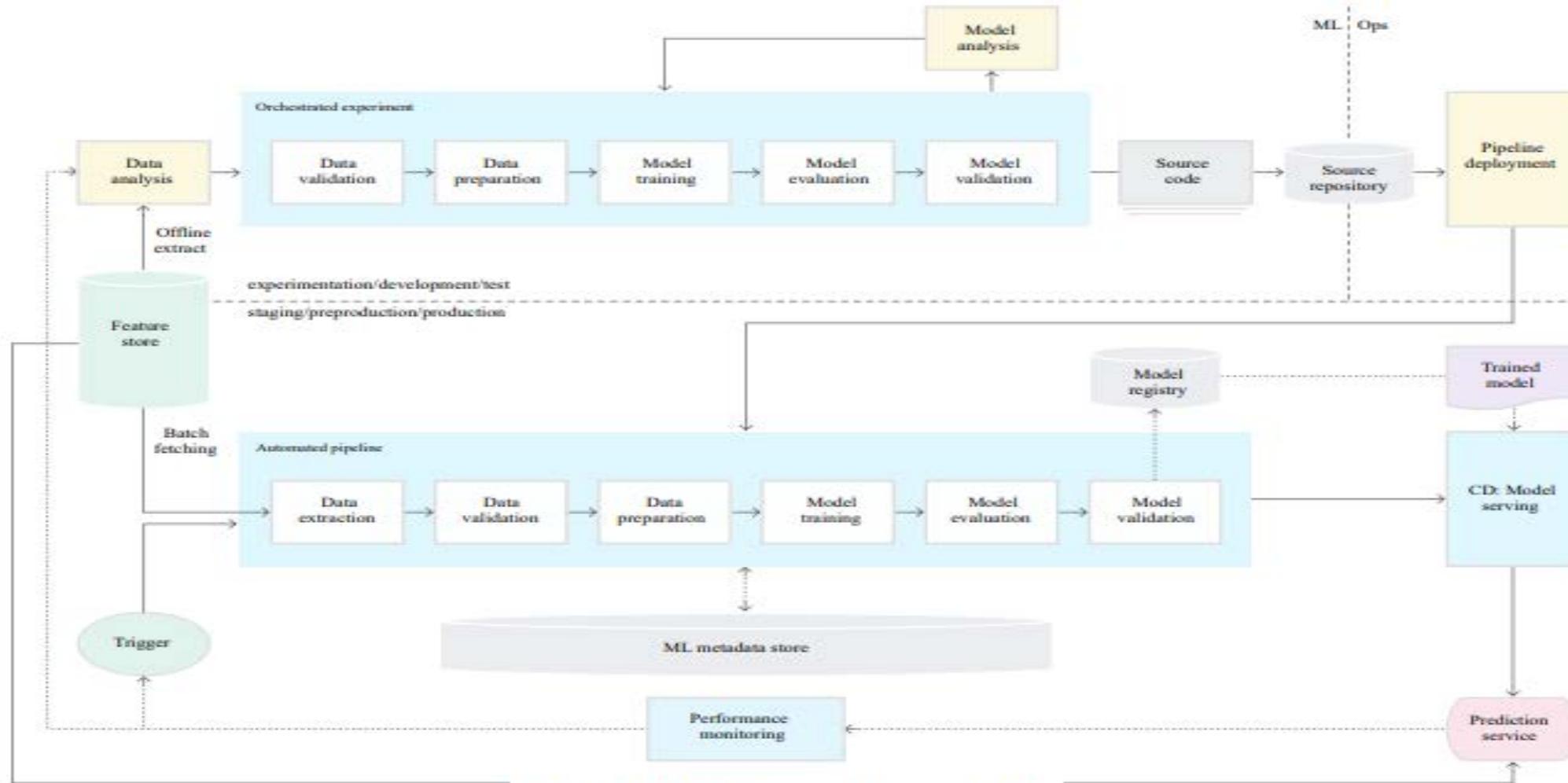


Figure 3. ML pipeline automation for CT.

MLOps level 1: ML pipeline automation(2)

Characteristics

- Rapid experiment
 - Steps of the ML experiment are **orchestrated**
 - Transition between steps is **automated**, which leads to rapid iteration of experiments and better readiness to move the whole pipeline to production
- CT of the model in production
 - Model is automatically trained in production using fresh data based on live pipeline **triggers**
- Experimental-operational symmetry
 - The pipeline implementation that is used in the **development or experiment environment** is used in the **preproduction and production environment**
 - key aspect of MLOps practice for unifying DevOps!
- Continuous delivery of models
 - ML pipeline in **production** continuously **delivers prediction services** to new models that are trained on **new data**
 - The model deployment step, which serves the trained and validated model as a prediction service for online predictions, is automated

MLOps level 1: ML pipeline automation(3)

Characteristics(2)

- Modularized code for components and pipelines
 - To construct ML pipelines, **components need to be reusable, composable, and potentially shareable across ML pipelines**
 - Source code for components must be **modularized**
 - In addition, **components** should ideally be **containerized** to do the following:
 - Decouple the execution environment from the custom code runtime
 - Make code reproducible between development and production environments
 - Isolate each component in the pipeline
- Pipeline deployment
 - In **level 0**, a **trained model is deployed** as a prediction service to production
 - In **level 1**, a **whole training pipeline is deployed**
 - which **automatically and recurrently runs** to serve the trained model as the prediction service

MLOps level 1: ML pipeline automation(4)

Data validation

- When ML pipeline is deployed to production, one or more of the triggers executes the pipeline
- Pipeline expects new, live data to produce a new model version that is trained on the new data
- Automated data validation and model validation steps are required in the production pipeline
- Data validation
 - Required before model training to decide whether to retrain the model or stop the execution of the pipeline
 - automatically make a decision if the following was identified by the pipeline
 - Data schema skews
 - presence of anomalies in the input data, which means that the downstream pipeline steps, including data processing and model training, receives data that doesn't comply with the expected schema
 - should stop the pipeline so the data science team can investigate
 - include receiving unexpected features, not receiving all the expected features, or receiving features with unexpected values
 - Data values skews
 - significant changes in the statistical properties of data, which means that data patterns are changing
 - need to trigger a retraining of the model to capture these changes

MLOps level 1: ML pipeline automation(5)

Model validation

- Model validation
 - Occurs **after successfully train the model** given the new data to evaluate and validate the model before it's promoted to production
- Consists of
 - **Producing evaluation metric values** using the trained model on a test dataset to assess the model's predictive quality
 - **Comparing the evaluation metric values** produced by newly trained model to the current model,
 - **Making sure that the performance of the model is consistent** on various segments of the data
 - **Making sure to test model for deployment**, including **infrastructure compatibility** and **consistency with the prediction service API**
 - In addition to offline model validation, a newly deployed model undergoes **online model validation**
 - **in a canary deployment or an A/B testing setup** — before it serves prediction for the online traffic

MLOps level 1: ML pipeline automation(6)

Feature store

- An **optional additional component** - centralized repository to **standardize** the definition, storage, and access of **features** for training and serving
- Provide an API
 - for both high-throughput **batch serving** and low-latency **real-time serving** for the feature values
 - to support both **training and serving** workloads
- Helps data scientists to:
 - **Discover and reuse available feature sets** for their entities, **instead of re-creating** the same or similar ones
 - **Avoid having similar features that have different definitions** by maintaining features and their related metadata
 - **Serve up-to-date feature values** from the feature store
 - **Avoid training-serving skew** by using the feature store as the **data source** for experimentation, continuous training, and online serving
 - This approach **makes sure** that the **features used for training are the same ones used during serving**:
 - For **experimentation**, data scientists can get **an offline extract** from the feature store to run their experiments.
 - For **continuous training**, the automated ML training pipeline can **fetch a batch of the up-to-date feature values** of the dataset that are used for the training task.
 - For **online prediction**, the prediction service can **fetch in a batch of the feature values related to the requested entity**, such as customer demographic features, product features, and current session aggregation features

MLOps level 1: ML pipeline automation(7)

Metadata management

- Information about each execution of the ML pipeline is recorded in order to help
 - with **data and artifacts lineage, reproducibility, and comparisons**
 - to debug errors and anomalies
- Each time **pipeline is executed**, the ML metadata store records the following **metadata**:
 - The pipeline and component **versions** that were executed
 - The start and end **date, time**, and how long the pipeline took to complete each of the steps
 - The **executor** of the pipeline
 - The **parameter** arguments that were passed to the pipeline
 - The pointers to the **artifacts** produced by each step of the pipeline
 - location of prepared data, validation anomalies, computed statistics, and extracted vocabulary from the categorical features
 - A pointer to the **previous trained model**
 - if needs to roll back to a previous model version
 - if needs to produce evaluation metrics for a previous model version when the pipeline is given new test data during the model validation step
 - The **model evaluation metrics** produced during the model evaluation step
 - for both the training and the testing sets
 - helps to **compare the performance of a newly trained model to the recorded performance of the previous model**

MLOps level 1: ML pipeline automation(8)

ML pipeline triggers

- Can automate the ML production pipelines to retrain the models with new data, depending on use case:
- On demand
 - Ad-hoc manual execution of the pipeline.
- On a schedule
 - New, labelled data is systematically available for the ML system on a daily, weekly, or monthly basis
 - Retraining frequency also depends on how frequently the data patterns change, and how expensive it is to retrain models
- On availability of new training data
 - New data is available on an ad-hoc basis when new data is collected and made available in the source databases
- On model performance degradation
 - The model is retrained when there is noticeable performance degradation
- On significant changes in the data distributions (concept drift)
 - notice significant changes on the data distributions of the features that are used to perform the prediction
 - changes suggest that model has gone stale, and that needs to be retrained on fresh data

MLOps level 1: ML pipeline automation(9)

Challenges

- Suitable when need **deploy new models based on new data**, rather than based on new ML ideas
 - New implementations of the pipeline **aren't frequently deployed**
 - Managing only a **few pipelines**
 - **Manually test the pipeline** and its components
- When need to try **new ML ideas** and **rapidly deploy new implementations** of the ML components
 - Need to manage **many ML pipelines** in production
 - Need a **CI/CD setup** to automate the build, test, and deployment of ML pipelines



MLOps level 2: CI/CD pipeline automation

Implementation of the ML pipeline using CI/CD

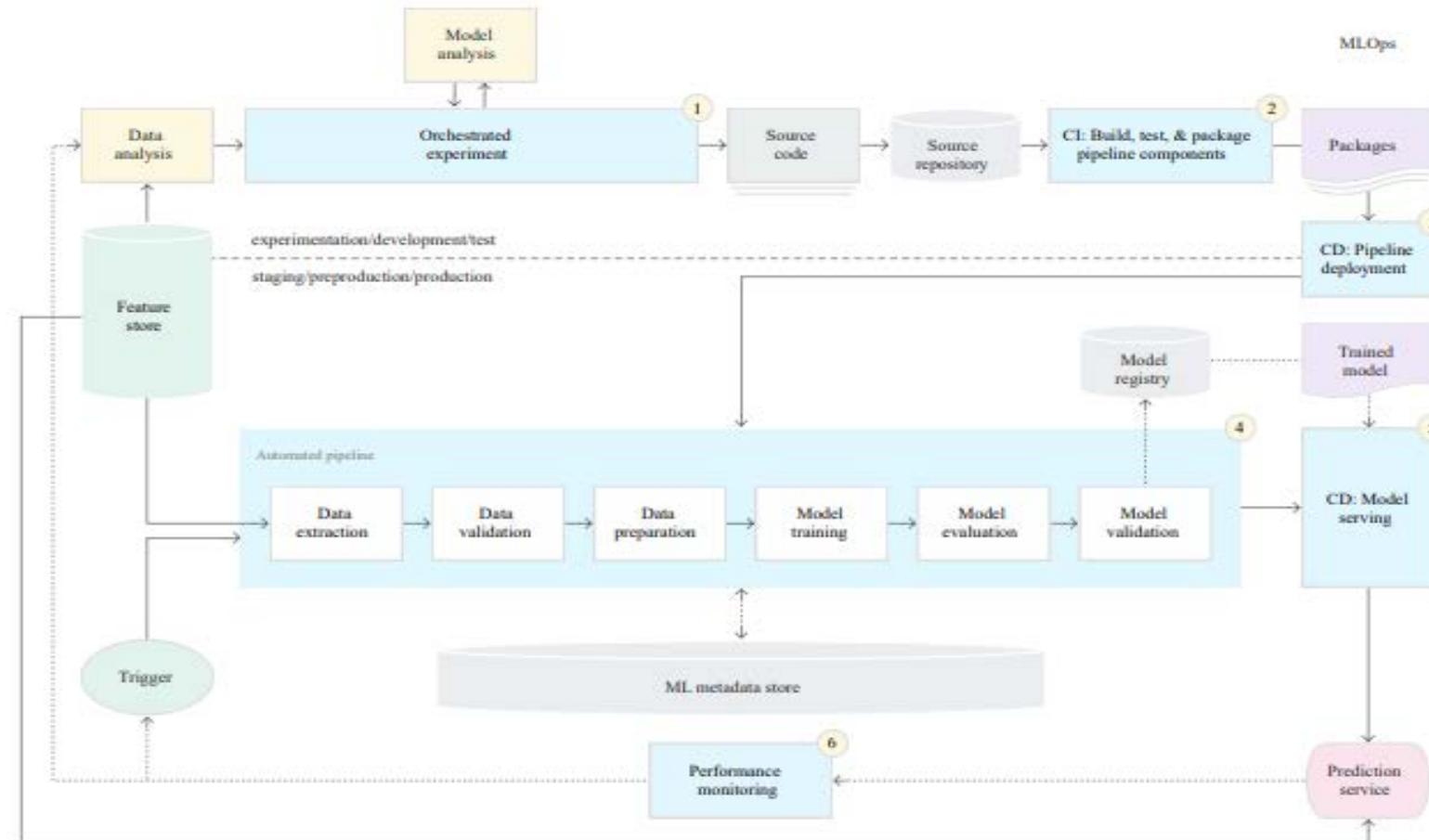


Figure 4. CI/CD and automated ML pipeline.

MLOps level 2: CI/CD pipeline automation(2)

Stages of the ML CI/CD automation pipeline

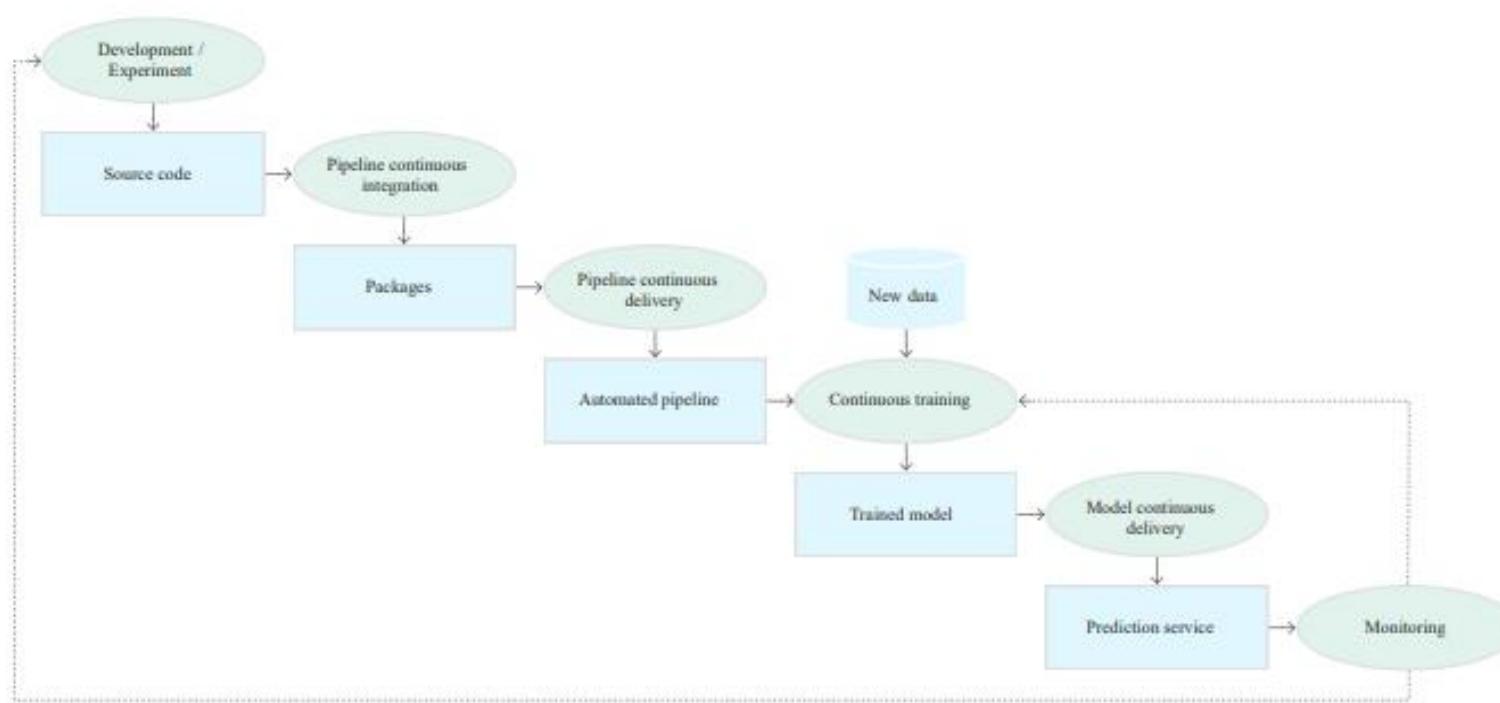


Figure 5. Stages of the CI/CD automated ML pipeline.

MLOps level 2: CI/CD pipeline automation(3)

Stages of the ML CI/CD automation pipeline(2)

- Development and experimentation
 - Iteratively try out new ML algorithms and new modeling where the experiment steps are orchestrated
 - Output - the source code of the ML pipeline steps that are then pushed to a source repository
- Pipeline continuous integration
 - Build source code and run various tests
 - Outputs - pipeline components (packages, executables, and artifacts) to be deployed in a later stage
- Pipeline continuous delivery
 - Deploy the artifacts produced by the CI stage to the target environment
 - Output - a deployed pipeline with the new implementation of the model
- Automated triggering
 - The pipeline is automatically executed in production based on a schedule or in response to a trigger
 - Output - a trained model that is pushed to the model registry
- Model continuous delivery
 - Serve the trained model as a prediction service for the predictions
 - Output - a deployed model prediction service
- Monitoring
 - Collect statistics on the model performance based on live data
 - Output - a trigger to execute the pipeline or to execute a new experiment cycle
- Data analysis
 - The data analysis step is still a manual process for data scientists before the pipeline starts a new iteration of the experiment
 - The model analysis step is also a manual process

MLOps level 2: CI/CD pipeline automation(4)

Continuous integration

- When new code is committed or pushed to the source code repository
 - the pipeline and its components are built, tested, and packaged
- CI process can include the following tests:
 - Unit testing feature engineering logic
 - Unit testing the different methods implemented in model
 - a function that accepts a categorical data column and encode the function as a one-hot feature
 - Testing model training doesn't produce NaN values due to dividing by zero or manipulating small or large values
 - Testing that each component in the pipeline produces the expected artifacts
 - Testing integration between pipeline components

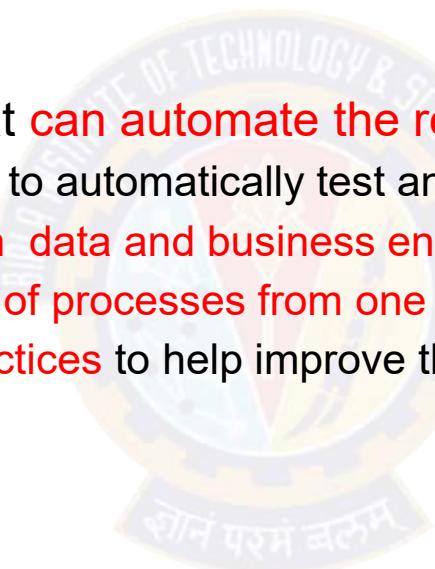
MLOps level 2: CI/CD pipeline automation(5)

Continuous delivery

- System continuously **delivers new pipeline implementations to the target environment**
 - that in turn delivers prediction services of the newly trained model.
- Rapid and reliable continuous delivery of pipelines and models depend on
 - Verifying the **compatibility of the model with the target infrastructure** before model is deployed
 - need to verify that the packages that are required by the model are installed in the serving environment
 - whether the memory, compute, and accelerator resources that are available
 - Testing the prediction **service by calling the service API with the expected inputs**
 - making sure that you get the response that you expect
 - captures problems that might occur when you update the model version and it expects a different input
 - Testing prediction service **performance**, which involves **load testing** the service
 - to capture metrics such as queries per seconds (QPS) and model latency
 - Validating the data either for retraining or batch prediction
 - Verifying that models meet the **predictive performance targets** before they are deployed
 - **Automated deployment to a test environment** - triggered by pushing code to the **development branch**
 - **Semi-automated deployment to a pre-production environment** - triggered by merging code to the **main branch** after reviewers approve the changes
 - **Manual deployment to a production environment** after several successful runs of the pipeline on the pre-production environment

Summary

- Implementing ML in a production environment **doesn't only mean deploying model as an API for prediction!**
- Means deploying an ML pipeline that **can automate the retraining and deployment of new models**
 - Setting up a CI/CD system enables to automatically test and deploy new pipeline implementations
 - Allows coping with rapid changes in data and business environment
 - Don't have to immediately move all of processes from one level to another
 - Can gradually implement these practices to help improve the automation of your ML system development and production





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

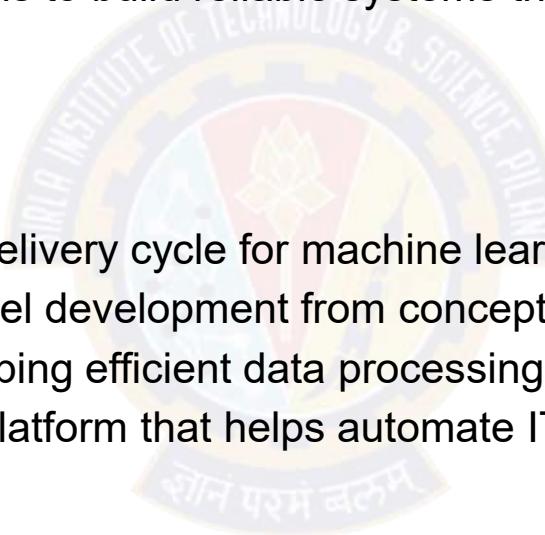
AllTheOps

Pravin Y Pawar

Extracted from
[AllTheOps](#)

MLOps, ModelOps, DataOps and AIOps

- *Ops are rapidly growing in importance as organizations look to leverage the power of artificial intelligence, machine learning and big data
 - Each approach allows organizations to build reliable systems that can effectively process large amounts of data quickly and efficiently
- AllOps
 - MLOps focuses on a continuous delivery cycle for machine learning models through automated pipelines
 - ModelOps is used to manage model development from conception to deployment
 - DataOps provides tools for developing efficient data processing pipelines
 - AIOps is an AI-driven operations platform that helps automate IT processes such as incident resolution

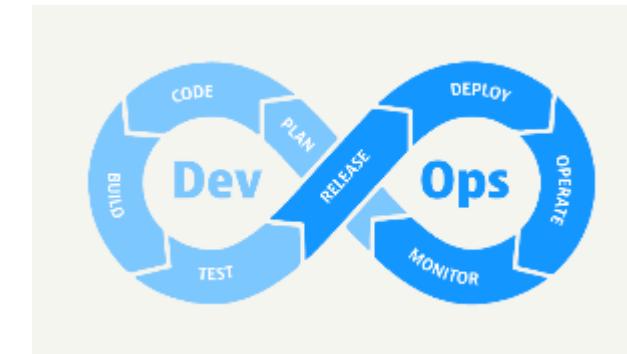
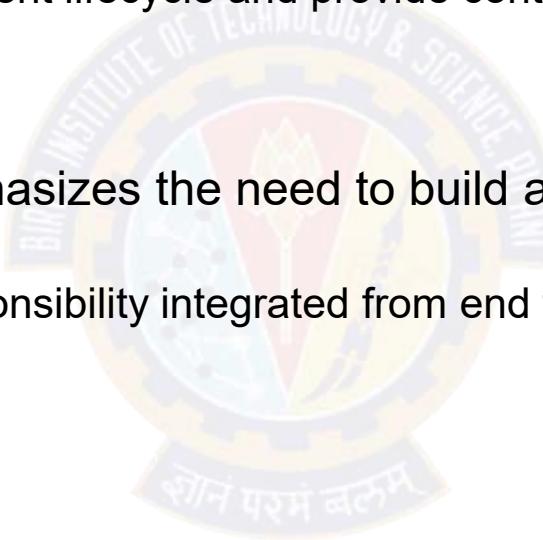


[LinkedIn](#)

DevOps

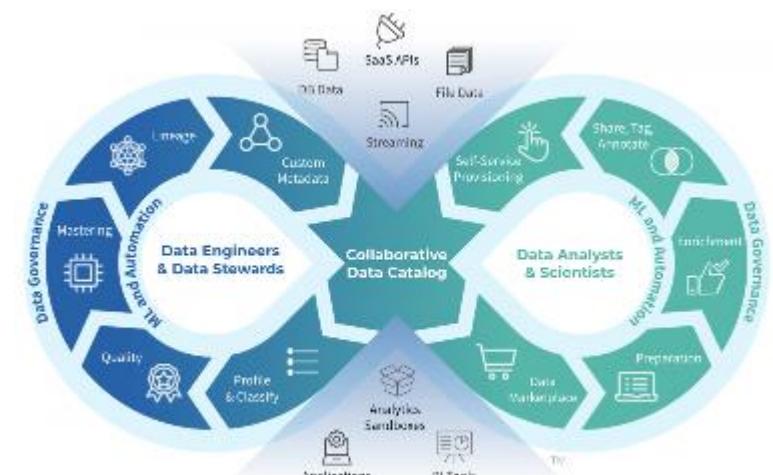
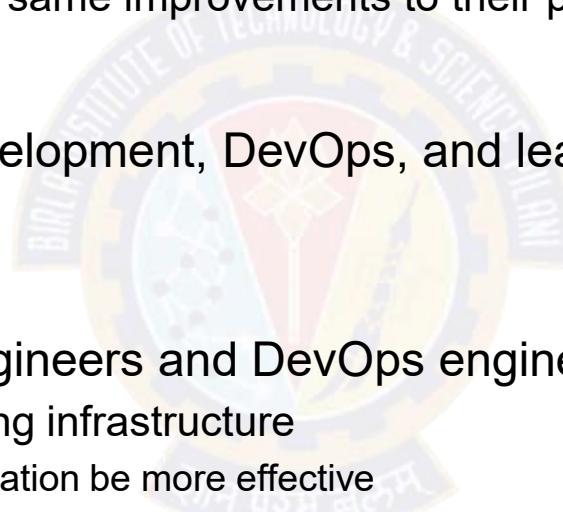
The one that started it all

- DevOps is a set of practices that combines software development (Dev) and information technology operations (Ops)
 - that aims to shorten the development lifecycle and provide continuous delivery with high software quality
- DevSecOps was defined and emphasizes the need to build a security foundation into DevOps initiatives
 - because security is a shared responsibility integrated from end to end,



DataOps

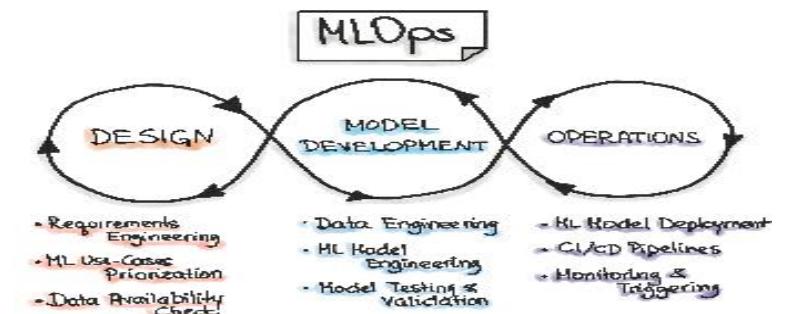
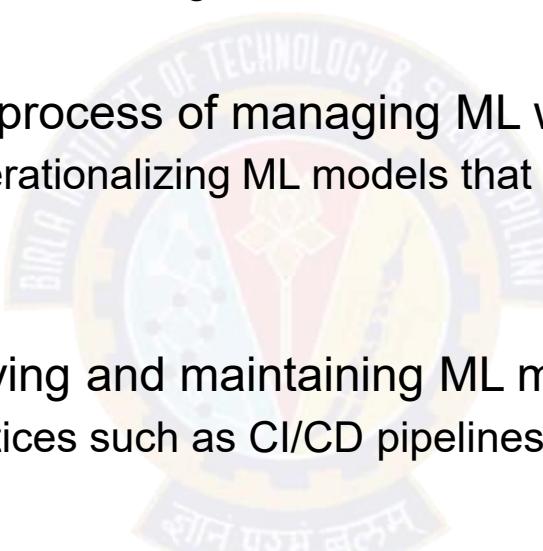
- DataOps was defined when
 - DevOps improved the quality and scalability of software development
 - Data analytics discipline sought to bring same improvements to their practices
- In a nutshell, DataOps applies agile development, DevOps, and lean manufacturing to data analytics (Data) and operations (Ops)
- DataOps is a reference of how Data Engineers and DevOps engineers work together
 - goal of the team is to focus on developing infrastructure
 - that helps people throughout an organization be more effective
 - with their analytics initiatives, whether they are ML or BI-focused
 - means building tools like ETL pipelines and dashboard infrastructure,
 - as well as the infrastructure needed to support ML and BI projects



[source](#)

MLOps

- MLOps is meant to standardize and streamline the lifecycle of machine learning models in production
 - by orchestrating the movement of machine learning models, data, and outcomes among the systems
- MLOps (or ML Operations) refers to the process of managing ML workflows
 - subset of ModelOps that focuses on operationalizing ML models that have been already created or are being actively used in production
- MLOps is the practice of building, deploying and maintaining ML models
 - using established engineering best practices such as CI/CD pipelines
- MLOps has emerged because ML engineers are increasingly being asked
 - to manage their own workflows in production
 - rather than leaving it up to data or infrastructure teams



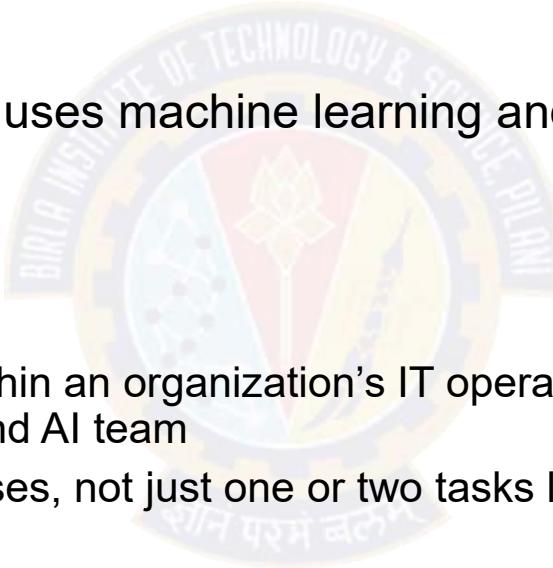
[source](#)

ModelOps

- ModelOps incorporates MLOps, which is the process of managing ML models throughout their lifecycle at an enterprise scale
 - Considered as a superset of MLOps, which refers to the processes involved to operationalize and manage AI models in use in production systems
- The advantage of ModelOps over MLOps is that MLOps focuses on the machine learning models only,
 - whereas Modelops is focused to operationalize all AI models
- The organization looking to set up ModelOps should set up MLOps first before moving on to ModelOps
 - The skills required for ModelOps are the same as MLOps, with some additional skills pertaining to the entire gamut of AI.

AIOps

- AIOps (Artificial Intelligence for IT Operations) is the use of machine learning and other AI technologies to automate many processes that are currently done manually in an organization
- AIOps is similar to MLOps in that it uses machine learning and other AI technologies to automate IT processes
- It is different from MLOps in that
 - the process automation occurs within an organization's IT operations department instead of an organization's machine learning and AI team
 - uses AI to automate many processes, not just one or two tasks like MLOps does





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

MLOps life-cycle, process and capabilities

Pravin Y Pawar

Google's Practitioners guide to MLOps:
A framework for continuous delivery and automation
of machine learning.

Google Cloud's AI Adoption Framework

- Google Cloud's AI Adoption Framework provides guidance for technology leaders
 - who want to build an effective artificial intelligence (AI) capability in order to transform their business
 - covers AI challenges around people, data, technology, and process,
 - structured in six different themes: learn, lead, access, secure, scale, and automate
- Scale concerns the extent to which use cloud managed ML services that scale with large amounts of data and large numbers of data processing and ML jobs, with reduced operational overhead
- Automate concerns the extent to which able to deploy, execute, and operate technology for data processing and ML pipelines in production efficiently, frequently, and reliably
- Organizations can use the framework to identify gaps in building an integrated ML platform
 - focus on the scale and automate themes from Google's AI Adoption Framework
- The decision about whether (or to which degree) to adopt each of these processes and capabilities in your organization depends on your business context
 - For example, you must determine the business value that the framework creates when compared to the cost of purchasing or building capabilities (for example, the cost in engineering hours)

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 a cohort of organizations that began AI pilots before 2019 have not been able to deploy even a single application in production
 - Algorithmia's survey of the state of enterprise machine learning found that 55% of companies surveyed have not deployed an ML model
- This is due to a variety of issues
 - Teams engage in a high degree of manual and one-off work
 - do not have reusable or reproducible components,
 - processes involve difficulties in handoffs between data scientists and IT
- Deloitte identified **lack of talent and integration issues** as factors that can stall or derail AI initiatives
- Algorithmia's survey highlighted that **challenges in deployment, scaling, and versioning efforts** still hinder teams from getting value from their investments in ML
- Capgemini Research noted that **the top three challenges** faced by organizations in achieving deployments at scale are **lack of mid- to senior-level talent, lack of change-management processes, and lack of strong governance models for achieving scale.**

Complexities of ML application

- Complexities include the following:
 - Preparing and maintaining high-quality data for training ML models
 - Tracking models in production to detect performance degradation
 - Performing ongoing experimentation of new data sources, ML algorithms, and hyperparameters, and then tracking these experiments
 - Maintaining the veracity of models by continuously retraining them on fresh data
 - Avoiding training-serving skews that are due to inconsistencies in data and in runtime dependencies between training environments and serving environments
 - Handling concerns about model fairness and adversarial attacks

ML engineering

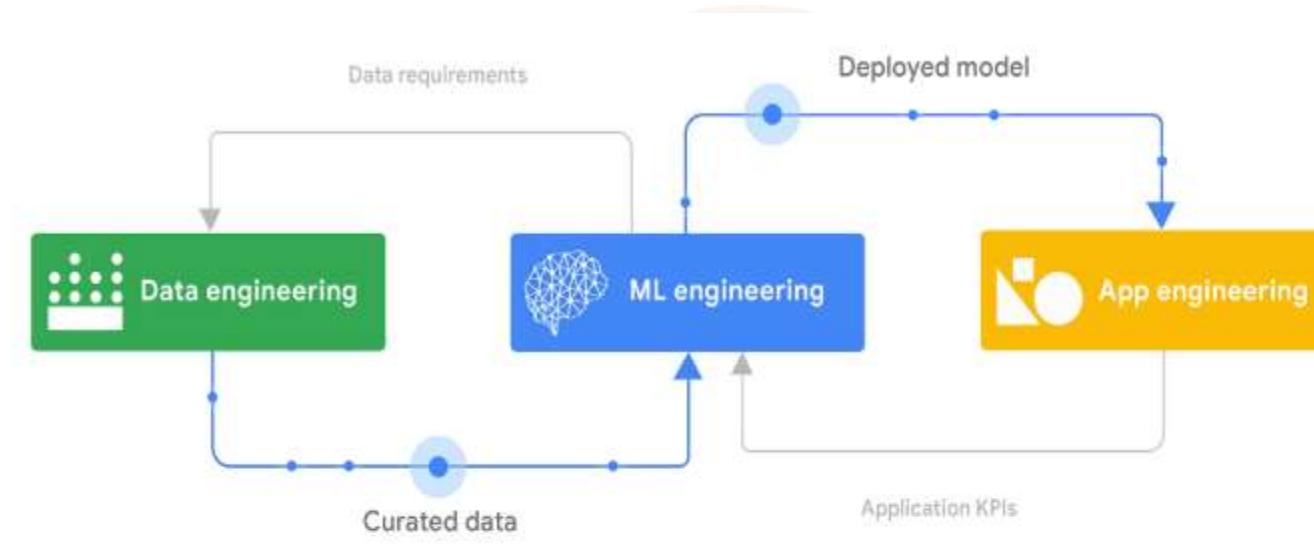
- The common theme is that ML systems cannot be built in an ad hoc manner,
 - isolated from other IT initiatives like DataOps and DevOps
- Also cannot be built without adopting and applying sound software engineering practices,
 - while taking into account the factors that make operationalizing ML different from operationalizing other types of software
- Organizations need an automated and streamlined ML process
 - help the organization successfully deploy ML models in production
 - helps manage risk when organizations scale the number of ML applications to more use cases in changing environments
 - helps ensure that the applications are still in line with business goals
- **This is where ML engineering can be essential!**
 - ML engineering is at the **center of building ML-enabled systems**, which **concerns the development and operationalizing of production-grade ML systems**
 - provides a **superset** of the discipline of software engineering that **handles the unique complexities of the practical applications of ML**

MLOps

- MLOps is a methodology for ML engineering that
 - unifies ML system development (the ML element) with ML system operations (the Ops element)
 - 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 in the way that DevOps and DataOps support application engineering and data engineering (analytics)!
 - When deploy an ML model, also need to worry about changes in the data, changes in the model, users trying to game the system, and so on
- MLOps practices can result in the following benefits over systems that do not follow MLOps practices:
 - Shorter development cycles, and as a result, shorter time to market
 - Better collaboration between teams
 - Increased reliability, performance, scalability, and security of ML systems
 - Streamlined operational and governance processes
 - Increased 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

Building an ML-enabled system(2)

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

- Data engineering involves ingesting, integrating, curating, and refining data
 - to facilitate a broad spectrum of operational tasks, data analytics tasks, and ML tasks
 - If an organization does not have robust data engineering processes and technologies,
 - it might not be set up for success with downstream business intelligence, advanced analytics, or ML projects
- ML models are built and deployed in production using curated data that is usually created by the data engineering team
 - The models do not operate in silos, they are components of, and support, a large range of application systems,
 - such as business intelligence systems, line of business applications, process control systems, and embedded systems
- Integrating an ML model into an application is a critical task involves
 - making sure first that the deployed model is used effectively by the applications,
 - monitoring model performance
 - should also collect and monitor relevant business KPIs which helps in understanding the impact of the ML model on the business and adapt accordingly

The MLOps lifecycle

Encompasses seven integrated and iterative processes



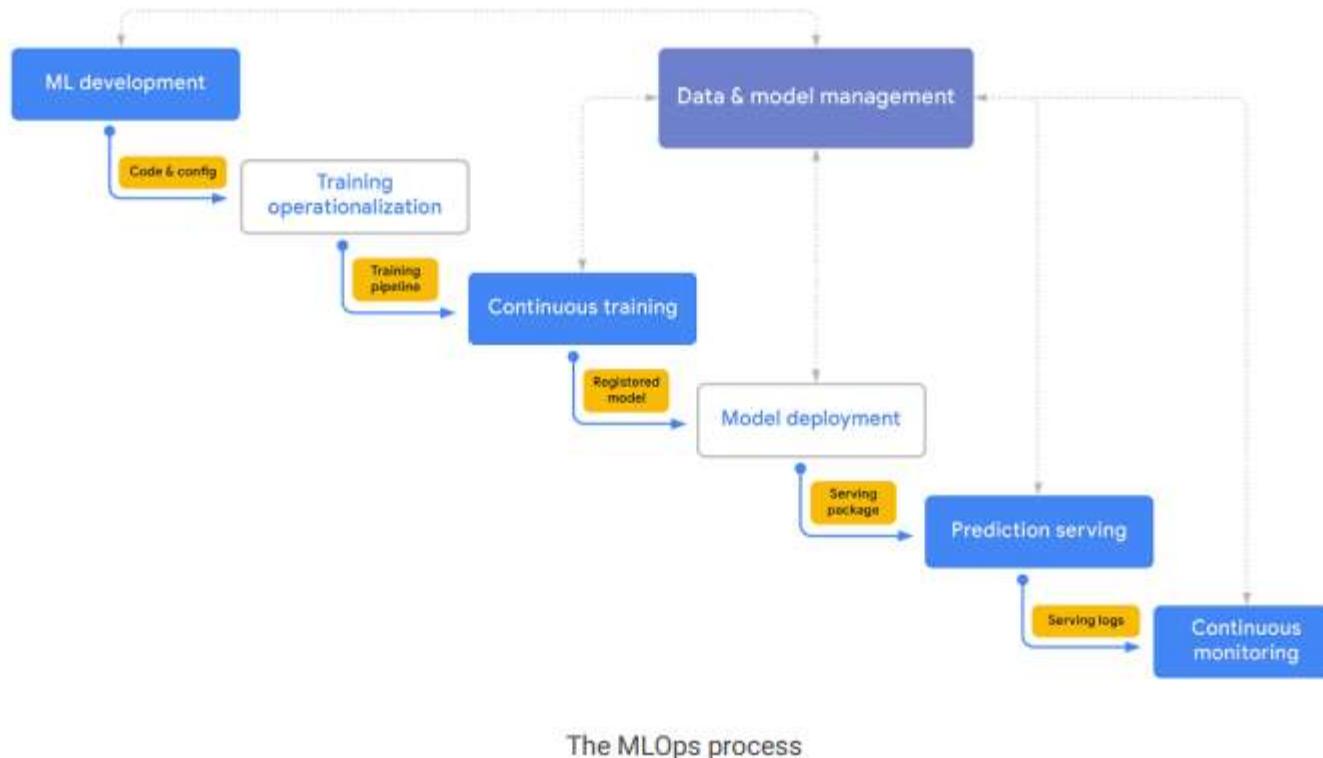
The MLOps lifecycle(2)

The processes can consist of

- ML development
 - concerns experimenting and developing a robust and reproducible model training procedure (training pipeline code),
 - which consists of multiple tasks from data preparation and transformation to model training and evaluation
- Training operationalization
 - concerns automating the process of packaging, testing, and deploying repeatable and reliable training pipelines
- Continuous training
 - concerns repeatedly executing the training pipeline in response to new data or to code changes, or on a schedule
 - potentially with new training settings
- Model deployment
 - concerns packaging, testing, and deploying a model to a serving environment for online experimentation and production serving
- Prediction serving
 - is about serving the model that is deployed in production for inference
- Continuous monitoring
 - is about monitoring the effectiveness and efficiency of a deployed model
- Data and model management
 - central, cross-cutting function for governing ML artifacts to support auditability, traceability, and compliance
 - also promote 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



MLOps: An end-to-end workflow(2)

not a waterfall workflow that has to sequentially pass through all the processes

- 1. The core activity during ML development phase is experimentation
 - As data scientists and ML researchers prototype model architectures and training routines,
 - they create labeled datasets, and they use features and other reusable ML artifacts
 - The primary output is a formalized training procedure,
 - which includes data preprocessing, model architecture, and model training settings
- 2. If the ML system requires continuous training (repeated retraining of the model),
 - the training procedure is operationalized as a training pipeline
 - 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 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 that are 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

MLOps: An end-to-end workflow(3)

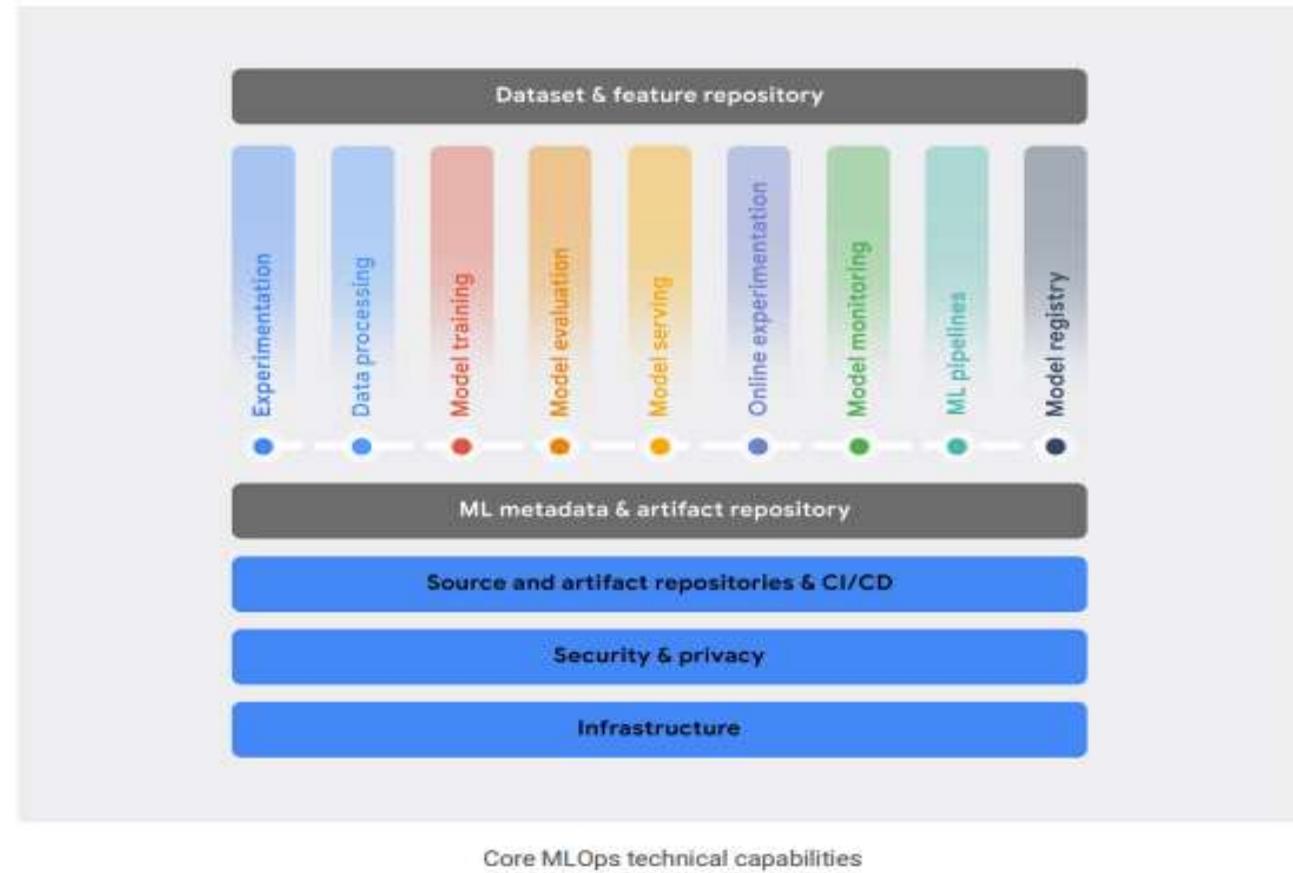
processes can be skipped, or the flow can repeat a given phase or a subsequence of the processes

- 4. The registered model is annotated, reviewed, and approved for release and is then deployed to a production environment
 - might be relatively opaque if using a no-code solution,
 - can involve building a custom CI/CD pipeline for progressive delivery
- 5. The deployed model serves predictions using the deployment pattern that is 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—for example, data and concept drift
 - can also be monitored for efficiency metrics like latency, throughput, hardware resource utilization, and execution errors

MLOps capabilities

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
 - which can be provided by a single integrated ML platform
 - Alternatively, they can be created by combining vendor tools that each are best suited to particular tasks
- In most cases, the processes are deployed in stages rather than all at once in a single deployment
 - For example, many organizations start by focusing on the processes for ML development, model deployment, and prediction serving
 - For these organizations, continuous training and continuous monitoring might not be necessary
 - if they are piloting a relatively small number of ML systems



MLOps capabilities(2)

Foundational and ML Specific

- Foundational capabilities
 - are required in order to support any IT workload, such as a reliable, scalable, and secure compute infrastructure
 - already have investments in these capabilities and can benefit by taking advantage of them for ML workflows
 - might span multiple clouds, or even operate partially on-premises
 - Ideally, this would include advanced capabilities such as specialized ML accelerators
- Standardized configuration management and CI/CD capabilities
 - required to build, test, release, and operate software systems rapidly and reliably, including ML systems
- MLOps capabilities
 - include 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 are an ML metadata and artifact repository and an ML dataset and feature repository

MLOps capability - Experimentation

- Lets data scientists and ML researchers collaboratively
 - perform exploratory data analysis,
 - create prototype model architectures
 - and implement training routines
- An ML environment should also let them write modular, reusable, and testable source code
 - that is version controlled
- Key functionalities in experimentation:
 - Provide notebook environments that are integrated with version control tools like Git
 - Track experiments, including information about the data, hyperparameters, and evaluation metrics for reproducibility and comparison
 - Analyze and visualize data and models
 - Support exploring datasets, finding experiments, and reviewing implementations
 - Integrate with other data services and ML services in platform

MLOps capability - Data processing

- Allows to prepare and transform large amounts of data for ML at scale in ML development,
 - in continuous training pipelines, and in prediction serving
- Key functionalities:
 - Support interactive execution (for example, from notebooks) for quick experimentation and for long-running jobs in production
 - Provide data connectors to a wide range of data sources and services, as well as data encoders and decoders for various data structures and formats
 - Provide both rich and efficient data transformations and ML feature engineering for structured (tabular) and unstructured data (text, image, and so on)
 - Support scalable batch and stream data processing for ML training and serving workloads

MLOps capability - Model training

- Lets you efficiently and cost-effectively run powerful algorithms for training ML models
- Model training should be able to scale with size of both models and datasets that are used for training
- Key functionalities:
 - Support common ML frameworks and support custom runtime environments
 - Support large-scale distributed training with different strategies for multiple GPUs and multiple workers
 - Enable on-demand use of ML accelerators
 - Allow efficient hyperparameter tuning and target optimization at scale
 - Ideally, provide built-in automated ML (AutoML) functionality,
 - including automated feature selection and engineering as well as automated model architecture search and selection

MLOps capability - Model evaluation

- Lets you assess the effectiveness of model,
 - interactively during experimentation and automatically in production
- Key functionalities:
 - Perform batch scoring of models on evaluation datasets at scale
 - Compute pre-defined or custom evaluation metrics for model on different slices of the data
 - Track trained-model predictive performance across different continuous-training executions
 - Visualize and compare performances of different models
 - Provide tools for what-if analysis and for identifying bias and fairness issues
 - Enable model behavior interpretation using various explainable AI techniques

MLOps capability - Model serving

- Lets you deploy and serve models in production environments
- Key functionalities:
 - Provide support for low-latency, near-real-time (online) prediction and high-throughput batch (offline) prediction
 - Provide built-in support for common ML serving frameworks and for custom runtime environments
 - Enable composite prediction routines, where multiple models are invoked hierarchically or simultaneously
 - Allow efficient use of ML inference accelerators with autoscaling to match spiky workloads and to balance cost with latency
 - Support logging of prediction serving requests and responses for analysis

MLOps capability - Online experimentation

- Lets you understand how newly trained models perform in production settings compared to the current models
 - before you release the new model to production
- For example, using a small subset of the serving population,
 - use online experimentation to understand the impact that a new recommendation system has on click-throughs and on conversation rates
 - results of online experimentation should be integrated with the model registry capability to facilitate the decision about releasing the model to production
- Online experimentation enhances the reliability of ML releases by helping in deciding to discard ill-performing models and to promote well-performing ones
- Key functionalities:
 - Support canary and shadow deployments
 - Support traffic splitting and A/B tests
 - Support multi-armed bandit (MAB) tests

MLOps capability - Model monitoring

- Lets you track the efficiency and effectiveness of the deployed models in production
 - to ensure predictive quality and business continuity
- Informs if models are stale and need to be investigated and updated
- Key functionalities:
 - Measure model efficiency metrics like latency and serving-resource utilization
 - Detect data skews, including schema anomalies and data and concept shifts and drifts
 - Integrate monitoring with the model evaluation capability for continuously assessing
 - the effectiveness performance of the deployed model when ground truth labels are available

MLOps capability - ML pipelines

- Lets you instrument, orchestrate, and automate complex ML training and prediction pipe-lines in production
 - ML workflows coordinate different components, where each component performs a specific task in the pipeline
- Key functionalities:
 - Trigger pipelines on demand, on a schedule, or in response to specified events
 - Enable local interactive execution for debugging during ML development
 - Integrate with the ML metadata tracking capability to capture pipeline execution parameters and to produce artifacts
 - Provide a set of built-in components for common ML tasks and also 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

- Lets you govern the lifecycle of the ML models in a central repository
 - ensures the quality of the production models and enables model discovery
- Key functionalities:
 - Register, organize, track, and version trained and deployed ML models
 - Store model metadata and runtime dependencies for deployability
 - Maintain model documentation and reporting—for example, using model cards
 - Integrate with the model evaluation and deployment capability and track online and offline evaluation metrics for the models
 - Govern the model launching process: review, approve, release, and roll back

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 their time
- Key functionalities:
 - 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 and model training, and for 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
 - Feature repository might contain an entity called customer, which includes features
 - like age group, postal code, and gender
 - 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:
 - 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



Thank You!

In our next session:



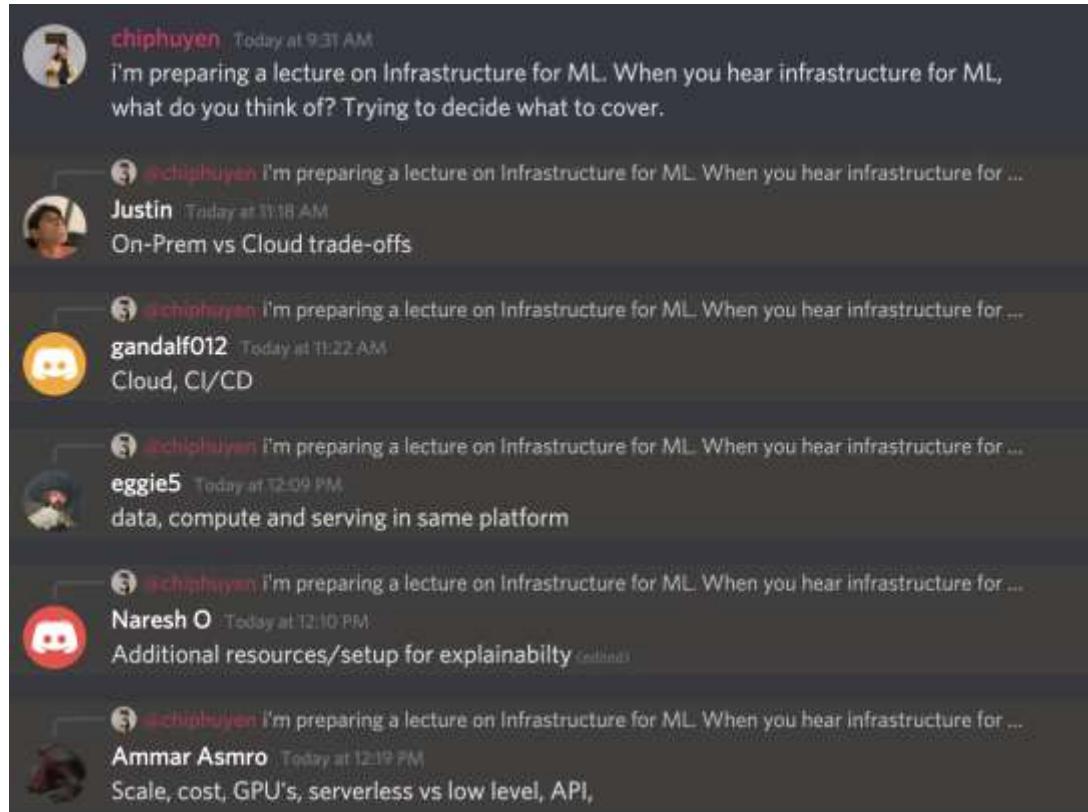
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Infrastructure: Storage and Compute

Pravin Y Pawar

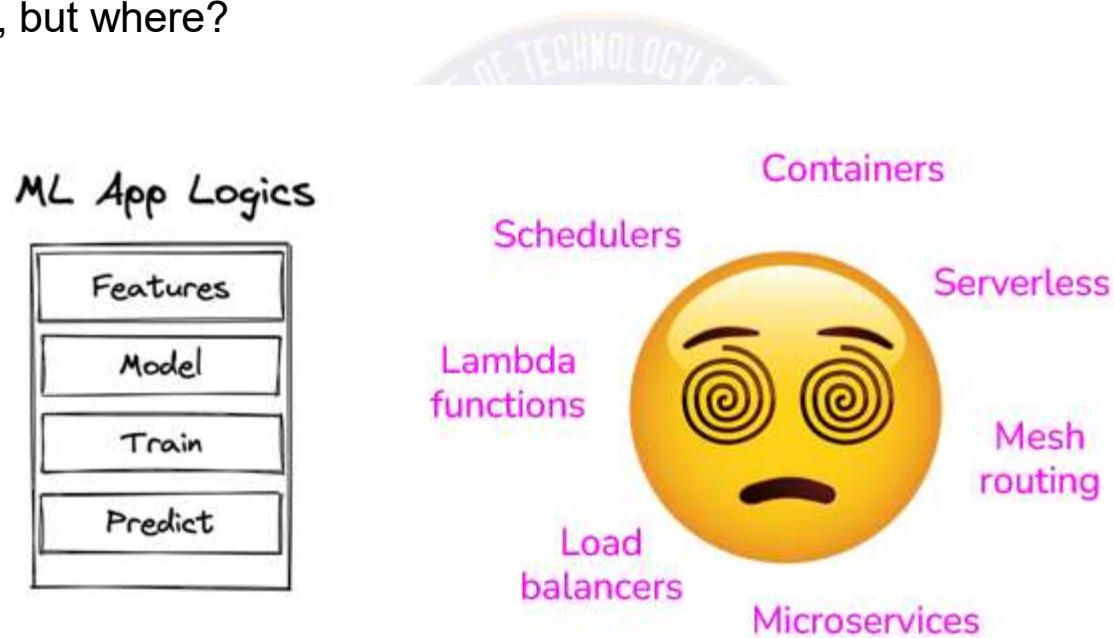
From “Designing Machine Learning Systems”
By Chip Huyen

What does infrastructure mean?



ML systems are complex

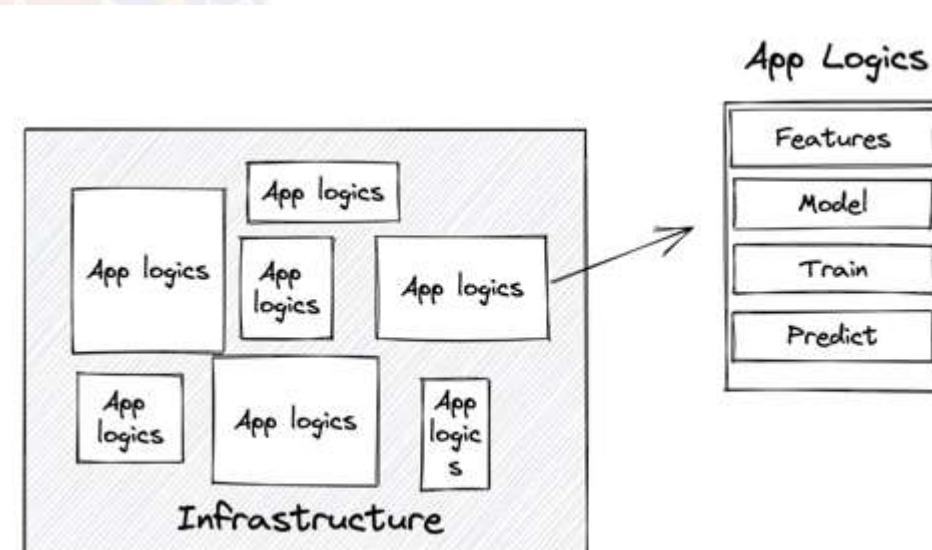
- More components
 - A request might jump 20-30 hops before response
 - A problem occurs, but where?



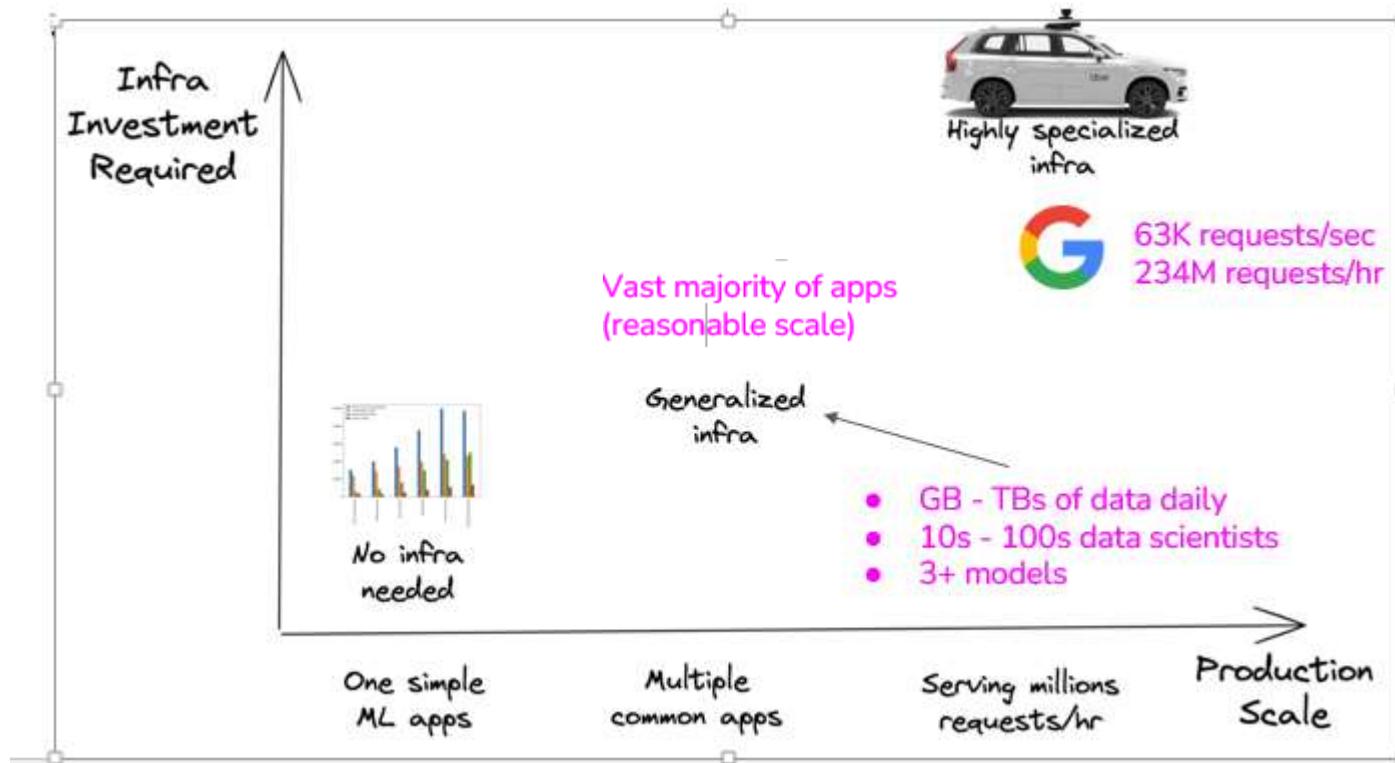
- More complex systems, better infrastructure needed

ML Infrastructure

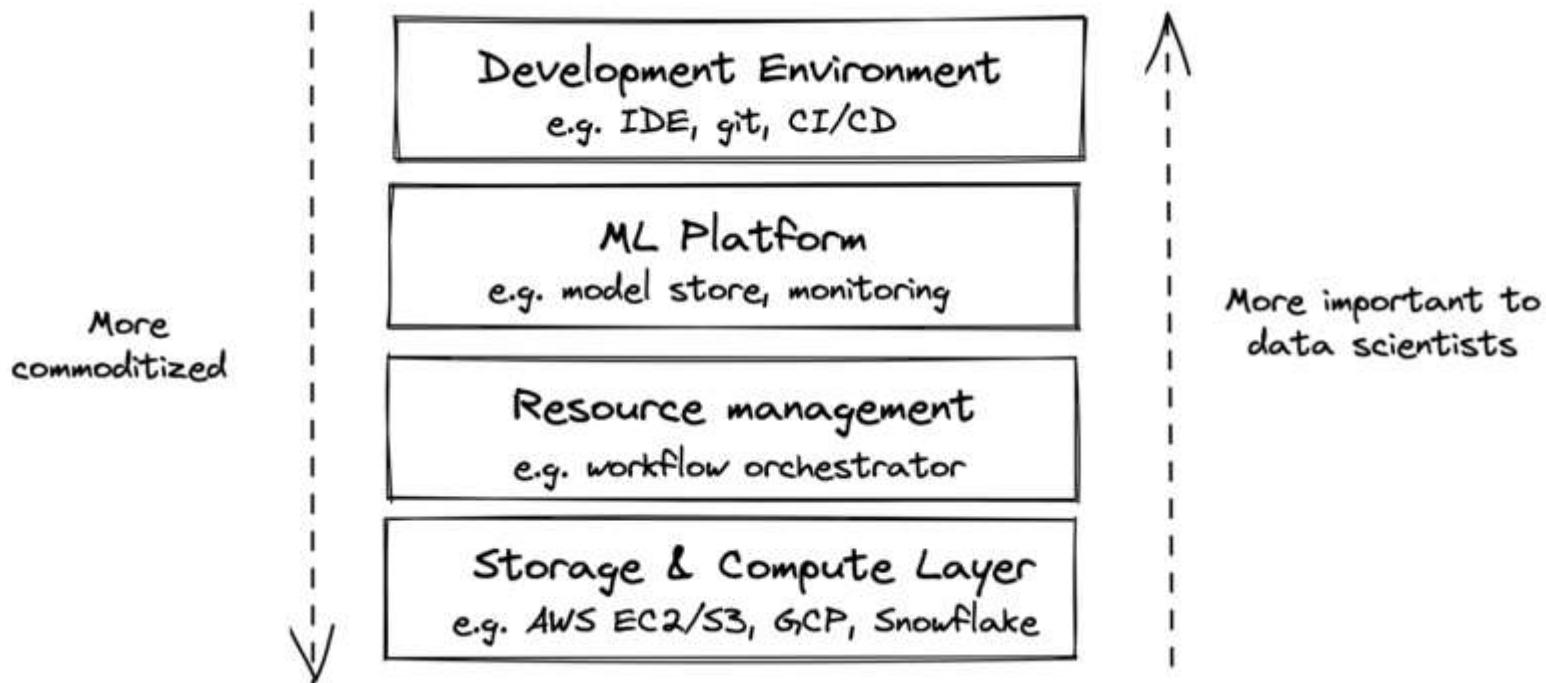
- Infrastructure
 - the set of fundamental facilities and systems that support the sustainable functionality of households and firms.
- ML infrastructure
 - the set of fundamental facilities that **support the development and maintenance of ML systems**



Every company's infrastructure needs are different



Infrastructure Layers



Storage Layer

Storage

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

Part 2. Data Systems Fundamentals

Data Sources

Data Formats

- JSON
- Row-major vs. Column-major Format
- Text vs. Binary Format

Data Models

- Relational Model
- NoSQL
 - Document Model
 - Graph Model

Structured vs. Unstructured Data

Data Storage Engines and Processing

- Transactional and Analytical Processing
- ETL: Extract, Transform, Load
 - ETL to ELT



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

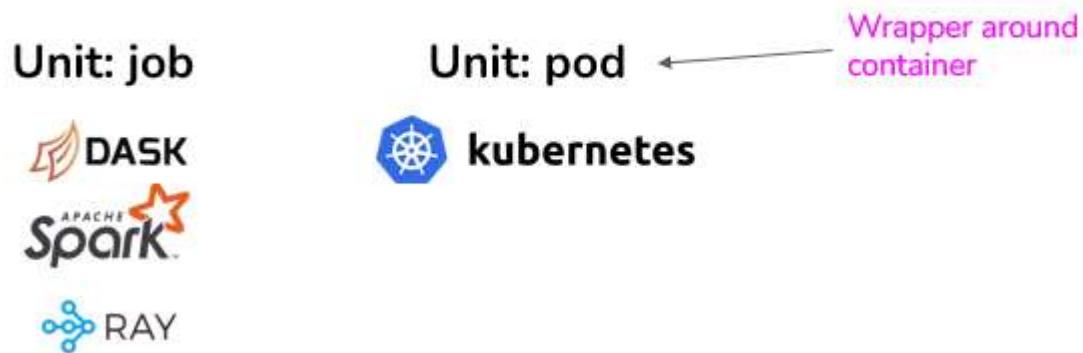
Compute layer: engine to execute your jobs

- 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
- Multiple CPUs can be joined to form a large compute unit to execute a large job



Compute layer: how to execute jobs

1. Load data into memory
2. Perform operations on that data
 - a. 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

Important metrics of compute layer:

1. Memory
2. Speed of computing ops

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
- “A Cloud TPU v2 can perform up to 180 teraflops, and the TPU v3 up to 420 teraflops.”
 - Google, 2021
- Contentious
 - What exactly is an ops?
 - If 2 ops are fused together, is it 1 or 2 ops?
 - Peak perf at 1 teraFLOPS doesn’t mean your app will run at 1 teraFLOPS

Compute layer: utilization

- Utilization = actual FLOPS / peak FLOPS
 - If peak 1 trillion FLOPS but job runs 300 billion FLOPS
 - \rightarrow utilization = 0.3
 - Dependent on how fast data can be loaded into memory
- Tensor Cores are very fast. So fast ... that they are idle most of the time as they are waiting for memory to arrive from global memory. For example, during BERT Large training, which uses huge matrices — the larger, the better for Tensor Cores — we have utilization of about 30%.

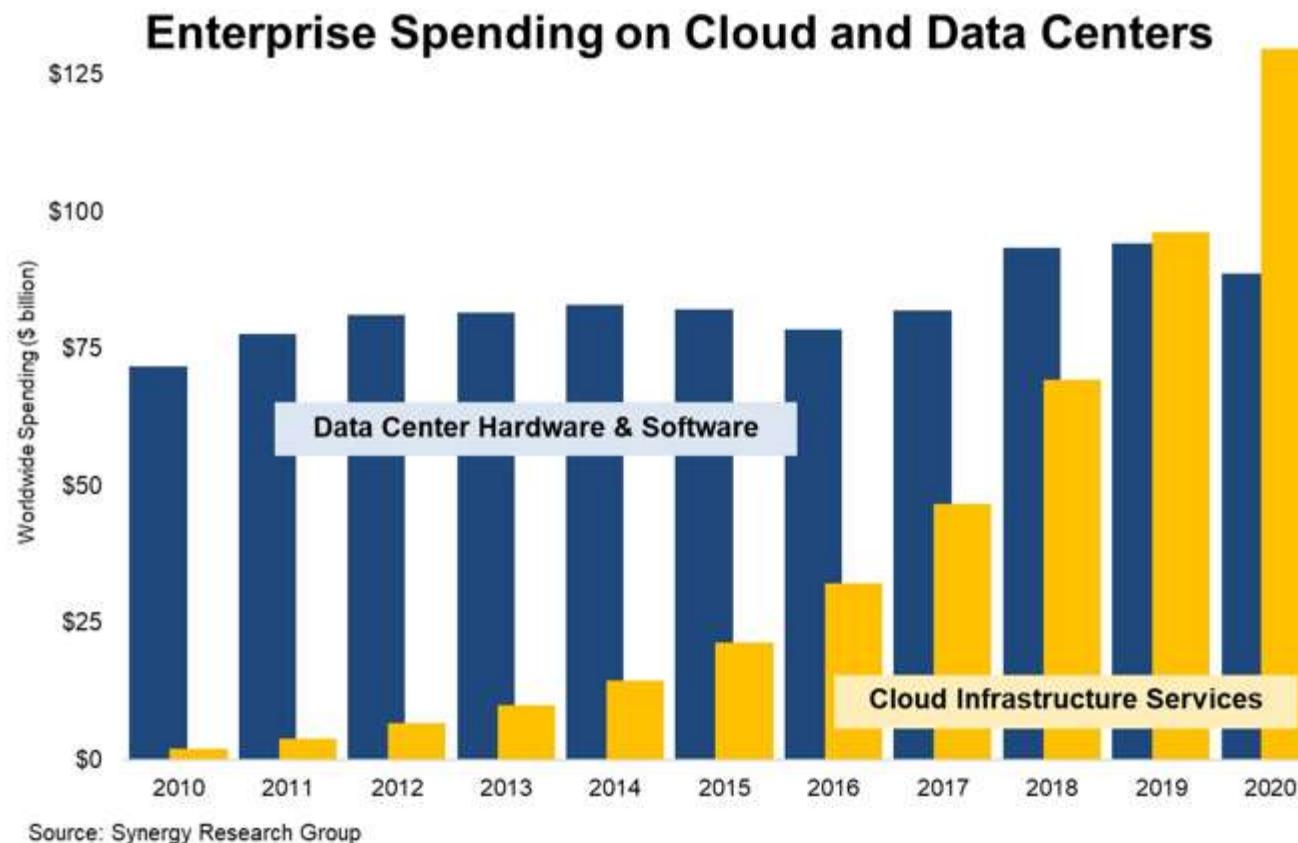
The higher
the better



Tim Dettmers, 2020

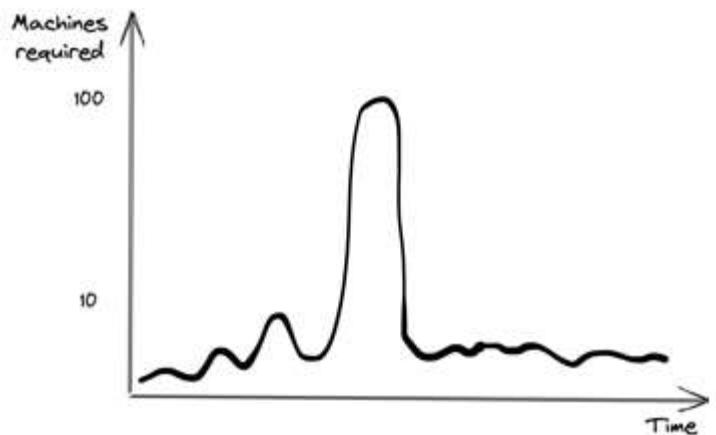
Public Cloud vs. Private Data Centers

- Like storage, compute is largely commoditized



Benefits of cloud

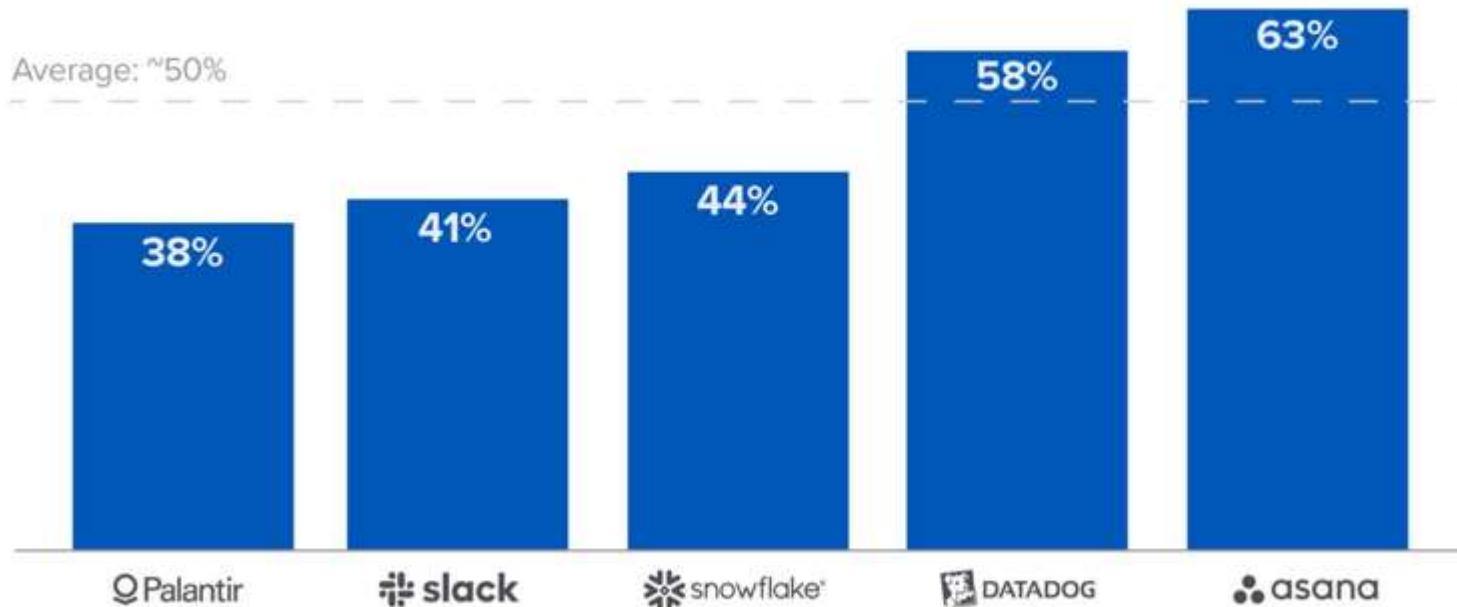
- Easy to get started
- Appealing to variable-sized workloads
 - Private: would need 100 machines upfront, most will be idle most of the time
 - Cloud: pay for 100 machines only when needed



Drawbacks of cloud: cost

- Cloud spending: ~50% cost of revenue

Estimated Annualized Committed Cloud Spend as % of Cost of Revenue



[The Cost of Cloud, a Trillion Dollar Paradox | Andreessen Horowitz \(2021\)](#)

Source: Company S-1 and 10K filings

Drawbacks of cloud: cost



“Across 50 of the top public software companies currently utilizing cloud infrastructure, an estimated \$100B of market value is being lost ... due to cloud impact on margins — relative to running the infrastructure themselves.”

[The Cost of Cloud, a Trillion Dollar Paradox | Andreessen Horowitz \(2021\)](#)



Cloud repatriation

- Process of moving workloads from cloud to private data centers



Multicloud strategy

- To optimize cost
- To avoid cloud vendor lock-in

“81% of respondents said they are working with two or more providers”

- [Gartner](#) (2019)



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

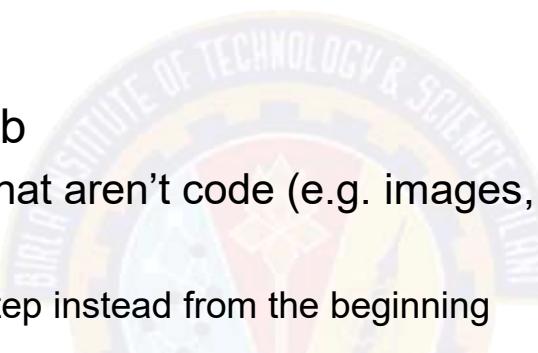
Development Environments

Pravin Y Pawar

From “Designing Machine Learning Systems”
By Chip Huyen

Development Environment: Notebooks

- 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 from the beginning

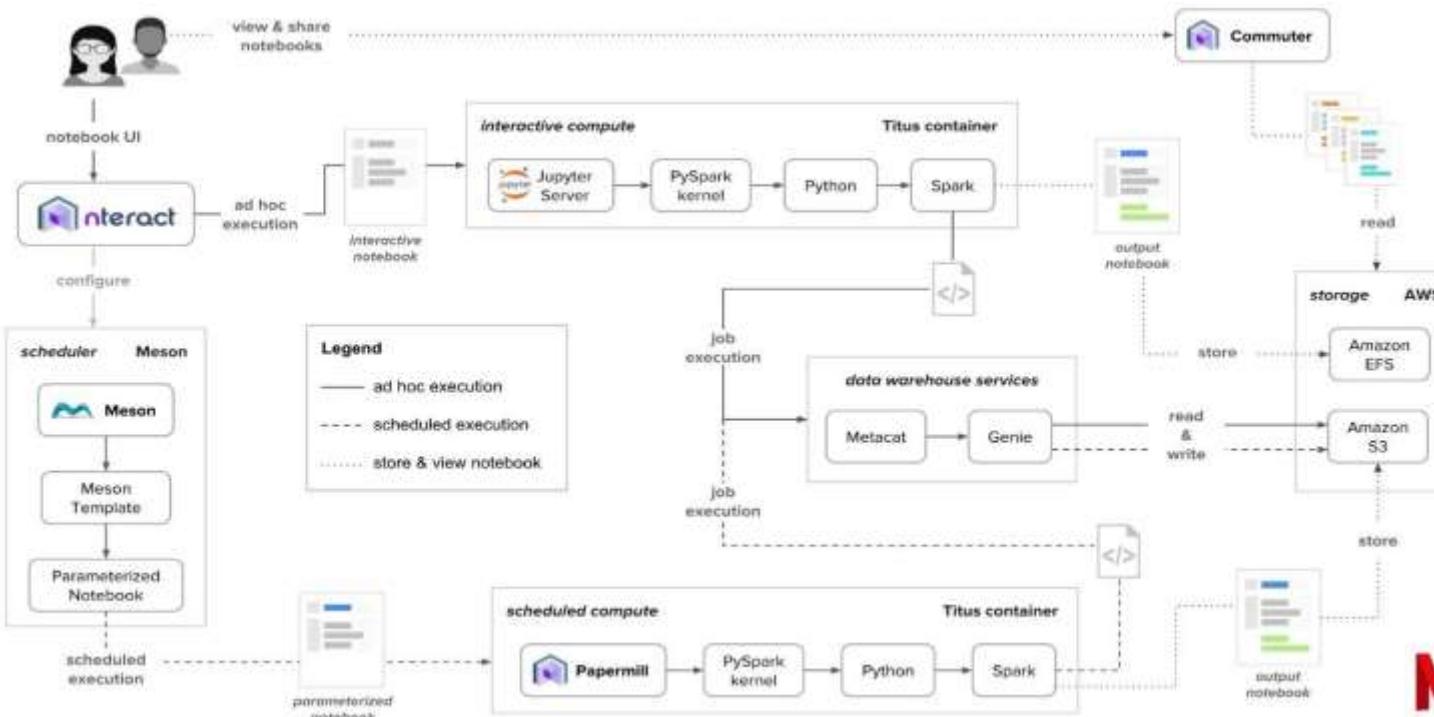


```
In [1]: import pandas as pd
In [2]: fname = "large-dataset.csv"
In [3]: df = pd.read_csv(fname)
In [4]: features = df[["Timestamp", "Cost"]]

KeyError Traceback (most recent call last)
-/miniconda3/envs/stove39/lib/python3.9/site-packages/pandas/core/indexes/base.py
ance)
 3360         try:
-> 3361             return self._engine.get_loc(casted_key)
 3362         except KeyError as err:
```

Development Environment @Netflix

Notebook at Netflix



Development Environment

- Versioning
 - Git: code versioning
 - DVC: data versioning
 - WandB: experiment versioning
- CI/CD test suite
 - test your code before pushing it to staging/prod



Dev env: underestimated



“if you have time to set up only one piece of infrastructure well, make it the development environment for data scientists.”

Ville Tuulos, [Effective Data Science Infrastructure](#) (2022)



Standardize dev environments

- Standardize dependencies with versions
- Standardize tools & versions
- Standardize hardware: cloud dev env



Standardize dev environments

- Standardize dependencies with versions

```
1 -f https://download.pytorch.org/whl/torch_stable.html
2 torch==1.10.0+cpu
3 numpy==1.21.3
4 pandas==1.3.5
5 scikit-learn==1.0.2
6 boto3==1.20.8
7 clickhouse-driver==0.2.2
8 clickhouse-sqlalchemy==0.1.7
9云pickle==2.0.0
10 dataclasses-json==0.5.4
11 dbt-clickhouse==0.21.0
12 fastavro==1.4.7
13 httpx==0.21.0
14 matplotlib==3.4.3
15 notebook==6.4.5
16 confluent-kafka==1.7.0
17 python-dotenv==0.19.2
18 requests==2.26.0
```



Standardize dev environments

- 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



kubernetes





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Runtime Environments

Pravin Y Pawar

Adapted from “Introducing MLOps” by
Mark Treveil and Dataiku Team

Preparing for Production

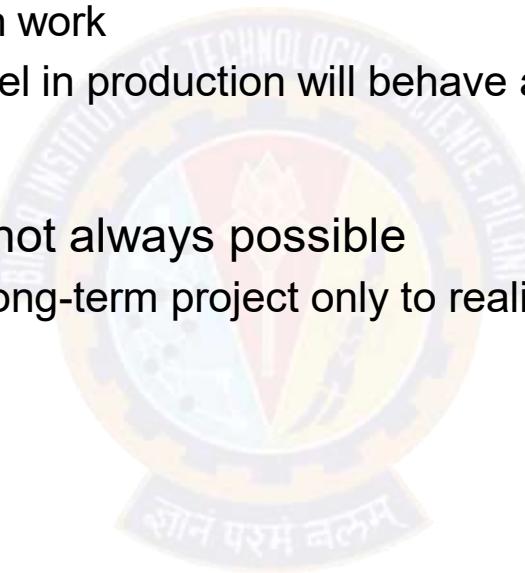
- Confirming that something works in the laboratory has never been a sure sign it will work well in the real world
 - machine learning models are no different
- Not only is the production environment typically very different from the development environment,
 - but the commercial risks associated with models in production are much greater
- Important that the complexities of the transition to production are understood and tested
 - the potential risks have been adequately mitigated

Runtime Environments

- The first step in sending a model to production is making sure it's technically possible!
- Production environments take a wide variety of forms:
 - custom-built services,
 - data science platforms,
 - dedicated services like TensorFlow Serving,
 - low-level infrastructure like Kubernetes clusters,
 - JVMs on embedded systems, etc.
- Ideal MLOps systems favor rapid, automated deployment over labor-intensive processes,
 - and runtime environments can have a big effect on which approach prevails

Idle Scenario vs Reality

- Ideally, models running in the development environment would be validated and sent as is to production
 - minimizes the amount of adaptation work
 - improves the chances that the model in production will behave as it did in development
- Unfortunately, this ideal scenario is not always possible
 - not unheard of that teams finish a long-term project only to realize it can't be put in production



Adaptation from Development to Production Environments

- On One end of the spectrum, the development and production platforms are from the same vendor or are otherwise interoperable
 - dev model can run without any modification in production
 - technical steps required to push the model into production are reduced to a few clicks or commands
- On the other end of the spectrum, there are cases where the model needs to be reimplemented
 - from scratch—possibly by another team possibly in another programming language
- Still the reality in many organizations - lack of appropriate tooling and processes
 - handing over a model for another team to reimplement and adapt for the production environment means that model won't reach production for months(maybe years), if at all
- In all cases, it is crucial to perform validation in an environment that mimics production as closely as possible
 - rather than in the development environment

Tooling considerations

- The format required to send to production should be considered early as it may have
 - a large impact on the model itself
 - on quantity of work required to productionalize it
- For example, when a model is developed using scikit-learn (Python) and production is a Java-based environment
 - expects PMML or ONNX as input, conversion is obviously required
- Teams should set up tooling while developing the model,
 - ideally before the first version of the model is finished or even started
- Failure to create this pipeline up front would block the validation process
 - of course, final validation should not be performed on the scikit-learn model
 - as it's not the one that will be put into production

Performance considerations

Common reason for conversion required is for performance

- For example, a Python model will typically have higher latency for scoring a single record than its equivalent converted to C++.
 - The resulting model will likely be dozens of times faster
- Performance also comes into play when the production model must run on a low-power device
 - In Deep neural networks, trained models can become extremely large with billions or hundreds of billions of parameters
 - Running them on small devices is simply impossible,
 - Running them on standard servers can be slow and expensive
- For these models, an optimized runtime is not enough
 - To obtain better performance, the model definition must be optimized
 - One solution is to use compression techniques
 - quantization
 - pruning
 - distillation

Data Access Before Validation and Launch to Production

technical aspect that needs to be addressed before validation and launch to production

- For example, a model evaluating apartment prices may use average market price in a zip code area
 - User or the system requesting the scoring will probably not provide this average
 - would most likely provide simply the zip code, meaning a lookup is necessary to fetch the value of the average
- In some cases, data can be frozen and bundled with the model
- When this is not possible, the production environment should access a database
 - have to have the
 - appropriate network connectivity, libraries, or drivers required to communicate with the data storage installed
 - authentication credentials stored in some form of production configuration
- Managing this setup and configuration can be quite complex in practice since
 - requires appropriate tooling and collaboration
 - in particular to scale to more than a few dozen models



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Machine Learning Platform

Pravin Y Pawar

Adapted from
"Building a Machine Learning Platform [Definitive Guide]"
by Stephen Oladele

The need

- Moving across the typical machine learning lifecycle can be a nightmare!
- It's a lot -
 - From gathering and processing data to building models through experiments,
 - deploying the best ones,
 - and managing them at scale for continuous value in production
- As the number of ML-powered apps and services grows,
 - gets overwhelming for data scientists and ML engineers to build and deploy models at scale
- Supporting the operations of data scientists and ML engineers requires to reduce—or eliminate—
 - the engineering overhead of building, deploying, and maintaining high-performance models
- To do that, you'd need to take a systematic approach to MLOps—enter platforms!

Machine learning platforms

- Increasingly looking to be the “fix” to successfully consolidate all the components of MLOps
 - from development to production!
- Platform give the tools and infrastructure needed by teams to build and operate models at scale
 - also applies standard engineering and MLOps principles to all use cases
- The catch:
 - understanding what makes a platform successful and building it is no easy feat!
 - With so many tools, frameworks, practices, and technologies available,
 - can be overwhelming to know where to start!

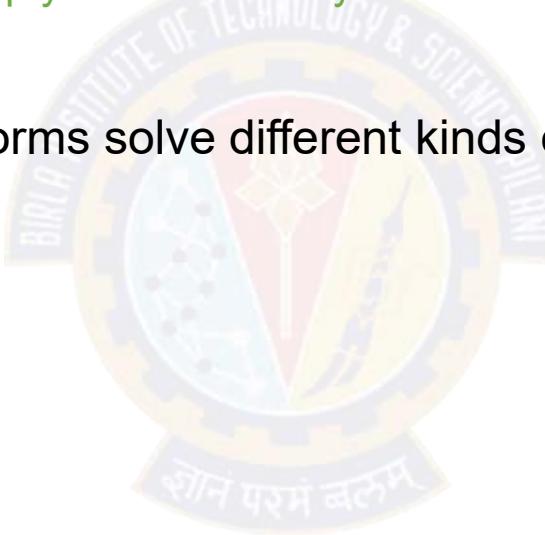
What is a machine learning platform?

- An ML platform **standardizes the technology stack** for data team around best practices to
 - reduce incidental complexities with machine learning
 - better enable teams across projects and workflows
- Machine learning operations (**MLOps**) should be **easier with ML platforms**
 - at all stages of a machine learning project's life cycle, from prototyping to production at scale,
 - as the number of models in production grows from one or a few to tens, hundreds, or thousands
- The platform should be
 - **designed to orchestrate machine learning workflow,**
 - **environment-agnostic (portable to multiple environments),**
 - **and work with different libraries and frameworks**
- "Data scientists only have to think about the where and when to deploy a model in a batch, not the how. The platform handles that."

Elijah Ben Izzy and Stefan Krawczyk in Deployment for Free;

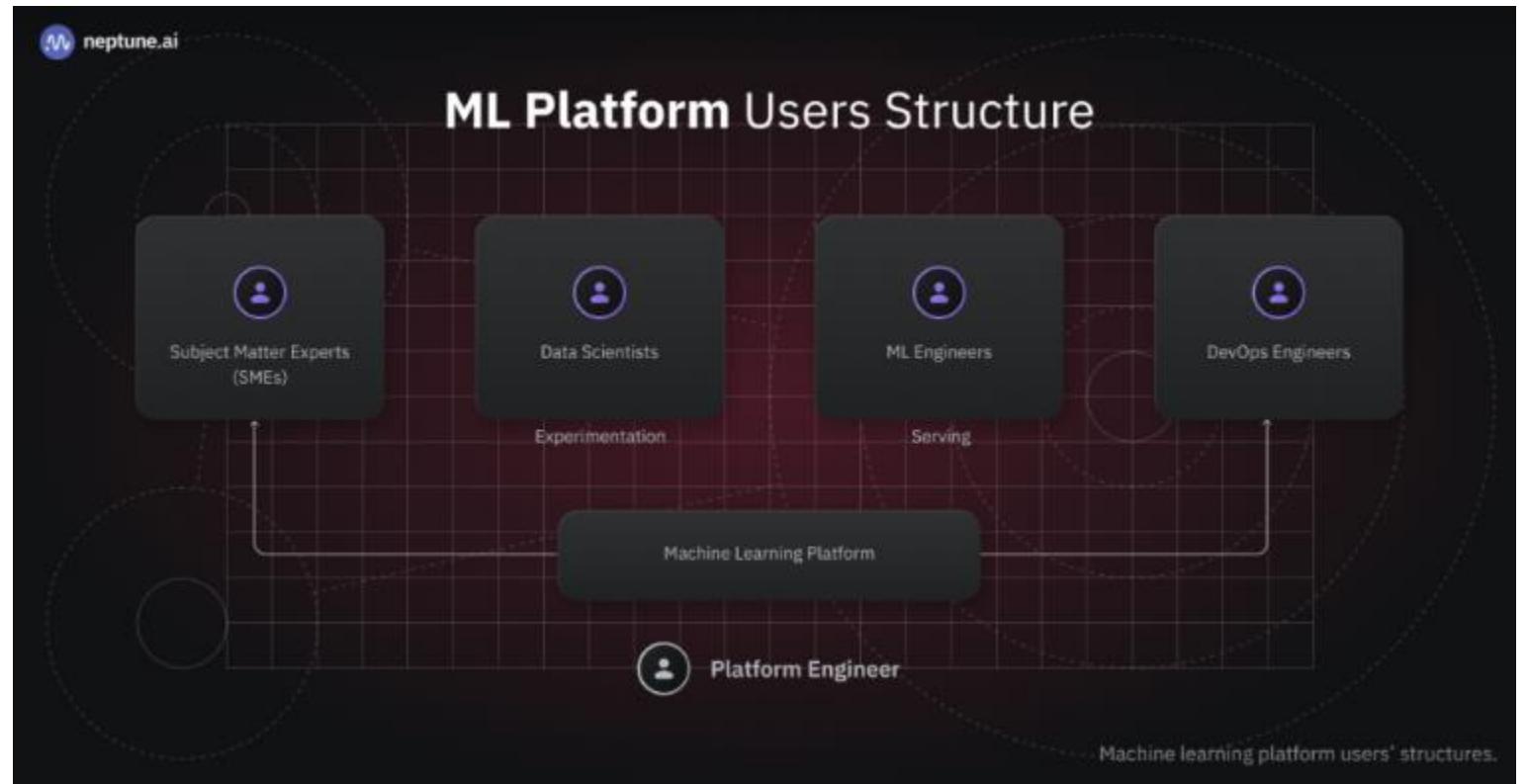
MLOps principles that ML platform should solve

- Understanding MLOps principles and how you can implement them can govern how you build your ML platform
 - The principles of MLOps can actively help you frame how you define the goals of your machine learning platform.
- **MLOps principles** that can help ML platforms solve different kinds of problems:
 - Reproducibility
 - Versioning
 - Automation
 - Monitoring
 - Testing
 - Collaboration
 - Scalability



Users of machine learning platforms

- Role of a **platform engineer**, or in most cases, an “**MLOps engineer**”
 - to practically architect and build solutions that make it easy for other users
 - to interact with the ML lifecycle
 - while proving appropriate abstractions from the core infrastructure



Platform Users: Subject Matter Experts (SMEs)

- Who they are?
- SMEs are the **non-developer experts** in the business problem
 - have critical roles to play across the entire ML lifecycle
 - Work with other users, to make sure
 - data reflects the business problem,
 - experimentation process is good enough for the business
 - and results reflect what would be valuable to the business
- Need to build interfaces into platforms for SMEs to:
 - Contribute to data labeling (if your data platform is not separate from the ML platform),
 - Perform model quality assurance for auditing and managing risks both in development and post-production,
 - Close feedback stages in production to make sure model performance metrics translate to real-world business value

Platform Users: ML Engineers and Data Scientists

- Who they are?
 - Depending on the **existing team structure and processes of the business**,
 - **ML engineers** may work **on delivering models to production**,
 - **Data scientists** may focus on **research and experimentation**
 - Some organizations hire either person to own the end-to-end ML project and not parts of it
- But the **lines between data scientists and ML engineers are blurring**
 - with the commoditization of model development processes and workflows!
- The following are some of the goals they'd like to achieve:
 - Frame business problem:
 - collaborate with subject matter experts to outline the business problem in such a way
 - that they can build a viable machine learning solution.
 - Model development:
 - access business data from upstream components,
 - work on the data (if needed),
 - run ML experiments (build, test, and strengthen models),
 - and then deploy the ML model
 - Productionalization:
 - often really subjective in teams because it's mostly the ML engineers that end up serving models

Platform Users: DevOps Engineers

- Who they are?
- Depending on the organization,
 - are either pure software engineers or simply tagged “DevOps engineers” (or IT engineers)
 - are responsible for CI/CD pipeline management across the entire organizational stack
 - are mostly responsible for operationalizing the organization’s software in production
- What do they want to accomplish?
 - Perform operational system development and testing
 - to assure the security, performance, and availability of ML models
 - as they integrate into the wider organizational stack

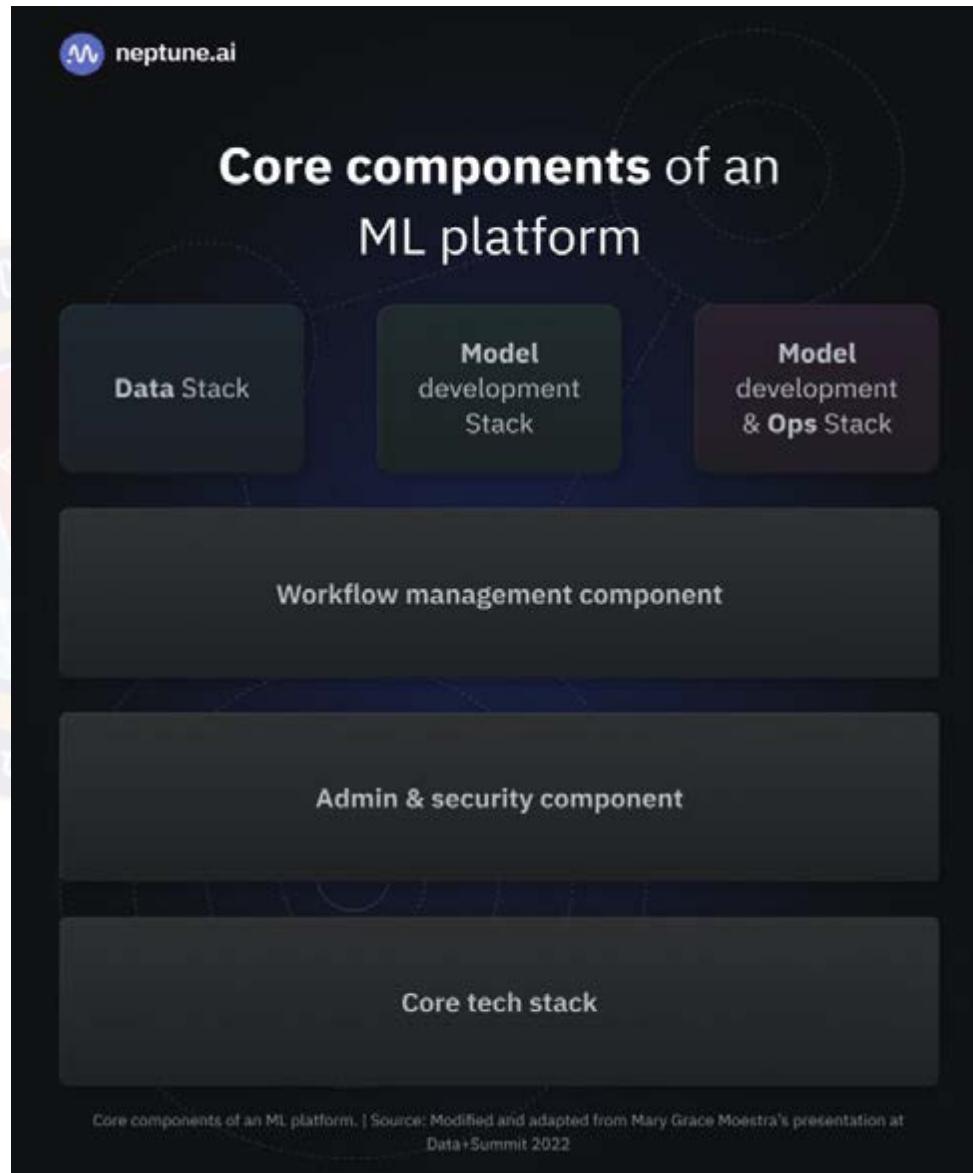
Platform Users: Others

- Data engineers
 - if the data platform is not particularly separate from the ML platform
- Analytics engineers and data analysts
 - if need to integrate third-party business intelligence tools and the data platform, is not separate



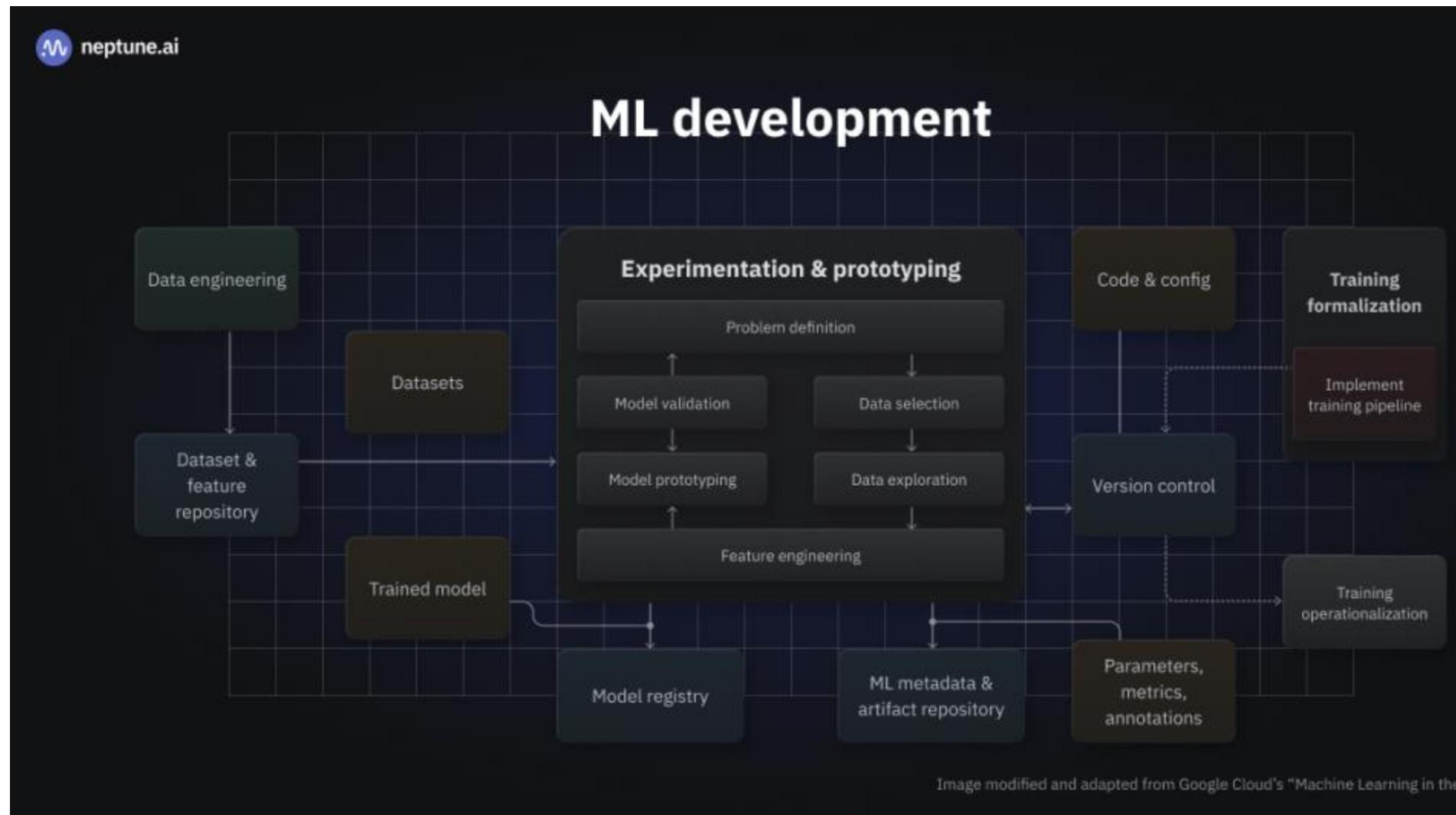
ML platform architecture

- The ML platform architecture serves as a blueprint for machine learning system
- The features of an ML platform and the core components that make up its architecture are:
 - Data stack and model development stack
 - Model deployment and operationalization stack
 - Workflow management component
 - Administrative and security component
 - Core technology stack



Data and model development stacks

- Main components:
- Data and feature store
- Experimentation component
- Model registry
- ML metadata and artifact repository

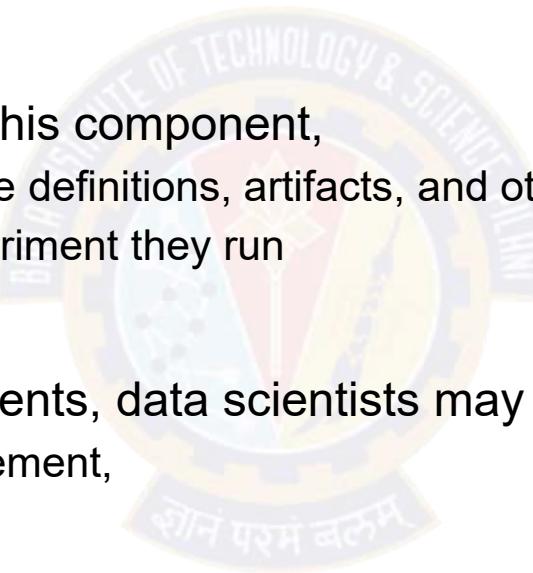


Data and feature store

- In a machine learning platform, feature stores (or repositories)
 - give data scientists a place to find and share the features they build from their datasets
 - ensures they use the same code to compute feature values for model training and inference to avoid training-serving skew
- Different teams may be involved in extracting features from different dataset sources
 - a centralized storage would ensure they could all use the same set of features to train models for different use cases
- The feature stores can be
 - offline (for finding features, training models, and batch inference services)
 - or online (for real-time model inference with low latency)
- The key benefit - decouples feature engineering from feature usage, allowing independent development and consumption of features
 - Features added to a feature store become immediately available for training and serving

Experimentation component

- Experiment tracking can help manage how an ML model changes over time
 - to meet data scientists' performance goals during training
- Data scientists develop models on this component,
 - which stores all parameters, feature definitions, artifacts, and other experiment-related information
 - that they care about for every experiment they run
- Based on the results of the experiments, data scientists may
 - decide to change the problem statement,
 - switch the ML task,
 - or use a different evaluation metric

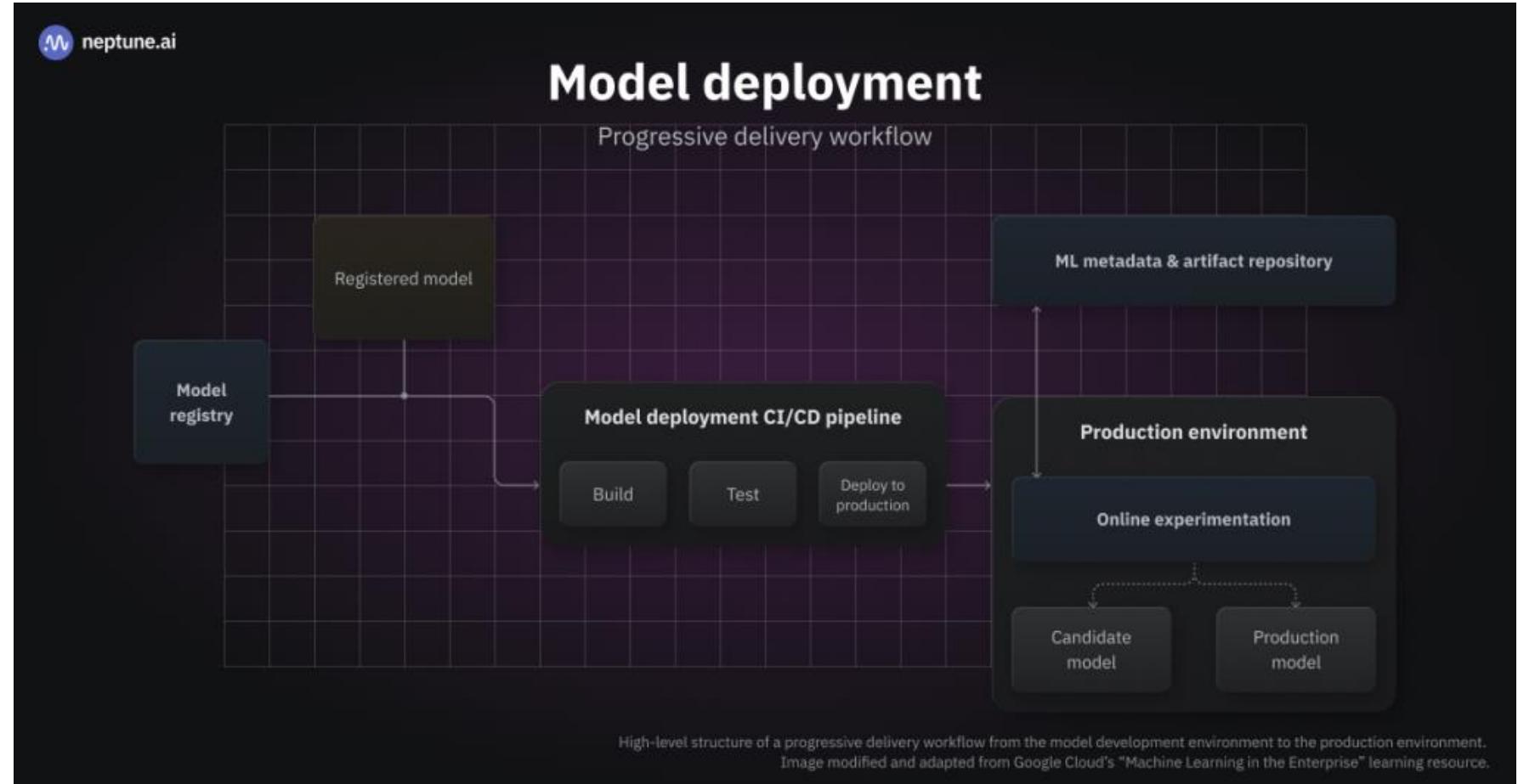


Model registry, ML metadata and artifact repository

- Model registry
 - helps to put some structure into the process of productionizing ML models
 - stores the validated training model and the metadata and artifacts that go with it
- Central repository stores and organizes models in a way that
 - makes it more efficient to organize models across the team,
 - making them easier to manage, deploy, and,
 - in most cases, avoid production errors (for example, putting the wrong model into production)
- ML metadata and artifact repository
 - Need the ML metadata and artifact repository to make it easier to compare model performance and test them in the production environment
 - A model can be tested against the production model, drawing from the ML metadata and artifact store to make those comparisons

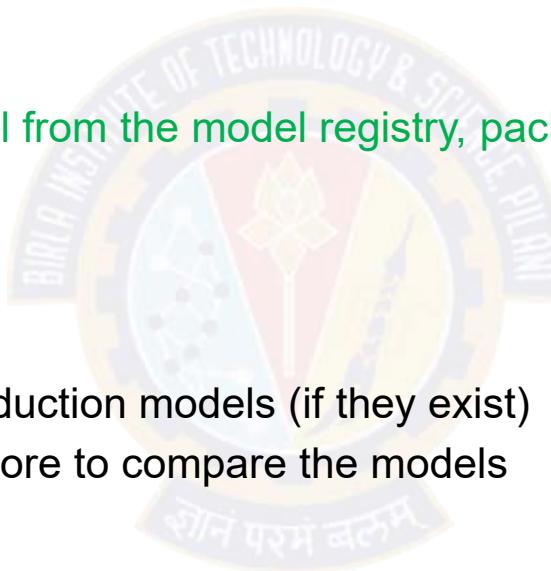
Model deployment and operationalization stack

- The main components:
- Production environment
- Model serving
- Monitoring and observability
- Responsible AI and explainability



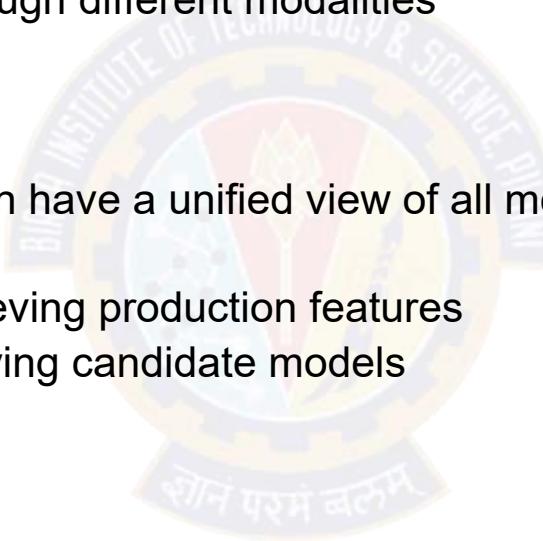
ML metadata and artifact repository

- Data scientists manually build and test models that needs to be deployed to the production environment
- In an ideal situation,
 - pipelines and orchestrators take a model from the model registry, package it, test it,
 - and then put it into production
- The **production environment component**
 - lets the model be tested against the production models (if they exist)
 - by using the ML metadata and artifact store to compare the models
- Could also decide to build configurations for deployment methods
 - like canary, shadow, and A/B deployment in the production environment



Model serving component

- When DSs (data scientists) or MLEs (machine learning engineers) deploy the models to their target environments as services
 - can serve predictions to consumers through different modalities
- The model serving component helps
 - organize the models in production so can have a unified view of all models
 - can successfully operationalize them
 - integrates with the feature store for retrieving production features
 - connects with the model registry for serving candidate models
- Popular model serving modalities:
 - Online inference
 - Streaming inference
 - Offline batch inference
 - Embedded inference

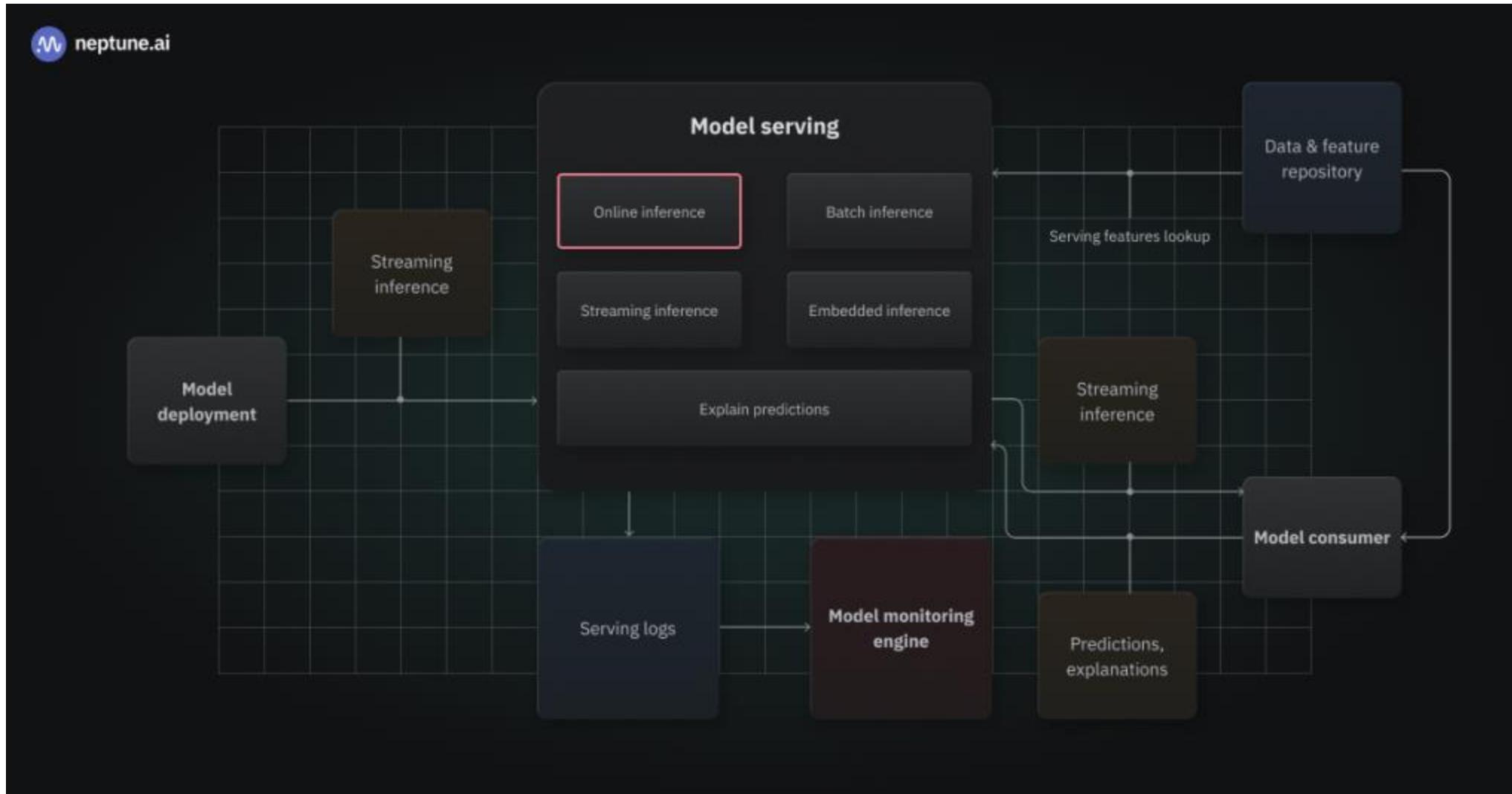


Model serving component(2)

Model serving modalities

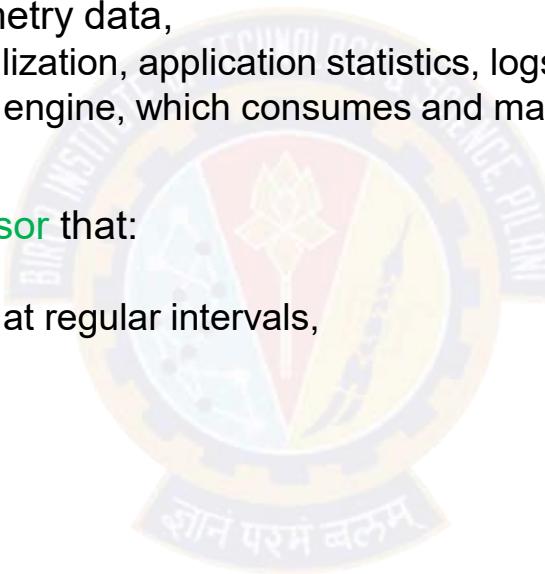
- Online inference
 - The ML service serves real-time predictions to clients as an API
 - (a function call, REST API, gRPC, or similar) for every request on demand
 - concern with this service would be scalability, but that's a typical operational challenge for software
- Streaming inference
 - The clients push the prediction request and input features into the feature store in real time
 - The service will consume the features in real time, generate predictions in near real-time,
 - such as in an event processing pipeline, and write the outputs to a prediction queue
 - The clients can read back predictions from the queue in real time and asynchronously.
- Offline batch inference
 - The client updates features in the feature store
 - An ML batch job runs periodically to perform inference
 - The job reads features, generates predictions, and writes them to a database
 - The client queries and reads the predictions from the database when needed
- Embedded inference
 - The ML service runs an embedded function that serves models on an edge device or embedded system

Types of Inferences



Monitoring component

- Implementing effective monitoring is key to successfully operating machine learning projects
- A monitoring agent regularly collects telemetry data,
 - such as audit trails, service resource utilization, application statistics, logs, errors, etc.
 - sends the data to the model monitoring engine, which consumes and manages it
- Inside the engine is a metrics data processor that:
 - Reads the telemetry data,
 - Calculates different operational metrics at regular intervals,
 - And stores them in a metrics database.
- The monitoring engine
 - has access to production data,
 - runs an ML metrics computer,
 - and stores the model performance metrics in the metrics database
- An analytics service provides reports and visualizations of the metrics data
 - When certain thresholds are passed in the computed metrics, an alerting service can send a message



Responsible AI and explainability component

- To fully trust ML systems, it's important to interpret these predictions
 - need to build platform to perform feature attribution for a given model prediction
 - these explanations show why the prediction was made
- Must implement this part together to make sure that the models and products meet the governance requirements, policies, and processes
- Since ML solutions also face threats from adversarial attacks that compromise the model and data used for training and inference
 - makes sense to inculcate a culture of security for ML assets too, and not just at the application layer (the administrative component)

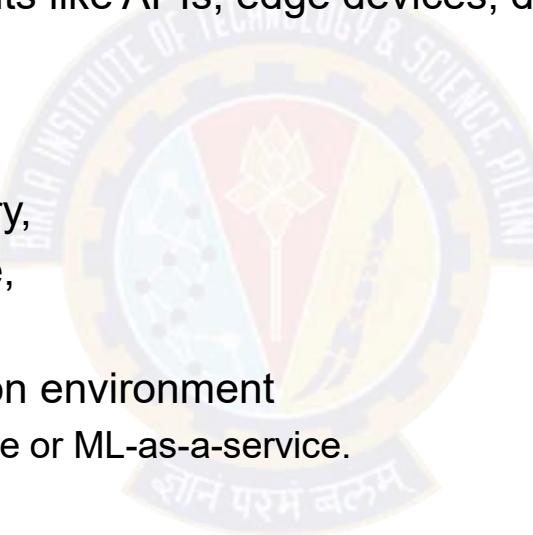
Workflow management component

- The main components here include:
 - Model deployment CI/CD pipeline
 - Training formalization (training pipeline)
 - Orchestrators
 - Test environment



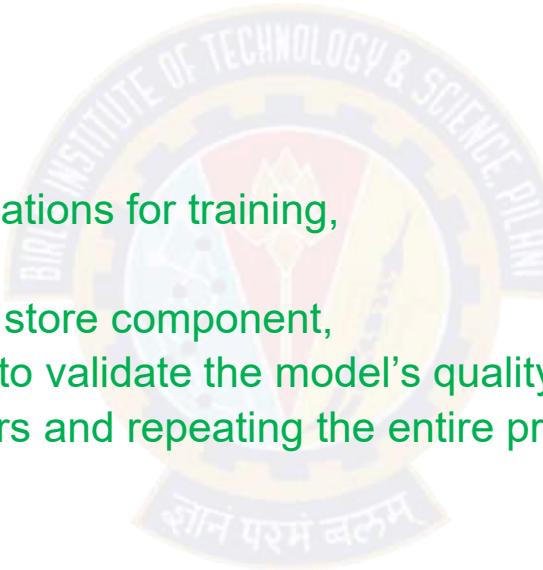
Model deployment CI/CD pipeline

- ML models that are used in production don't work as stand-alone software solutions!
 - Instead, they must be built into other software components to work as a whole
 - requires integration with components like APIs, edge devices, databases, microservices, etc.
- The CI/CD pipeline
 - retrieves the model from the registry,
 - packages it as executable software,
 - tests it for regression,
 - and then deploys it to the production environment
 - which could be embedded software or ML-as-a-service.
- The idea of this component is **automation**
 - goal is to **quickly rebuild pipeline assets ready for production** when you push new training code to the corresponding repository



Training formalization (training pipeline)

- Helps you manage repeatable ML training and testing workflows with little human intervention
 - The training pipeline functions to **automate** those workflows.
- From:
 - Collecting data from the feature store,
 - To setting some hyperparameter combinations for training,
 - Building and evaluating the model,
 - Retrieving the test data from the feature store component,
 - Testing the model and reviewing results to validate the model's quality,
 - If needed, updating the model parameters and repeating the entire process.
- The pipelines primarily use schedulers
 - would help manage the training lifecycle through a DAG (directed acyclic graph)
 - makes the experimentation process traceable and reproducible,
 - provided the other components discussed earlier have been implemented alongside it



Orchestrators and Test environment

- Orchestrators
- The orchestrators coordinate
 - how ML tasks run
 - where they get the resources to run their jobs
- Orchestrators are concerned with lower-level abstractions
 - like machines, instances, clusters, service-level grouping, replication, and so on
- Integral to managing the regular workflows data scientists run and how the tasks in those workflows communicate with the ML platform
- Test environment
 - The test environment gives data scientists the infrastructure and tools
 - they need to test their models against reference or production data,
 - usually at the sub-class level,
 - to see how they might work in the real world before moving them to production
- Can have different test cases for ML models and pipeline

Core technology stack

- The main components of this stack include:
 - Programming Language
 - Collaboration
 - Libraries and Frameworks
 - Infrastructure and Compute



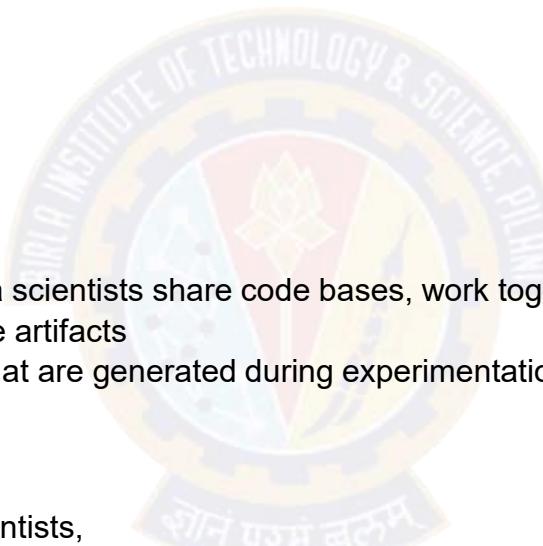
The programming language

- The programming language is another crucial component of the ML platform
 - the language would you use to develop the ML platform
 - the language your users would perform ML development with
- Python
 - The most popular language with strong community support
 - that would likely ensure you are making your users' workflow efficient would likely be
- But then again, understand their existing stack and skillset, so you know how to complement or migrate it.

Collaboration

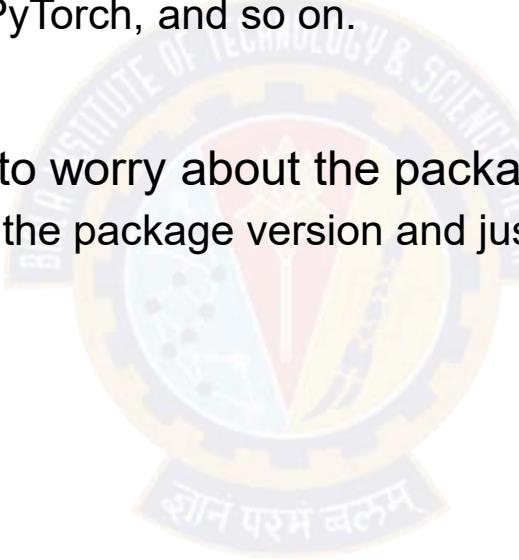
One of the most important principles of MLOps that should be integrated into any platform is collaboration

- Component has to do with how all the platform users can collaborate with each other and across other teams.
- The main components here include:
 - Source control repository
 - Notebooks and IDEs
 - Third-party tools and integrations
- Source code repository
 - During experimentation, this component lets your data scientists share code bases, work together, peer review, merge, and make changes
 - A source code repository is used to keep track of code artifacts
 - like notebooks, scripts, tests, and configuration files that are generated during experimentation
- Notebooks and IDEs
 - The notebook is the experimentation hub for data scientists,
 - there needs to be an agreement on what tools will be ideal for the team long-term—components that will be around in 5–10 years
- Third-party tools and integrations
 - Sometimes, data team might need to integrate with some external tool that wasn't built with the platform, perhaps due to occasional needs
 - For example, the data team might want to use an external BI tool to make reports



Libraries and frameworks

- This component lets you natively integrate machine learning libraries and frameworks
 - That your users mostly leverage into the platform
 - Some examples are TensorFlow, PyTorch, and so on.
- You never want your data scientist to worry about the packages they use to develop
 - should be able to know that this is the package version and just import it on their workspace



Infrastructure and compute

arguably the most important layer to figure out, along with the data component

- ML platform will run on this layer!
 - With the different moving parts and components you have seen, it can be quite tricky to tame and manage this layer of the platform
- The infrastructure layer allows for scalability at both the data storage level and the compute level,
 - which is where models, pipelines, and applications are run
- The considerations include:
 - Are your existing tools and services running on the Cloud, on-prem, or a hybrid?
 - Are the infrastructure services (like storage, databases, for example) open source, running on-prem, or running as managed services?
 - These considerations would help you understand how to approach designing platform.
- But ideally, want an infrastructure layer that reduces the friction between moving across the different stacks,
 - from data to model development, deployment, and operationalization
- In most cases, that'd arguably be the cloud, but of course, not every use case can leverage cloud infrastructure,
 - so keep that principle of “less friction” in mind regardless of where infrastructure is located.



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Considerations when deciding on the scope of the ML platform

Pravin Y Pawar

Adapted from
"Building a Machine Learning Platform [Definitive Guide]"
by Stephen Oladele

Enterprise machine learning platform vs startup ML platform

- Especially when meeting data scientists' needs in an enterprise setting
 - Two themes for mastering MLOps — Transparency and pure efficiency
 - because it's mostly speed and agility that matter
- In most startups, data scientists can get away with deploying, testing, and monitoring models ad hoc,
 - especially with a handful of models
- However, in the enterprise, they would waste enormous amounts of time reinventing the wheel with every ML model
 - which may never result in scalable ML processes for the organization

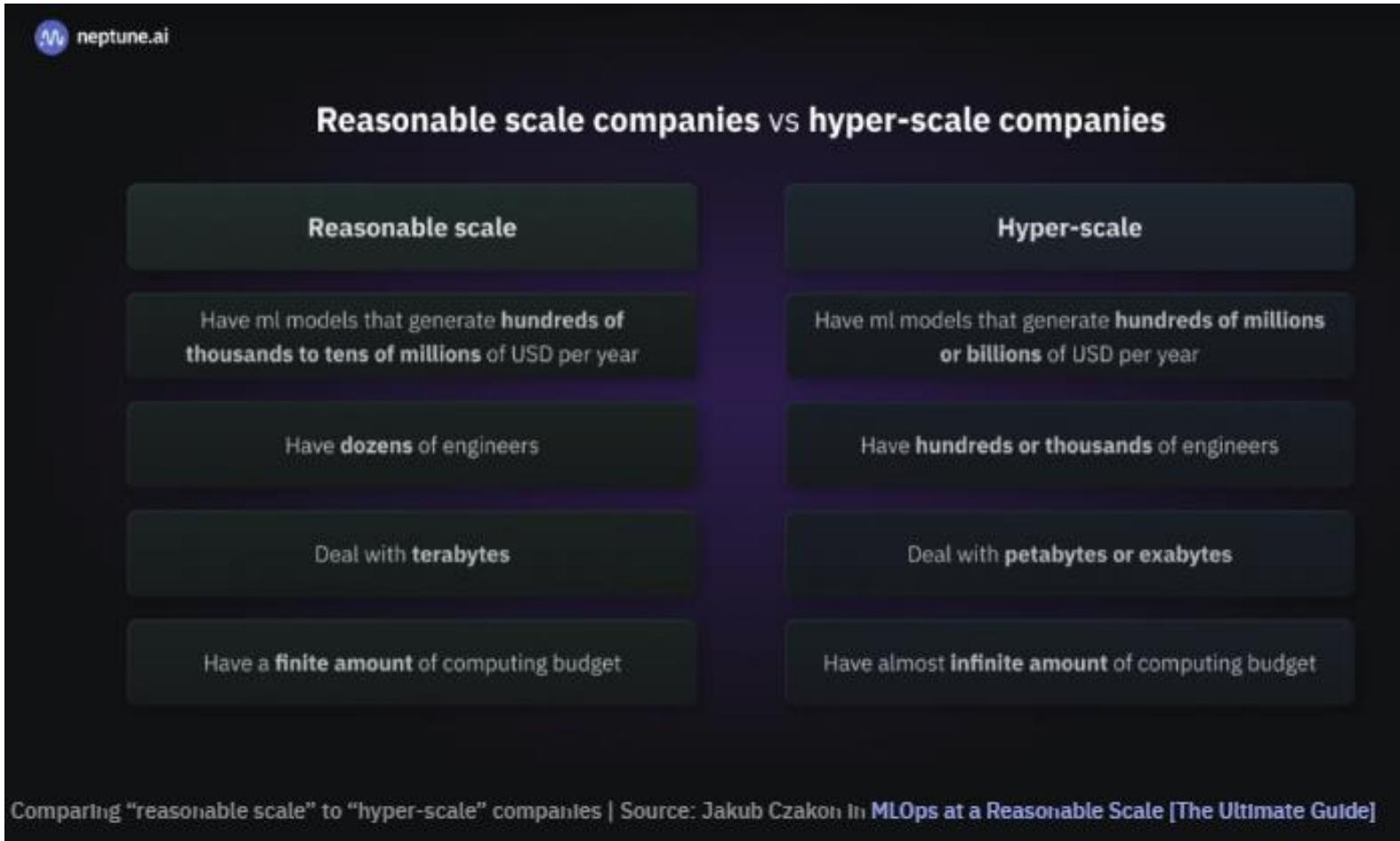
Differences in ML platforms at start-ups compared to enterprises:

	Startups	Enterprises
Number of ML services	The platform may have a handful of models to support development and production based on a small number of use cases.	The platform could support thousands to hundreds of models in research and production.
Data volume	Often in the order of gigabytes or terabytes per day.	On average, most enterprises, especially those with digital products, often deal with data volumes in the order of petabytes per day.
Business impact	ML services may have ROIs in the hundreds of thousands to tens of millions of US dollars per year.	Enterprise ML services may have a yearly ROI in the hundreds of millions or billions of dollars.
Infrastructure	The number of use cases supported determines the requirements and how specialized those use cases are.	Due to the scale of models, research, and use cases, the infrastructure needs are often on the high side of the spectrum.
Tooling	Evaluating tools for ML platforms is frequently simple, with considerations for cloud, open source, or managed services.	Evaluating tools for ML platforms is based on strict criteria involving several stakeholders.
Team sizes	The platform team is usually a handful of engineers or tens of engineers supporting data scientists.	The platform is typically built by hundreds or thousands of engineers who support many data scientists and researchers from various teams.
MLOps blueprint	The MLOps maturity level of the platforms is usually at level 0 due to many ad-hoc processes.	Due to the number of engineers and the overhead involved in managing the systems, the maturity levels are usually higher, around level 1 or 2.

Reasonable scale ML platform

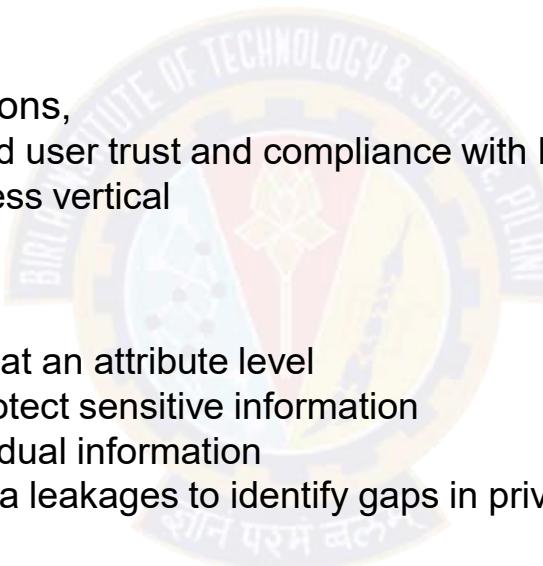
- In 2021, Jacopo Tagliabue coined the term “reasonable scale,” which refers to companies that:
 - Have ML models that generate hundreds of thousands to tens of millions of US dollars per year (rather than hundreds of millions or billions)
 - Have dozens of engineers (rather than hundreds or thousands)
 - Deal with data in the order of gigabytes and terabytes (rather than petabytes or exabytes)
 - Have a finite amount of computing budget
 - A reasonable scale ML platform helps achieve those requirements listed above
- Compared to a reasonable scale platform,
 - a “hyper-scale” platform should support hundreds to thousands of models in production
 - that likely generate millions to billions of dollars in revenue per year
 - also have hundreds to thousands of users, support multiple use cases, and handle petabyte-scale data

Reasonable scale vs Hyper-scale companies



Data-sensitive ML platform

- Data privacy is important when building an ML platform,
 - but how should sensitive data be kept from being accidentally or maliciously leaked (adversarial attacks)
 - when data scientists use it for training or inference?
- If organizational use cases require privacy protections,
 - need to build your ML platform with customer and user trust and compliance with laws, regulations, and standards in mind
 - includes compliance with the data laws of business vertical
- Consider building the following into ML platform:
 - Access controls to data components specifically at an attribute level
 - Use or build tools to anonymize datasets and protect sensitive information
 - Use or build data aggregation tools to hide individual information
 - Add tools for modeling threats and analyzing data leakages to identify gaps in privacy implementation and fix them
- Useful tools and frameworks to build data-sensitive components as part of ML platform:
 - Tensorflow Privacy
 - ML Privacy Meter
 - PySyft
 - CrypTFlow



Human-in-the-loop ML platforms

- Human oversight is crucial for every ML platform!
 - As much as you'd like to automate every component of the platform to make your life and that of your data scientists easier,
 - it's most likely not going to be possible
- For example,
 - finding “unknown unknowns” of data quality problems on platforms that use new data to trigger retraining feedback loops might be hard to automate
 - And when the platform automates the entire process, it'll likely produce and deploy a bad-quality model
- Your components need to have interfaces that relevant stakeholders and subject matter experts can interact with to evaluate data and model quality
 - need to design the platform with them to understand
 - the components that require manual evaluation
 - those that can be solved with automated monitoring

Building an ML platform for special industry verticals

- Across different industries and business verticals, the use cases and ML product goals will differ
 - can be difficult to build a platform that caters to most use cases
 - one of the reasons why an all-size-fits-all ML platform may not work for most teams!
- Need a platform built on components that can be flexible to handle different objects and are reusable across use cases.
- The major considerations to make when planning an ML platform across special industry verticals include:
 - Data type:
 - For the different types of use cases team works on, what's the most prevalent data type, and can ML platform be flexible enough to handle them?
 - Model type:
 - How would you build your platform to allocate development and production resources effectively?
 - Legal requirements:
 - What are the legal and governance requirements the platform needs to comply with, from the data component to the predictions that get to the end users?
 - Team structure and collaboration:
 - How is the team working on this use case structured, and what stakeholders would be actively involved in the projects the platform needs to serve?

End to end vs canonical stack ML platform

- You can
 - buy an end-to-end ML platform that would seem to solve all your platform needs
 - or build a canonical stack of point solutions based on each component
- Ultimately, deciding whether to get one platform or mix and match different solutions will be
 - very context-dependent
 - and dependent on understanding where your business is as well as your users
- There are certainly pros and cons to both approaches!

Closed vs open end-to-end ML platform

- Closed platforms will generally provide an ecosystem
 - that would potentially allow for seamless integrations across entire stack
 - but would likely confine to that ecosystem in what's known as a "lock-in."
- With open platforms,
 - can easily swap components and leverage open-standard tools
 - can provide flexibility for team and get them to adopt and use the tool without too much overhead
- For example,
 - with Kubeflow, an open-source ML platform, can swap out components easily,
 - whereas that's not the case with a closed solution like DataRobot (or Algorithmia)
- When it comes to a managed or open source ML platform,
 - it all depends on the existing infrastructure as well as other build vs. buy decisions made
 - If organization runs its workloads on AWS, it might be worth it to leverage AWS SageMaker.



Thank You!

In our next session:



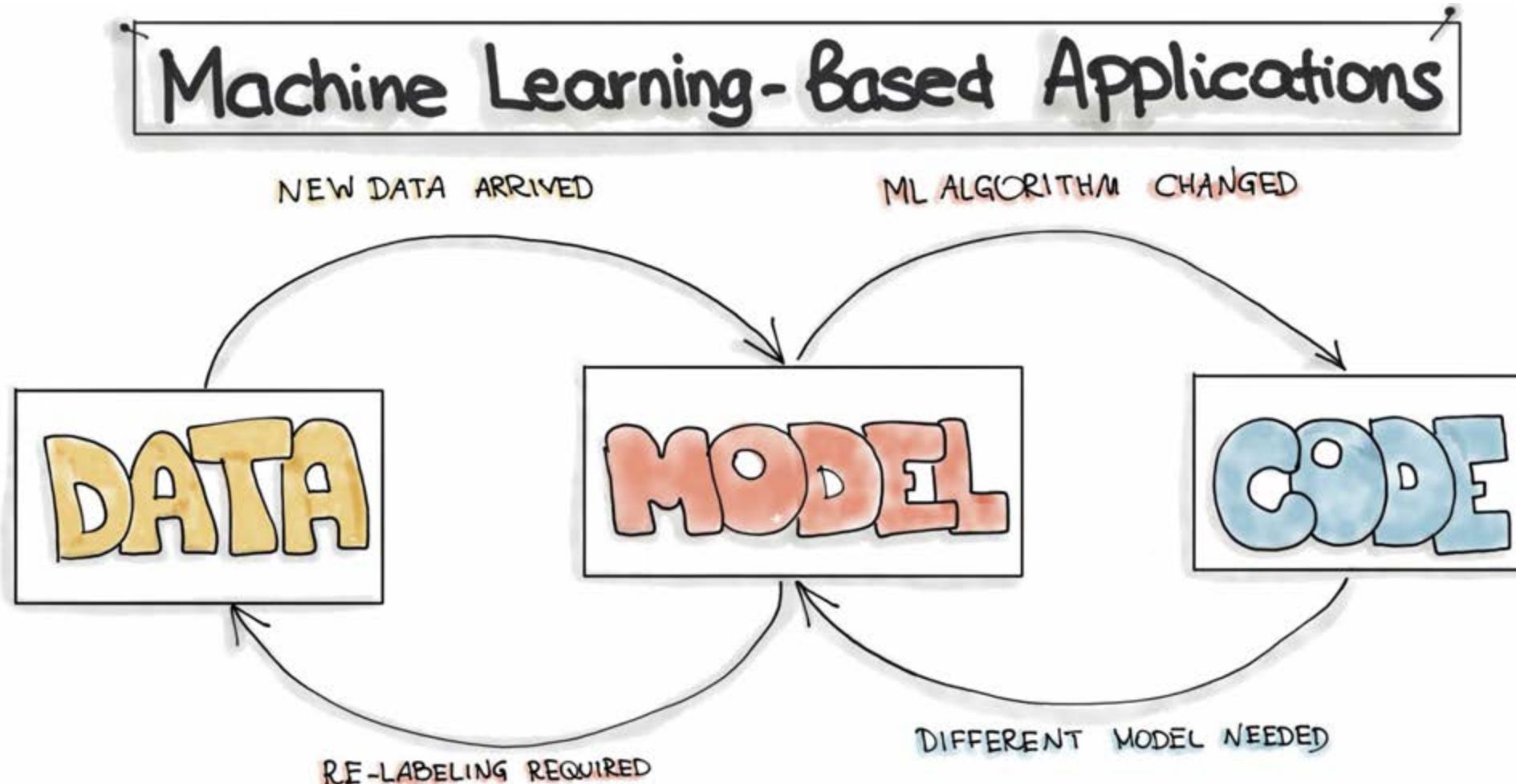
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Tools Landscape

Pravin Y Pawar

ML Based Applications

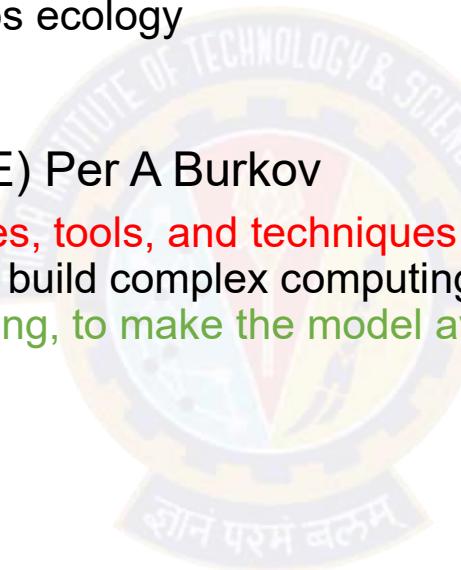
Three essential components - Data, ML Model, and Code



MLOps / ML Engineering

Two Definitions

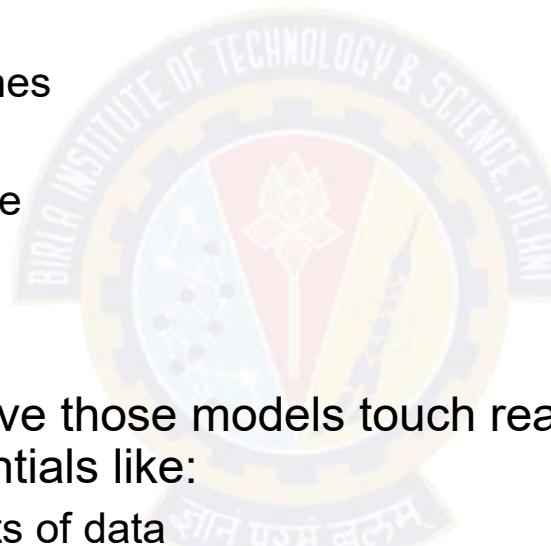
- MLOps SIG version
 - The extension of the DevOps methodology to include Machine Learning and Data Science assets as first-class citizens within the DevOps ecology
- Machine Learning Engineering (MLE) Per A Burkov
 - MLE is the use of **scientific principles, tools, and techniques** of **machine learning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



Moving to Production

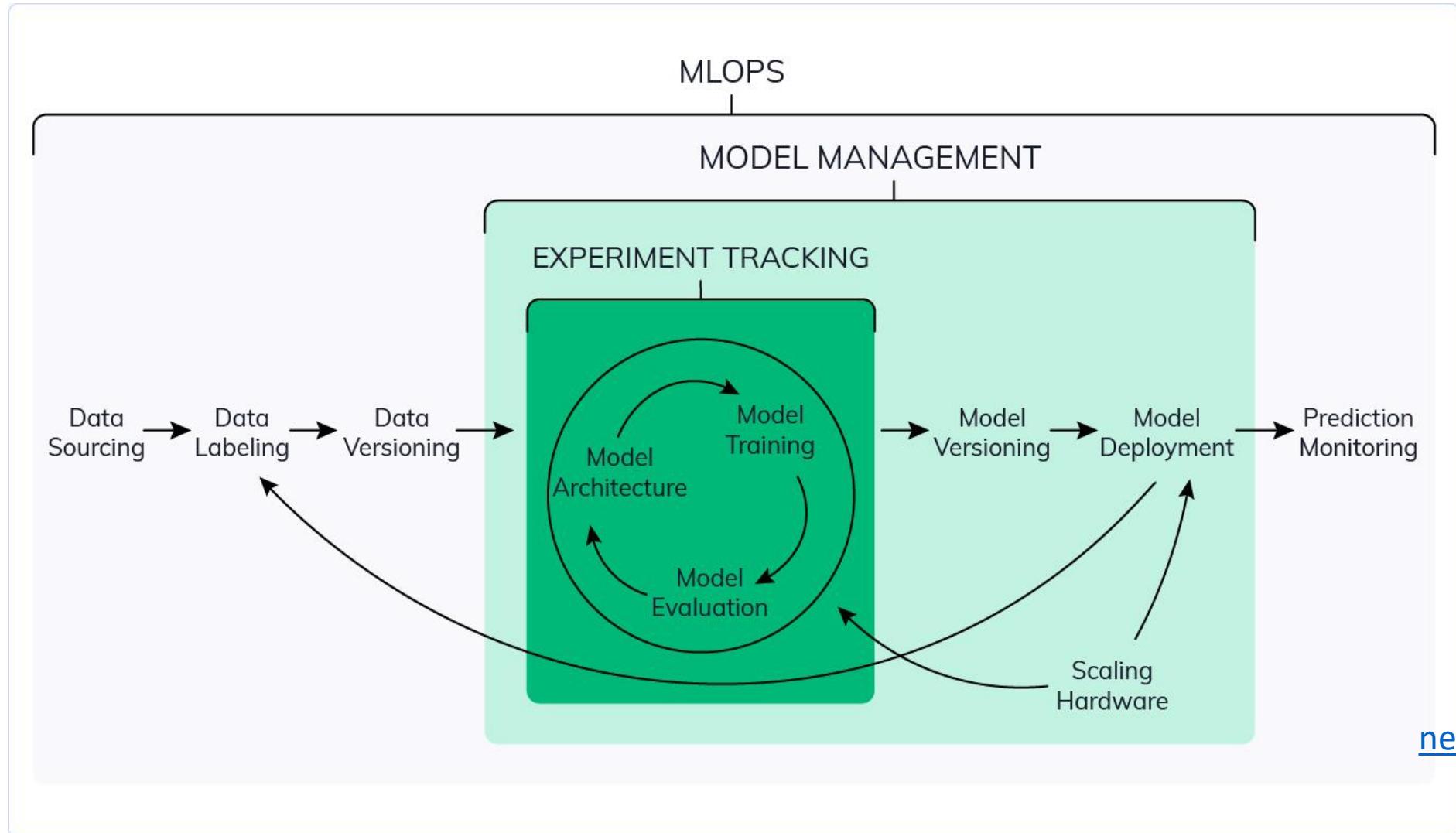
Problems (and requirements)

- Many pain points that both legacy companies (and many tech companies doing commercial ML) have regarding:
 - Moving to the cloud
 - Creating and managing ML pipelines
 - Scaling
 - Dealing with sensitive data at scale
 - Etc.
- Want to be serious and actually have those models touch real-life business problems and real people, have to deal with the essentials like:
 - acquiring & cleaning large amounts of data
 - setting up tracking and versioning for experiments and model training runs
 - setting up the deployment and monitoring pipelines for the models that do get to production
 - need to find a way to scale ML operations to the needs of the business and/or users of ML models

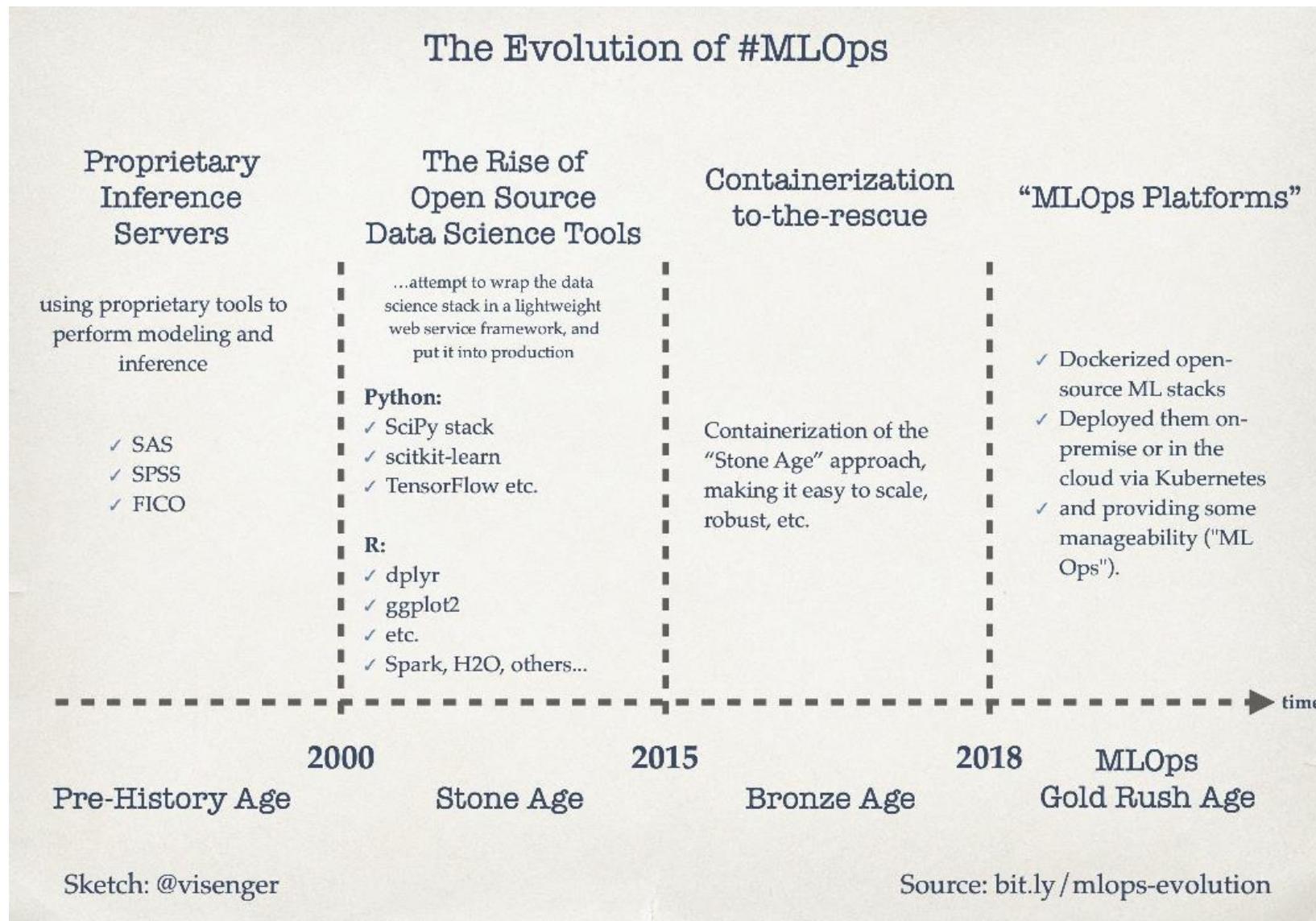


MLOps (Revisited)

Key phases of MLOps



The Evolution of the MLOps



MLOps Tools Landscape

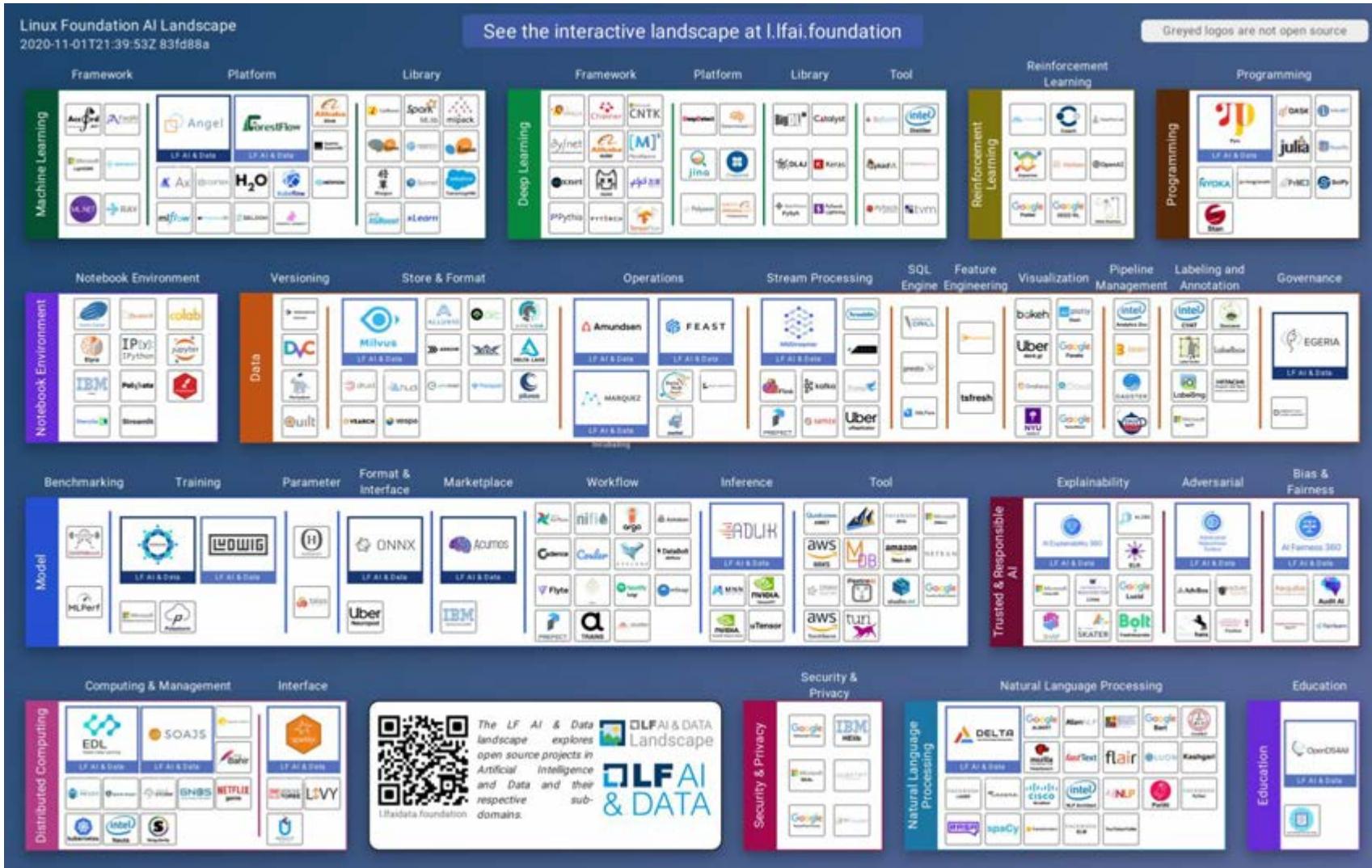
Difficult to find the best MLOps tools for your use case

Development IDE	Code Versioning	Data Labelling	Data Exploration and Visualization	Feature Engineering	Feature Store	Model Training	Building POC application	Model Debugging and Visualization	Model Tuning and HPO	Experiment Tracking	Model Packaging	Model Serving	Workflow Orchestration	Data Versioning	Model Registry	Model Monitoring
jupyter	git	labelbox	radar	mlflow	airflow	TensorBoard	redash	wave	target	wave	dagster	serve	kedro	wave	wave	arize
drift	airflow	dagster	aws sagemaker	mlflow	airflow	redash	jupyter	airflow	hyperfine	yellow dots	brain	brain	black diamond	mlflow	whylabs	
PC	git	labelbox	crown	datacube	airflow	hexagon	shiny	grey box	circle	red arrow	airflow	kedro	pachyderm	yellow dots	mlflow	

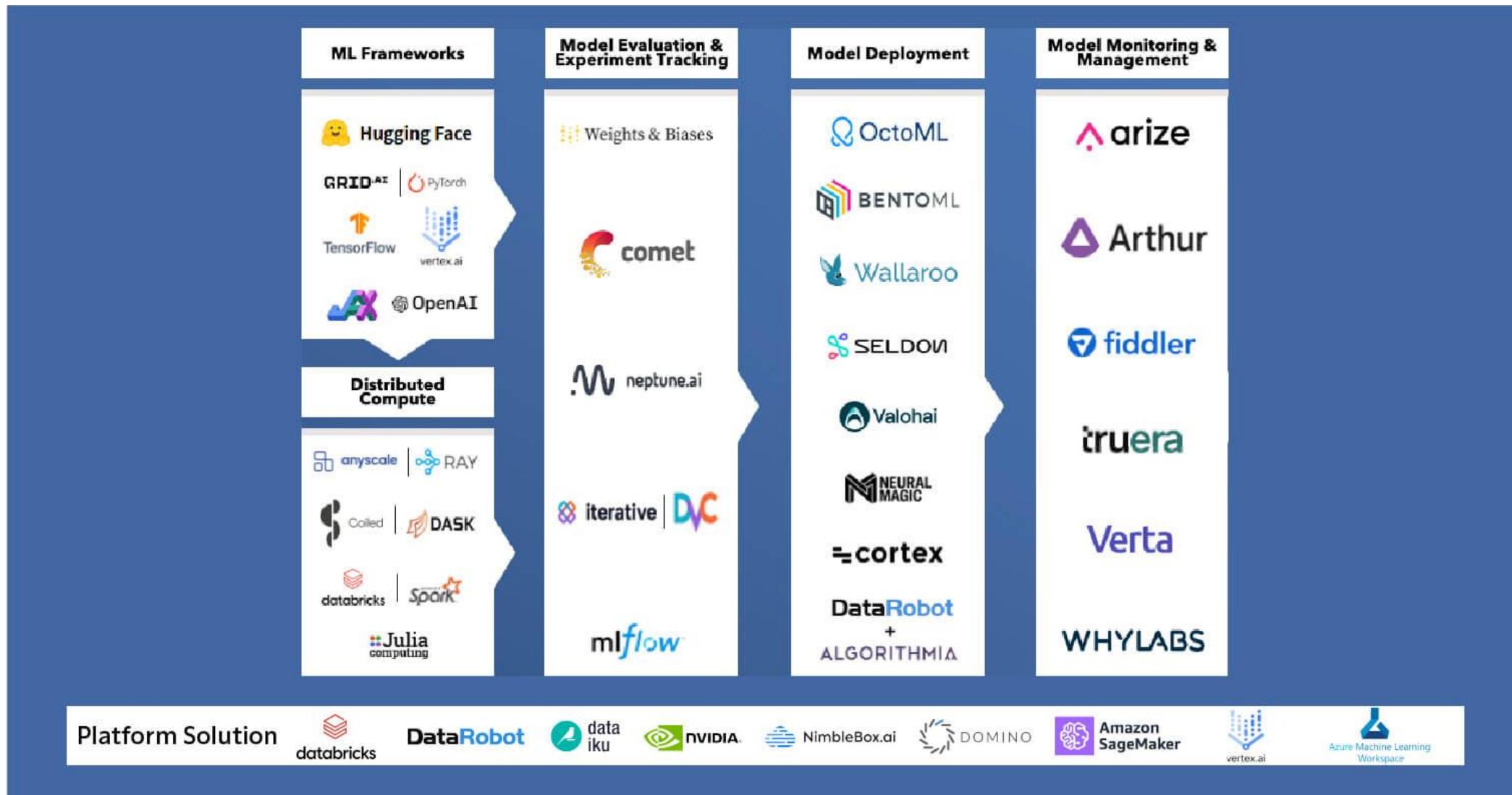
[neptune](#)

MLOps Tools Landscape

More Complicated!



Another view



MLOps Tools Broad Categories



Two MLOps solutions categories

End-to-end platform

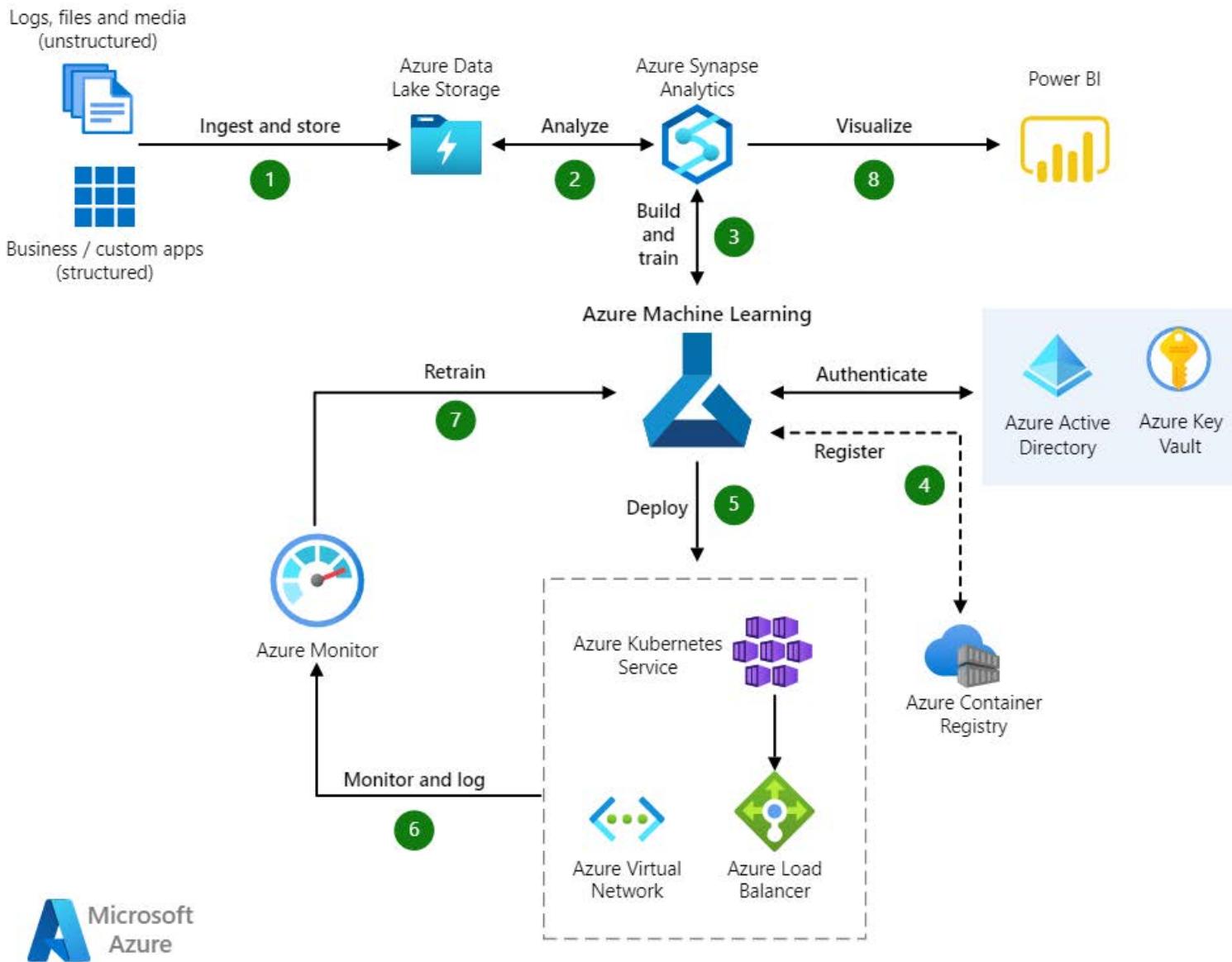
- Fully managed services that provide developers and data scientists
 - with the ability to build, train, and deploy ML models quickly.
- The top commercial solutions are:
 - Amazon Sagemaker, a suite of tools to build, train, deploy, and monitor machine learning models
 - Microsoft Azure MLOps suite:
 - Azure Machine Learning to build, train, and validate reproducible ML pipelines
 - Azure Pipelines to automate ML deployments
 - Azure Monitor to track and analyze metrics
 - Azure Kubernetes Services and other additional tools.
 - Google Cloud MLOps suite:
 - Dataflow to extract, validate, and transform data as well as to evaluate models
 - AI Platform Notebook to develop and train models
 - Cloud Build to build and test machine learning pipelines
 - TFX to deploy ML pipelines
 - Kubeflow Pipelines to arrange ML deployments on top of Google Kubernetes Engine (GKE)

Two MLOps solutions categories

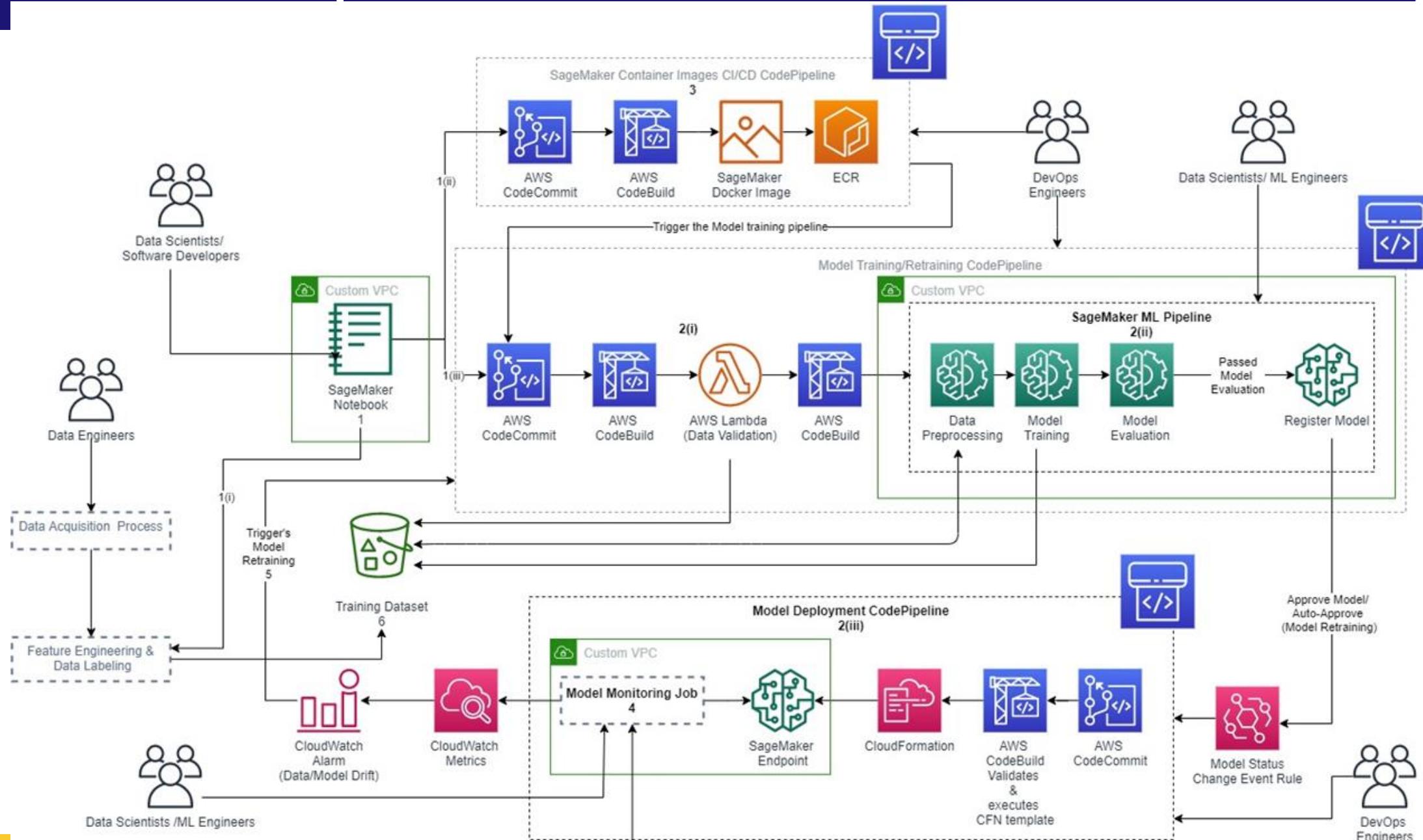
Custom-built MLOps solution (the ecosystem of tools)

- End-to-end solutions are great, but one can also build own solution with favorite tools,
 - by dividing MLOps pipeline into multiple microservices
- Can help to avoid a single point of failure (SPOF), and make pipeline
 - which are robust, easier to audit, debug, and more customizable
- With a microservices provider is having problems, can easily plug in a new one
 - ensure that each service is interconnected instead of embedded together
 - can have separate tools for model management and experiment tracking
- Many MLOps tools available, top picks are:
 - Project Jupyter
 - Airflow
 - Kubeflow
 - MLflow
 - neptune.ai

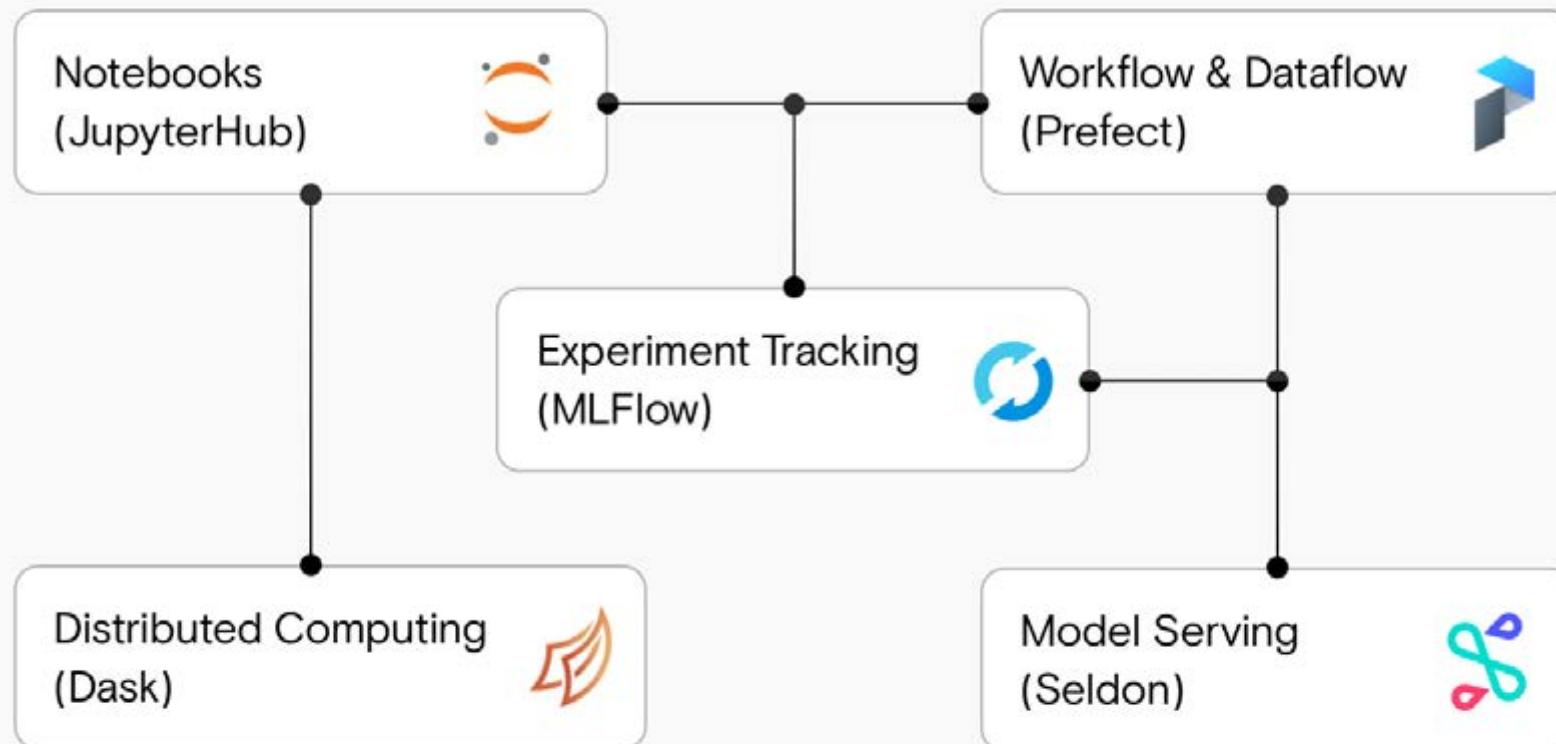
Azure Machine Learning architecture



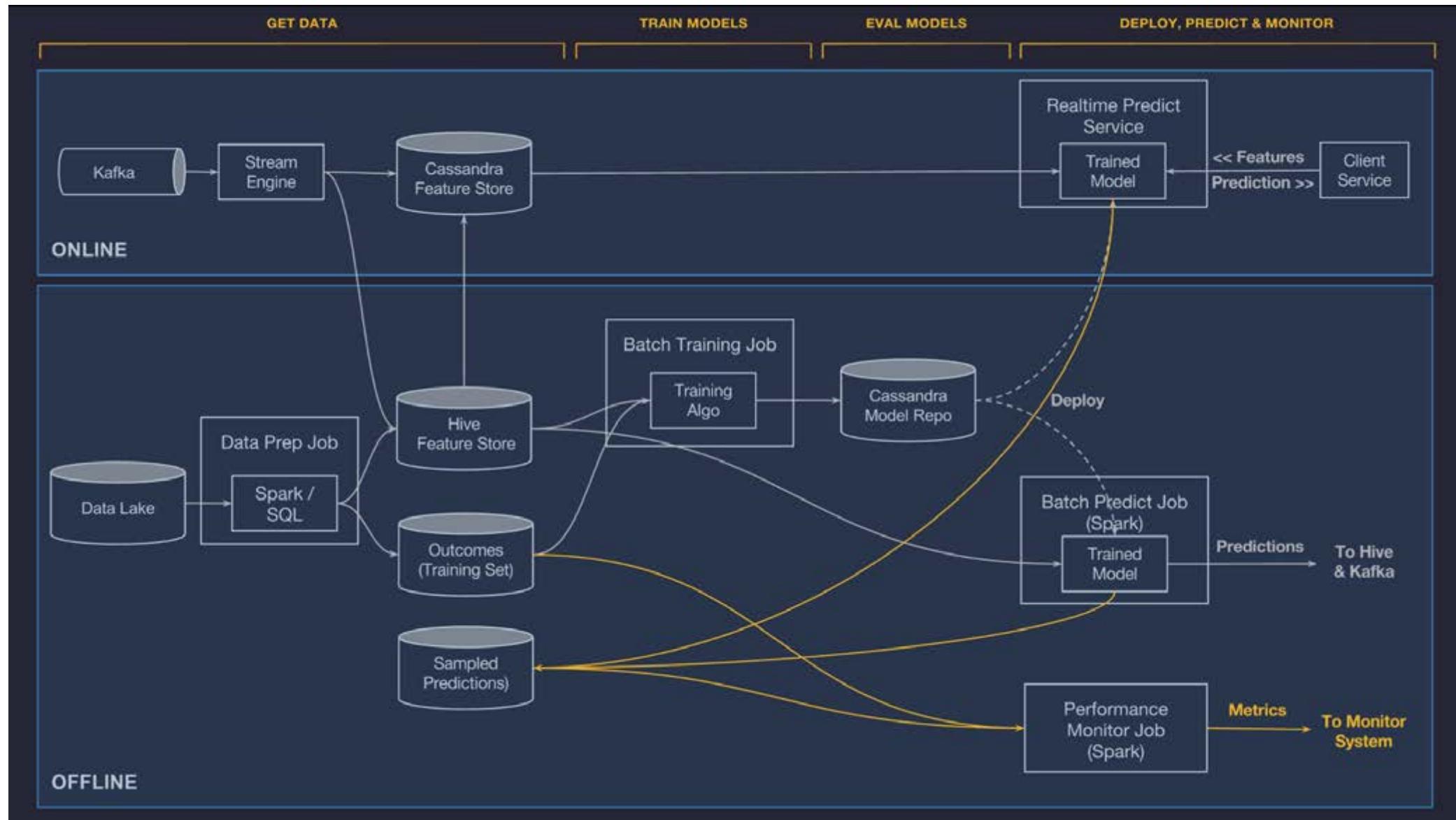
AWS MLOps Reference Architecture



Open MLOps

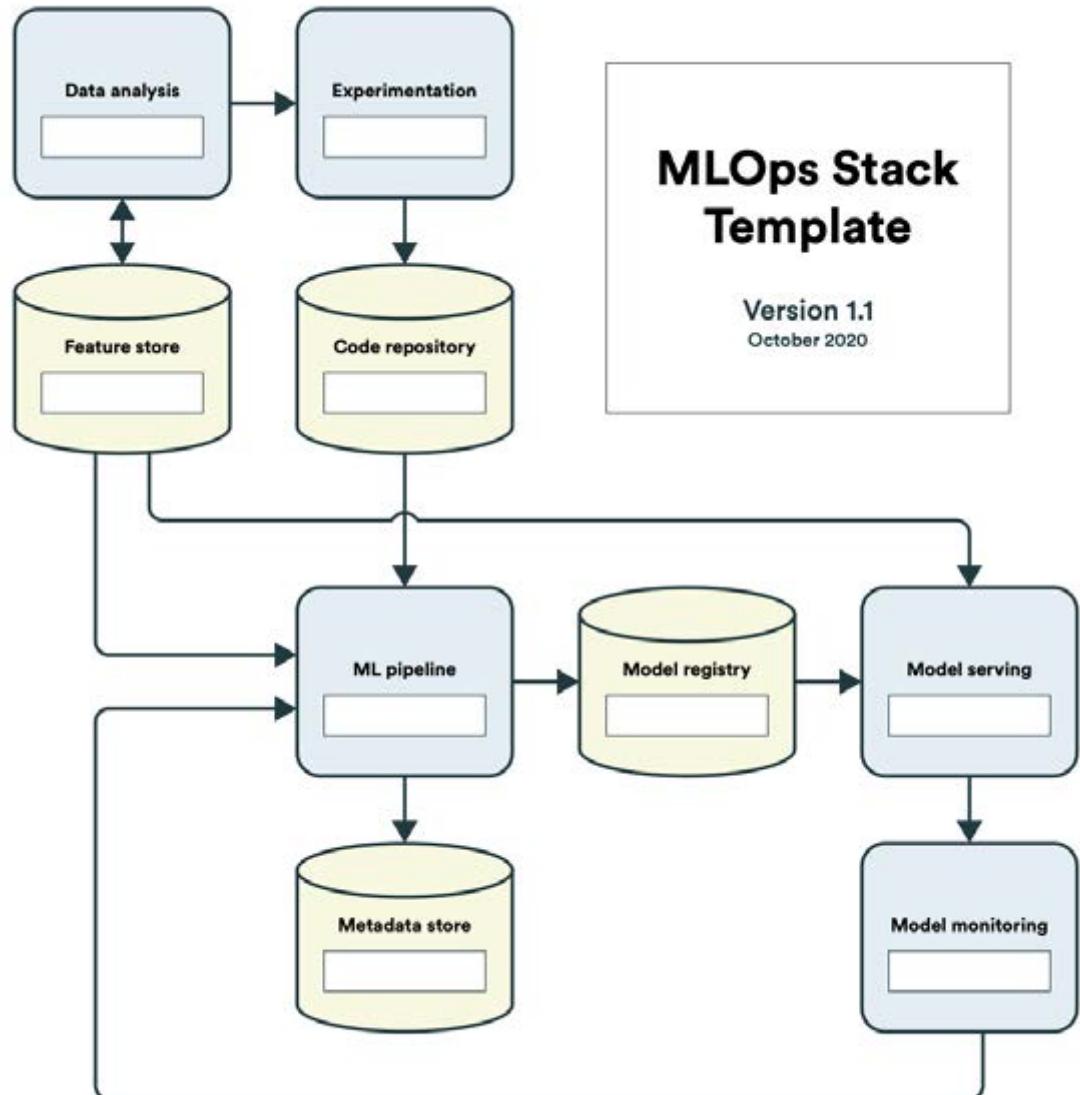


Uber's Michelangelo Platform



Uber engg

MLOps Stack Template



Component	Requirements	Tooling
Data analysis		
Experimentation		
Feature store		
Code repository		
ML pipeline		
Metadata store		
Model registry		
Model serving		
Model monitoring		



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Experiment Tracking

Pravin Y Pawar

Adapted from
[ML Experiment Tracking: What It Is, Why It Matters, and How to Implement It](#)
by Jakub Czakon

a story that have heard too many times

"... 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 lot of experiments

- And the truth is, when develop ML models, will run **a lot of experiments**
- Those experiments may:
 - use different models and model hyper parameters
 - use different training or evaluation data,
 - run different code (including this small change that you wanted to test quickly)
 - run the same code in a different environment (not knowing which PyTorch or Tensorflow version was installed)
 - And as a result, they can produce completely different evaluation metrics.
- Keeping track of all that information can very quickly become really hard
- Especially
 - if want to organize and compare those experiments
 - feel confident that you know which setup produced the best result
- This is where **ML experiment tracking** comes in!

ML experiment tracking

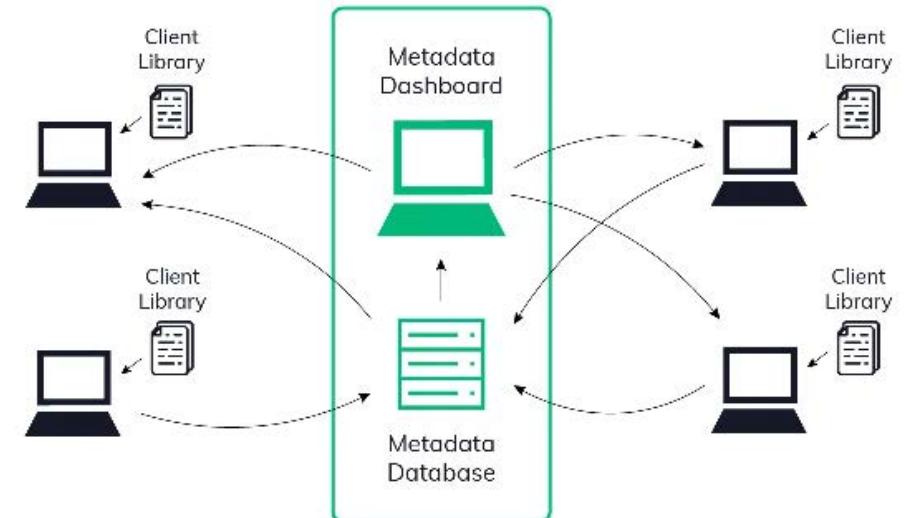
What?

- Experiment tracking is the process of saving all experiment related information that you care about for every experiment you run
- This “metadata you care about” will strongly depend on your project, but it may include:
 - Scripts used for running the experiment
 - Environment configuration files
 - Versions of the data used for training and evaluation
 - Parameter configurations
 - Evaluation metrics
 - Model weights
 - Performance visualizations (confusion matrix, ROC curve)
 - Example predictions on the validation set (common in computer vision)
- Of course, you want to have this information available after the experiment has finished,
 - but ideally, you’d like to see some of it as your experiment is running as well.

ML experiment tracking

Components

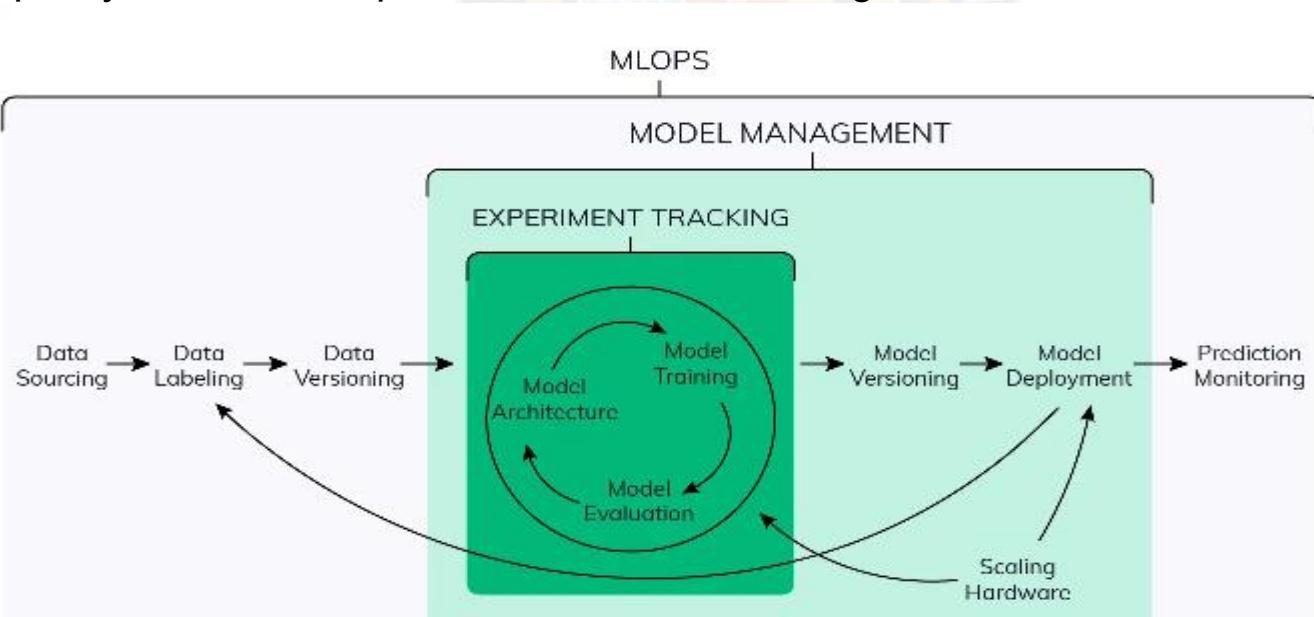
- To do experiment tracking properly, need some sort of a system that deals with all this metadata
- Typically, such a system will have 3 components:
 - Experiment database: A place where experiment metadata is stored and can be logged and queried
 - Experiment dashboard: A visual interface to your experiment database - A place where can see experiment metadata
 - Client library: Which gives methods for logging and querying data from the experiment database
- Of course, you can implement each component in many different ways, but the general picture will be very similar



Experiment tracking vs ML model management vs MLOps

isn't experiment tracking like MLOps or something?

- Experiment tracking (also referred to as experiment management) is a part of MLOps:
 - a larger ecosystem of tools and methodologies that deals with the operationalization of machine learning
- MLOps deals with every part of ML project lifecycle
 - from developing models by scheduling distributed training jobs, managing model serving,
 - monitoring the quality of models in production, and re-training those models when needed



Experiment tracking vs ML model management vs MLOps(2)

how is experiment tracking different from ML model management?

- Experiment tracking focuses on the iterative model development phase
 - when you try many things to get your model performance to the level you need
- **ML model management starts when models go to production:**
 - streamlines moving models from experimentation to production
 - helps with model versioning
 - organizes model artifacts in an ML model registry
 - helps with testing various model versions in the production environment
 - enables rolling back to an old model version if the new one seems to be going crazy
 - But not every model gets deployed.
- **Experiment tracking is useful even if your models don't make it to production (yet).**
 - And in many projects, especially those that are research-focused, they may never actually get there
 - But having all the metadata about every experiment you run ensures that you will be ready when this magical moment happens.

Why does experiment tracking matter?

4 ways in which experiment tracking can actually improve ML workflow

neptune.ai

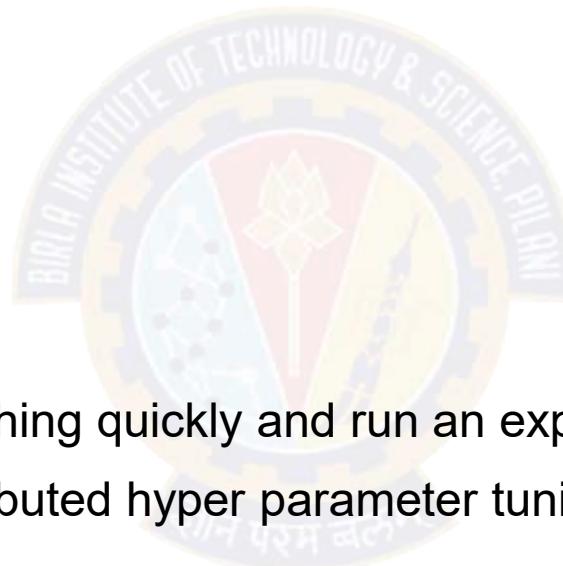
4 ways **experiment tracking** can make your **workflow better**

- 01.**
One source of truth
All your experiments and models are organized in a single place.
- 02.**
Automated process
It lets you compare experiments, analyze results, and debug model training with little extra work.
- 03.**
Better collaboration
You can see what everyone is doing, share results, and access experiment data programmatically.
- 04.**
Live monitoring
You can see your ML runs live and manage experiments from anywhere and anytime.

neptune.ai/blog/ml-experiment-tracking

All ML experiments are organized in a single place

- There are many ways to run ML experiments or model training jobs:
 - Private laptop
 - PC at work
 - A dedicated instance in the Cloud
 - University cluster
 - Kaggle kernel or Google Colab
 - And many more.
- Sometimes just want to test something quickly and run an experiment in a notebook
- Sometimes want to spin up a distributed hyper parameter tuning job
- Either way, during the course of a project (especially when there are more people working on it),
 - can end up having your experiment results scattered across many machines

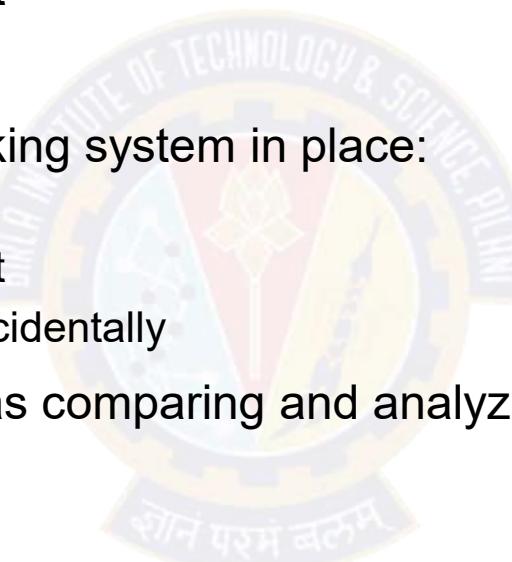


ML experiments are organized in a single place(2)

- With the experiment tracking system, all of experiment results are logged to one experiment repository by design
 - keeping all of experiment metadata in a single place, regardless of where its run, makes experimentation process so much easier to manage
- Specifically, a centralized experiment repository makes it easy to:
 - Search and filter experiments to find the information you need quickly
 - Compare their metrics and parameters with no additional work
 - Drill down and see what exactly it was that you tried (code, data versions, architectures)
 - Reproduce or re-run experiments when you need to
 - Access experiment metadata even if you don't have access to the server where you ran them
- Additionally, **can sleep peacefully** knowing that all the ideas tried are safely stored, and can always go back to them later

Compare experiments, analyze results, debug model training with little extra work

- Whether debugging training runs, looking for improvement ideas, or auditing current best models, comparing experiments is important
- But when don't have any experiment tracking system in place:
 - the way you log things can change,
 - you may forget to log something important
 - you may simply lose some information accidentally
- In those situations, something as simple as comparing and analyzing experiments can get difficult or even impossible
- With an experiment tracking system, experiments are stored in a single place,
 - follow the same protocol for logging them, so those comparisons can go really deep



Benefits of tracking

- Proper experiment tracking makes it easy to:
 - Compare parameters and metrics
 - Overlay learning curves
 - Group and compare experiments based on data versions or parameter values
 - Compare Confusion Matrices, ROC curves, or other performance charts
 - Compare best/worst predictions on test or validation sets
 - View code diffs (and/or notebook diffs)
 - Look at hardware consumption during training runs for various models
 - Look at prediction explanations like Feature Importance, SHAP or Lime
 - Compare rich-format artifacts like video or audio
 - ... Compare anything else you logged
- Modern experiment tracking tools will give many of those comparison features (almost) for free
 - Some tools even go as far as to automatically find diffs between experiments or show which parameters have the biggest impact on model performance
- When have all the pieces in one place, might be able to find new insights and ideas
 - just by looking at all the metadata logged
 - especially true when you are not working alone!

What should be tracked in any ML experiment?

- Things that should be keep track of regardless of the project are:
 - Code:
 - preprocessing, training and evaluation scripts, notebooks used for designing features, other utilities.
 - All the code that is needed to run (and re-run) the experiment
 - Environment:
 - The easiest way to keep track of the environment is to save the environment configuration files like `Dockerfile` (Docker), `requirements.txt` (pip) or `conda.yml` (conda)
 - Can also save the Docker image on Docker Hub, but I find saving configuration files easier
 - Data:
 - saving data versions (as a hash or locations to data files) makes it easy to see what your model was trained on
 - can also use modern data versioning tools like DVC (and save the .dvc files to your experiment tracking tool)
 - Parameters:
 - saving your run configuration is absolutely crucial
 - Metrics:
 - logging evaluation metrics on train, validation, and test sets for every run is pretty obvious
- Keeping track of those things will let you reproduce experiments, do basic debugging, and understand what happened at a high-level.
 - That said, can always log more things to gain even more insights.

How to set up experiment tracking?

- There are (at least) a few options. The most popular being:
 - Spreadsheets + naming conventions
 - Versioning configuration files with Github
 - Using modern experiment tracking tools



Spreadsheets + naming conventions

(but please don't use it)

- A common approach
 - Simply create a big spreadsheet to put all of the information (metrics, parameters, etc) and a directory structure
 - where things are named in a certain way
 - names usually end up being really long like 'model_v1_lr01_batchsize64_no_preprocessing_result_accuracy082.h5'
 - Whenever an experiment is run, look at the results and copy them to the spreadsheet
- In some situations, it can be just enough to solve experiment tracking problems
 - may not be the best solution but it is quick and simple
 - ...things can fall apart really quickly!

Spreadsheets + naming conventions(2)

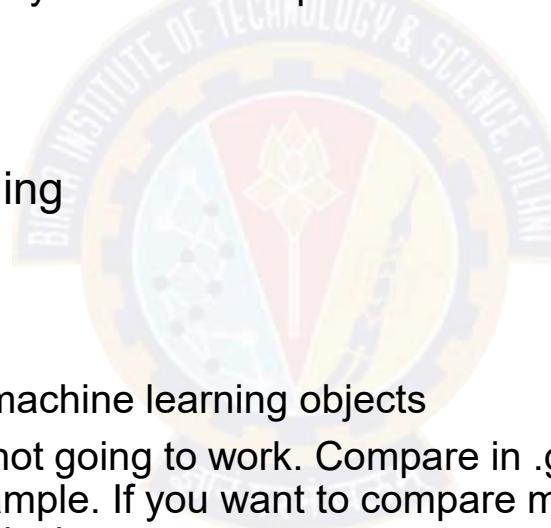
(but please don't use it)

- A few major reasons why tracking experiments in spreadsheets doesn't work for many people:
 - have to remember to track them
 - If something doesn't happen automatically, things get messy, especially with more people involved
 - have to be sure that you or your team will not overwrite things in the spreadsheet by accident
 - Spreadsheets are not easy to version, so if this happens, you are in trouble
 - have to remember to use the naming conventions
 - If someone on your team messes this up, you may not know where the experiment artifacts (model weights, performance charts) for the experiments you ran are.
 - have to back up your artifact directories (remember that things break)
 - When your spreadsheet grows, it becomes less and less usable
 - Searching for things and comparing hundreds of experiments in a spreadsheet (especially if you have multiple people that want to use it at the same time) is not a great experience.

Versioning configuration files with Github

Can version metadata files in GitHub

- Another option is to version all of experiment metadata in Github
 - to commit metrics, parameters, charts, and whatever want to keep track of to Github when running experiment
 - can be done with post-commit hooks where you create or update some files (configs, charts, etc) automatically after experiment finishes
- ... Github wasn't built for ... machine learning
- It can work in some setups but:
 - .git and Github wasn't built for comparing machine learning objects
 - Comparing more than two experiments is not going to work. Compare in .git systems was designed for comparing two branches, master and develop, for example. If you want to compare multiple experiments, take a look at metrics and overlay learning curves you are out of luck.
 - Organizing many experiments is difficult (if not impossible). You can have branches with ideas or a branch per experiment but the more experiments you run the less usable it becomes
 - You will not be able to monitor your experiments live, the information will be saved after your experiment is finished.



Using modern experiment tracking tools

- While can try and adjust general tools to work for machine learning experiments,
 - could just use one of the solutions built specifically for tracking, organizing, and comparing experiments
- Experiment is logged to a central experiment database and displayed in the experiment dashboard,
 - where it can be searched, compared, and drill down to whatever information needed
- Today there are at least a few good tools for experiment tracking!
- Designed to treat machine learning experiments as first-class citizens, and they will always:
 - be easier to use for a machine learning person than general tools
 - have better integrations with the ML ecosystem
 - have more experiment-focused features than the general solutions



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Model Registry – what?

Pravin Y Pawar

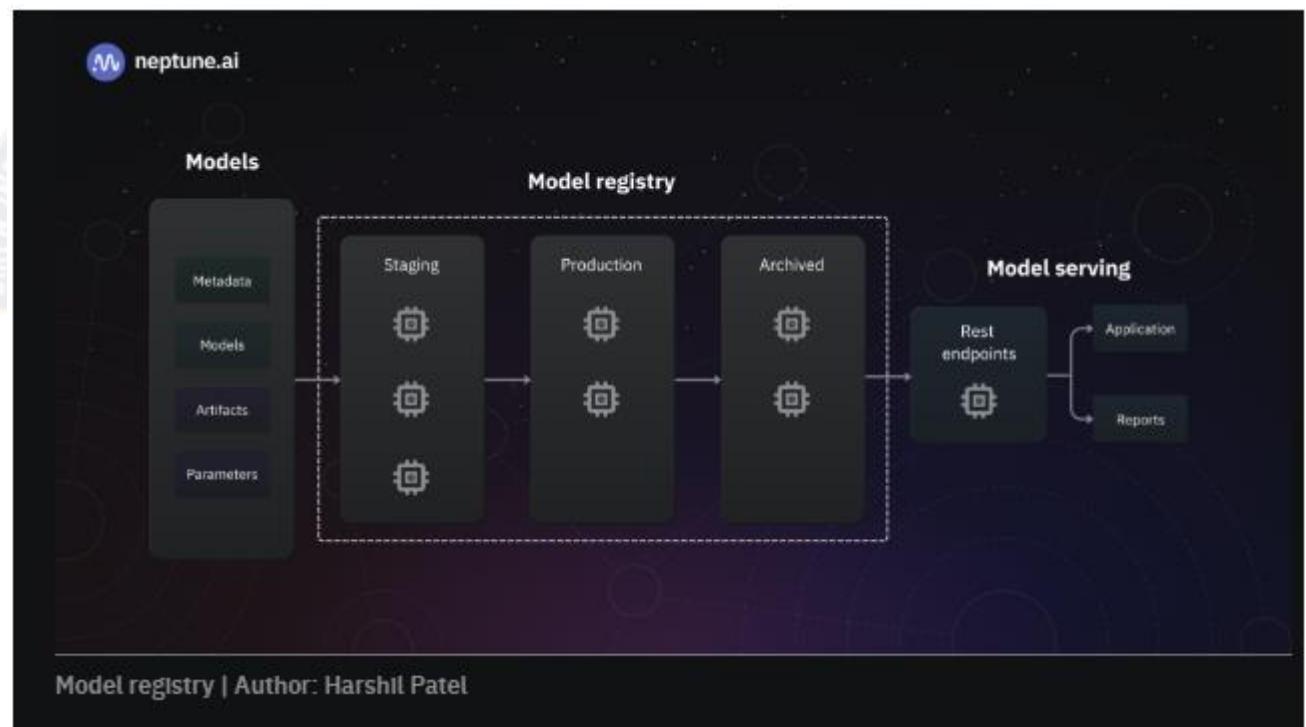
Adapted [from ML Model Registry](#)
By Stephen Oladele

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 make sure 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?
- They want to understand
 - what is running in production,
 - how to improve it
 - or roll back to previous versions
- It makes perfect sense!

What is a model registry?

- An ML model registry serves as a centralized repository, enabling effective model management and documentation
- Allows for
 - clear naming conventions,
 - comprehensive metadata,
 - and improved collaboration between data scientists and operations teams,
 - ensuring smooth deployment and utilization of trained models
- A data scientist can push trained models to the model registry
 - Once in the registry, models are ready to be tested, validated, and deployed to production



Model registry vs model repository

- Model Repository is a storage location for machine learning models
- Model Registry is a more comprehensive system that tracks and manages the full lifecycle of machine learning models.
- However, both are often used interchangeably, and the specific definitions may vary depending on the context or the tools and platforms being used.

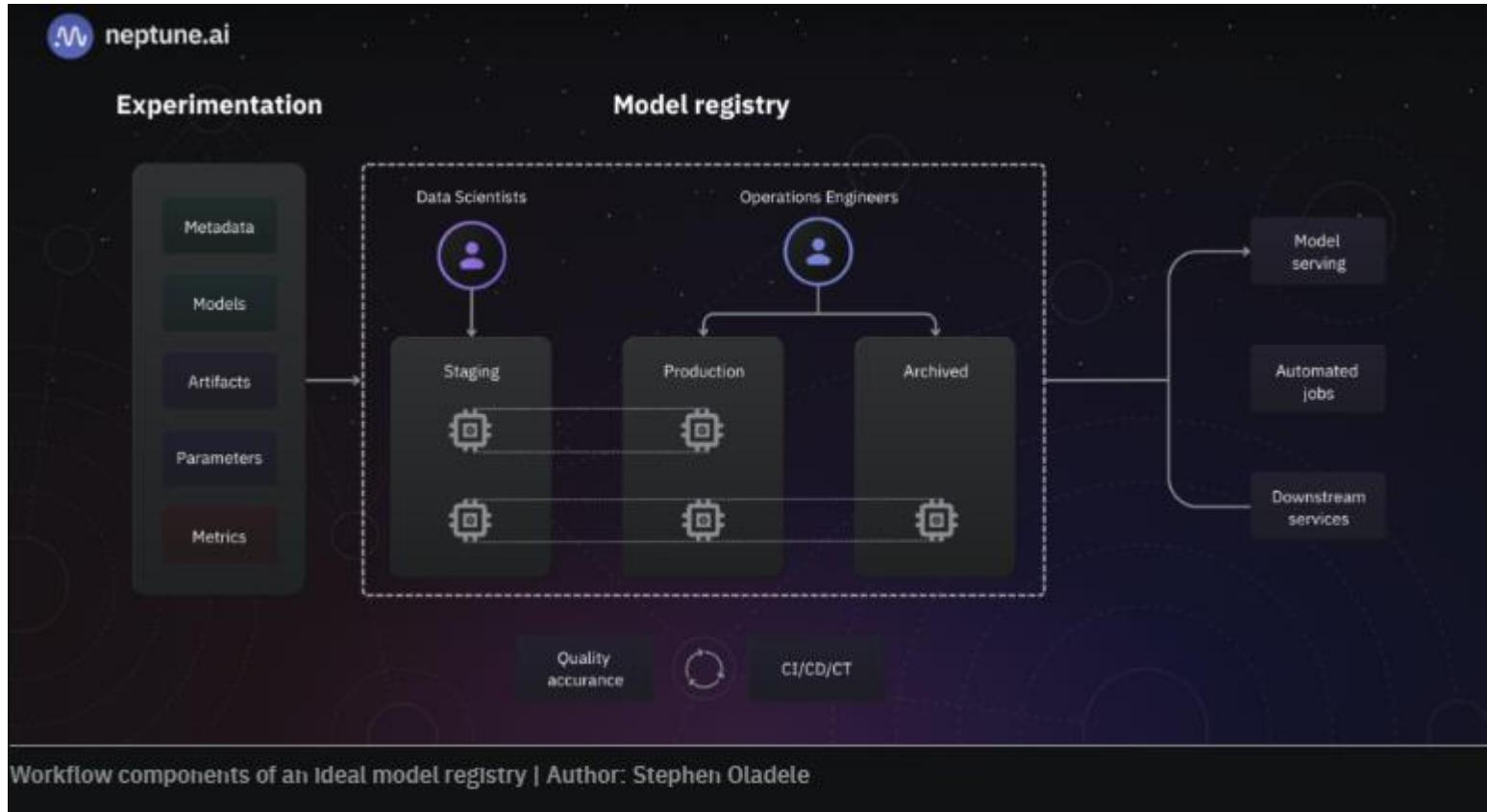
Feature	Model Repository	Model Registry
Storage	Stores machine learning models in a file format	Stores machine learning models and associated metadata
Version control	No	Yes
Lineage tracking	No	Yes
Access control	No	Yes
Other features	None	Can track the performance of models over time, ensure reliability and reproducibility of models, and protect sensitive models from unauthorized access

Model registry vs Experiment tracking

- Model registry has to integrate with the Experiment management system (which tracks the experiments) to register models from various experiment runs to make them easier to find and work with.

Parameters	Model registry	Experiment tracking
Purpose	To store trained, production-ready, and retired models in a central repository.	To track experiment runs of different parameter configurations and combinations.
Priority	To make sure models are discoverable and can be accessed by any user or system.	To make sure experiments are easier to manage and collaborate on.
Integration	Integrates with the experiment tracking system to register models from successful experiments including the model and experiment metadata.	Integrates with the training pipeline to perform experiment runs and track experiment details including the dataset version and metadata.
MLOps	A crucial piece of MLOps and production models.	Most useful in the model development phase, has an indirect impact on operationalizing the model.

Model registry key components

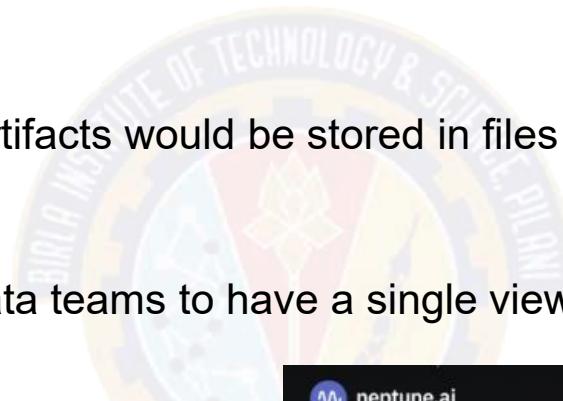


Model registry key features and functionalities

- Acts as a centralized storage for effective collaboration
 - Model registry provides a central storage unit that holds models (including model artifacts) for easy retrieval by an application (or service)
 - Without the model registry, the model artifacts would be stored in files that are difficult to track and saved to whatever source code repository is established
 - The centralized storage also enables data teams to have a single view of the status of all models, making collaboration easier

Bridges the gap between experiment and production activities

- Model registry acts as a glue between ML experimentation and Operations, enabling model development, software development, and operational teams to collaborate.



Model registry key features and functionalities(2)

Providing a central UI

- Model registry provides teams with visibility over their models
- With a central interface, teams can:
 - Search for models,
 - View the status of models (if they are being staged, deployed, or retired),
 - Approve or disapprove models across different stages,
 - And view the necessary documentation.
- This makes model discovery easier for everyone on the team.
- If a model needs to be deployed, the operations teams can easily:
 - Search for it,
 - Look up the validation results and other metrics,
 - Package the model (if needed),
 - And move it from the staging environment to the production environment.
- This improves the way cross-functional teams collaborate on ML projects.

Model registry key features and functionalities(3)

Enables model versioning by tracking different versions of a model

- Model registry enables model versioning by tracking the different versions of a model as they are developed and improved
- This allows to compare different versions of the model, track its performance, and select the best version for deployment
- Here's how it typically works:
 - Model Registration: When a new model is developed or trained, it is registered in the model registry.
 - Version Control: The model registry maintains a history of all registered models and their versions.
 - Model Comparison: The model registry allows users to compare performance metrics, model architectures, hyperparameters, and other relevant information in different versions of a model.
 - Model Tracking: As new versions of the model are developed or trained, they are registered in the model registry as well, incrementing the version number.
 - Retention and Archiving: The model registry typically retains older versions of the models, ensuring a complete history and traceability.
 - By enabling model versioning, the model registry ensures that different iterations of a model can be stored, tracked, compared, and accessed conveniently.

Model registry key features and functionalities(4)

Integrates with experiment management systems or training pipelines

- Model registry integrates with systems that output the trained model
- The trained models could be the raw artifacts (model weights, configuration, and metadata) or models that have been serialized into a file (e.g., an ONNX file)

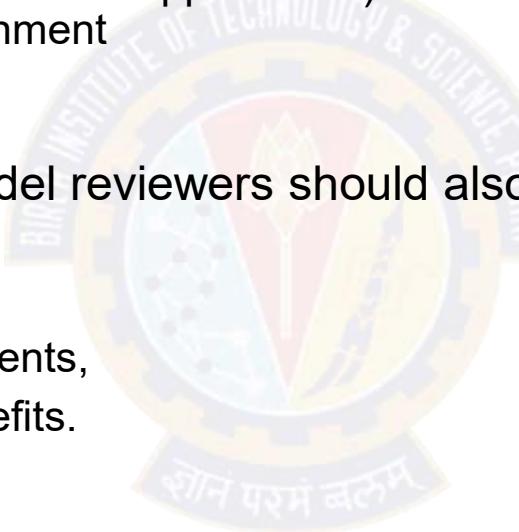
for compatibility with the production environment or containerized (using Docker) to be exported to the production environment.

- Model registries:
 - Register the model,
 - Assign a version to it,
 - Note the version of the dataset the model was trained on,
 - Add annotations and tags,
 - Retrieve the parameters, validation results (including metrics and visualizations), and other relevant model metadata on the model from the experiment management system.
- To make collaboration easier, the registry also includes details such as:
 - The model owner or developer,
 - Experiment run id the model was trained under,
 - Versioned model source code,
 - Environment runtime dependencies used to train the model (and the versions),
 - Comments and model change history,
 - And the model documentation.

Model registry key features and functionalities(5)

Integrates with the staging environment for trained models

- Model Registry provides the functionality for integrating with the staging environment for running all types of checks and balances on the model
 - can include integration testing (with other applications) and other QA tests before the model can be promoted to the production environment
- In the staging environment, the model reviewers should also be able to perform fairness checks on the model to ensure it:
 - Outputs explainable results,
 - Complies with regulatory requirements,
 - And provides useful business benefits.



Model registry key features and functionalities(6)

Integrate with model delivery (CI/CD) tools and services for automation

- Automation is a critical part of building any scalable software
 - In machine learning, building automated pipelines will allow to spend more time building new products rather than maintaining old models
- A Model registry integrates with pipeline automation tools and provides custom APIs that can allow to plug into custom workflow tools
 - For example, using webhooks to trigger downstream actions based on predefined events in the registry.
- Should also be able to configure model promotion schemes through different environments like development (training), staging (testing), and production (serving)
- Performance is a crucial requirement for building automated pipelines
 - Model registries should be highly available for automated jobs that are event or schedule-based to enable continuous training and delivery of the model

Model registry key features and functionalities(7)

- Provides an interface for downstream systems to consume models
 - Model registries provide interfaces that enable downstream services to consume the model through API integration
 - The integration can also track offline and online evaluation metrics for the models
 - makes it easy to build an automated setup with CI/CD/CT with ML pipelines
 - downstream service could be either a model user, an automated job, or a REST serving that can consume the most stable—or any—version of the model
- Integrate with model deployment tools
 - Eventually, models have to be deployed, and the more efficient the deployment process, the better.
 - Model Registries:
 - Integrate with downstream services and REST serving services that can consume the model and serve it in the production environment.
 - Collect real-time (or aggregated) metrics on the production model to log performance details of the model. This will be helpful for comparison between models (deployed and staged), as well as auditing the production model for review



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

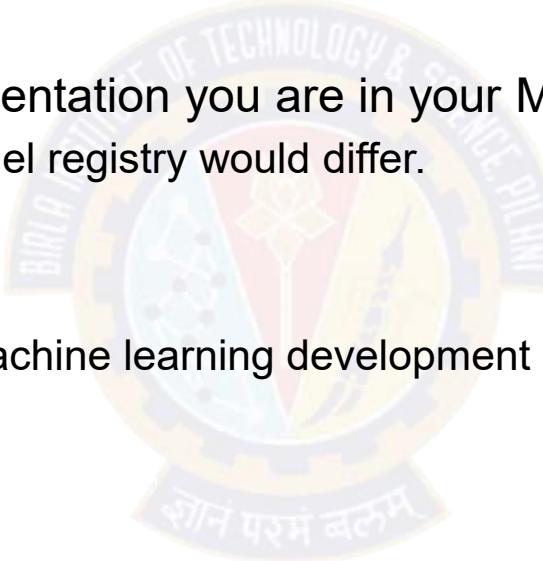
ML Model Registry - How?

Pravin Y Pawar

Adapted [from ML Model Registry](#)
By Stephen Oladele

Where does a model registry fit in the MLOps stack?

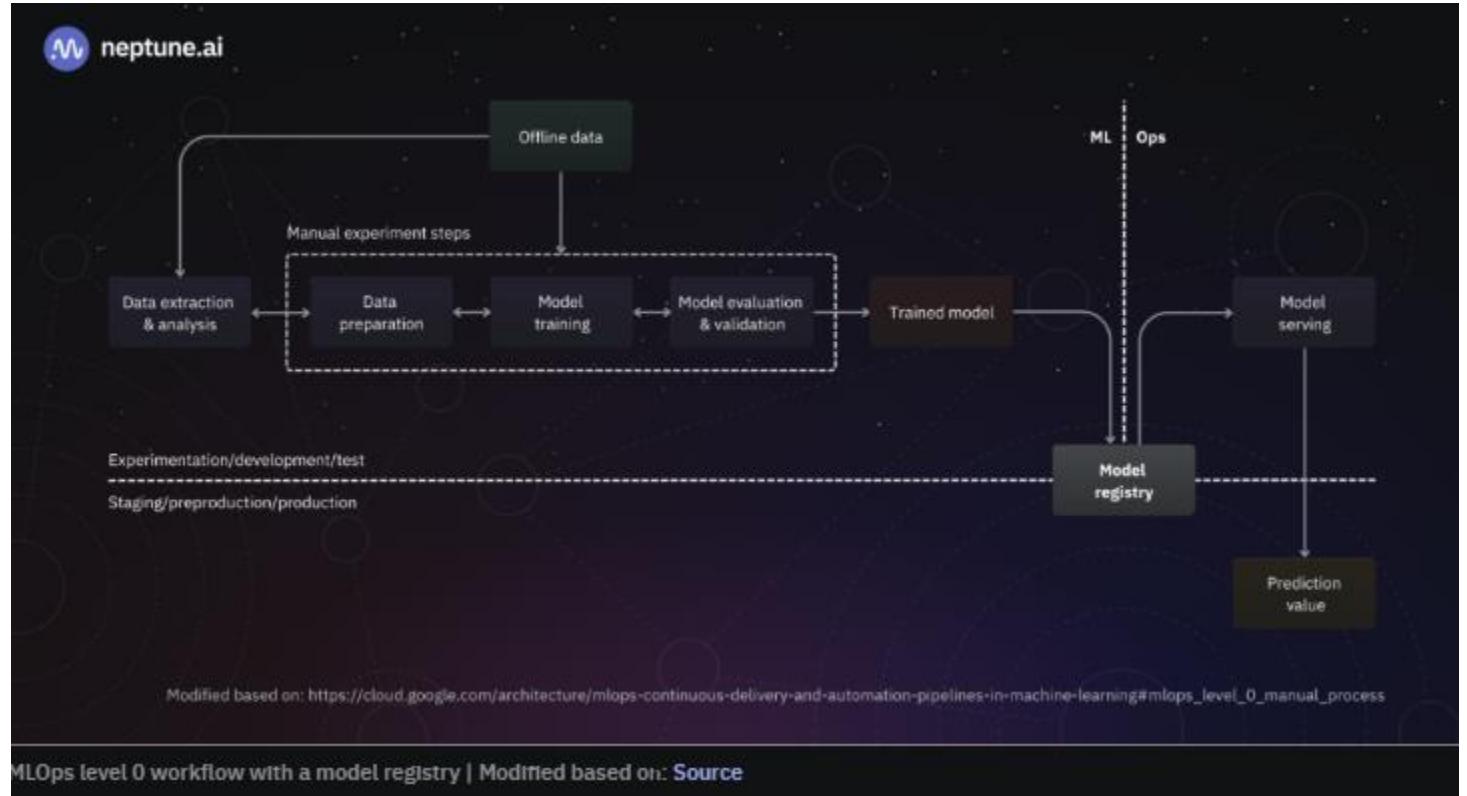
- If want to run machine learning projects efficiently and at scale,
 - most likely need to add a model registry to MLOps stack.
- Depending on what level of implementation you are in your MLOps stack,
 - needs and requirements for a model registry would differ.
- Where does it fit?
 - the model registry sits between machine learning development and deployment.



Where does a model registry fit in the MLOps stack?(2)

Model registry in MLOps level 0

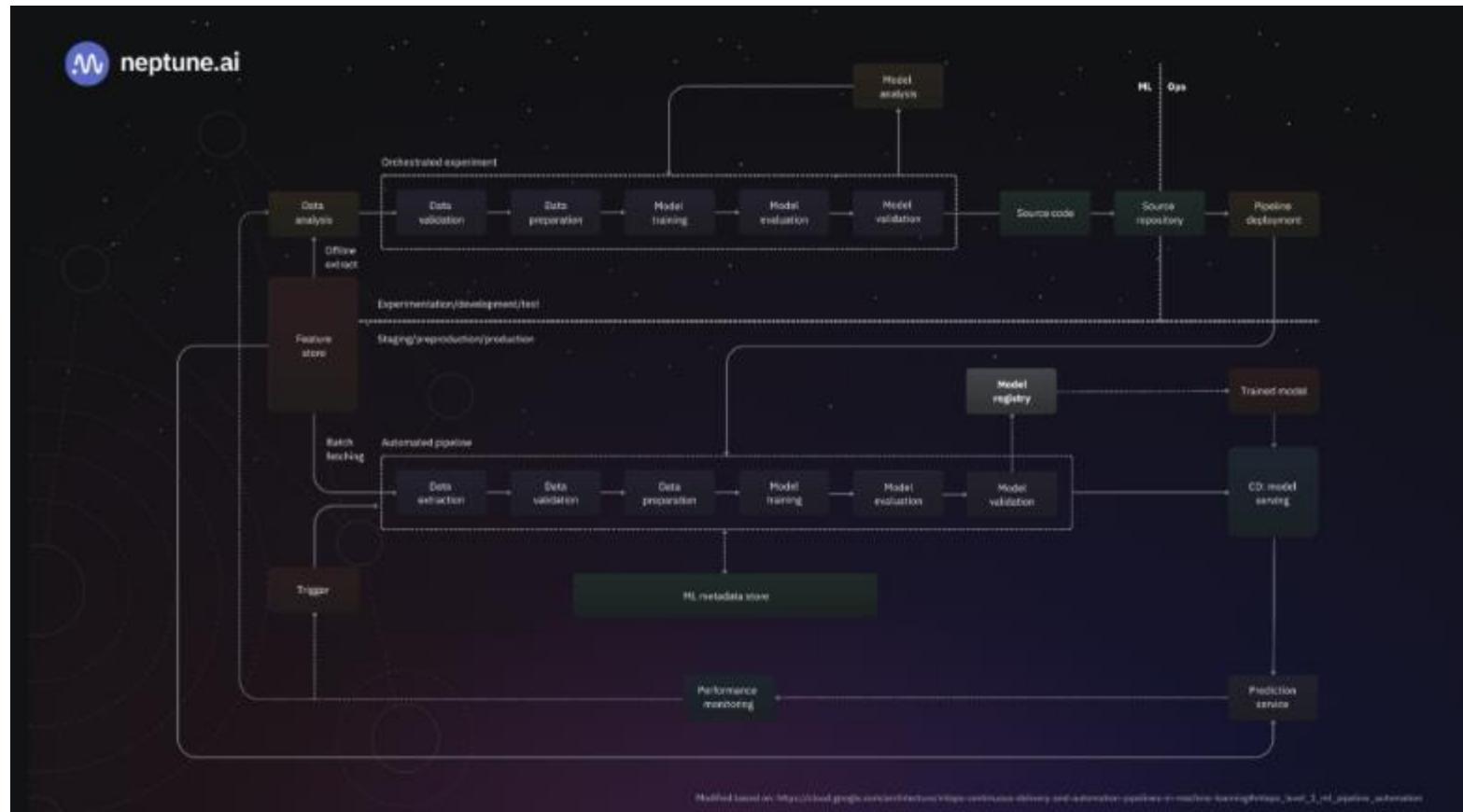
- The output from the experimentation step is fed into the model registry.
 - involves a manual process where the data scientist prepares the model artifact and metadata and could also package them (serialization, containerization) before registering them
- The operations team can push the packaged model to the staging environment
 - for testing before deploying it to a prediction service engine that can integrate with other applications.



Where does a model registry fit in the MLOps stack? (3)

Model registry in MLOps level 1

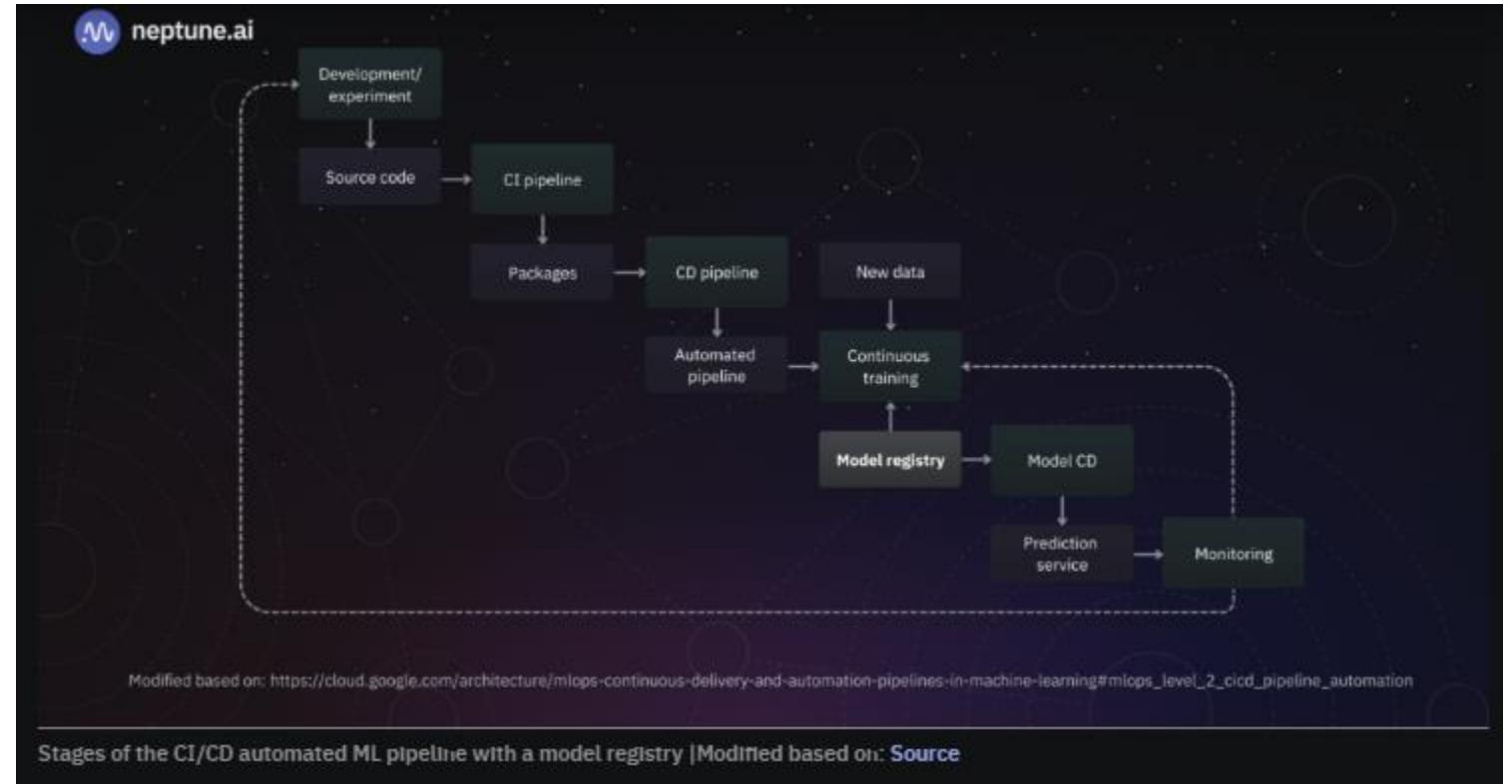
- As opposed to level 0 (where the workflow is a manual process), the goal of the workflow in level 1 is to perform continuous training of the model by automating the ML pipeline. This is one process a model registry enables well because of its ability to integrate with the pipeline.
- At this level, the entire pipeline is deployed, and when models are trained on the provided dataset, the output (trained model and metadata) is fed into the model registry where it can be staged, and if it passes the necessary tests and checks, it can be fed to the continuous delivery pipeline for release.



Where does a model registry fit in the MLOps stack?(4)

Model registry in MLOps level 2

- The role of the model registry in level 2 of the MLOps workflow is also the same as that of level 1—the automated pipeline delivers the trained model to the model registry where it is staged, may be passed through QA checks, and sent to the continuous delivery pipeline:
- The model registry serves as a crucial component in any automated pipeline because event triggers can be integrated with it to promote models with good metrics upon re-training on fresh data or archive models.



Setting up ML model registry – build, maintain or buy?

- Setting up a model registry for MLOps workflow will require you to decide on either
 - Building one,
 - Maintaining one,
 - or Buying one

Setting up ML model registry – build, maintain or buy?(2)

Building a model registry solution

- Like any software solution, if you understand the key functionalities and requirements, you can build a system yourself! This is the case with a model registry.
- May want to set up the following:
 - Object storage for models and artifacts
 - Database for logging model details
 - API integration for both receiving models, promoting models across various environments, and collecting model information from the different environments
 - User interface (UI) for ML teams to interact with a visual workflow
- While building the solution yourself might seem ideal, you should consider the following factors:
 - Incentive: What's the incentive to build out your solution? Is it for customization or for owning a proprietary license to the solution?
 - Human resources: Do you have the talents and skills to build out your solution?
 - Time: How long would it take you to build out a solution and is it worth the wait?
 - Operations: When the solution is eventually built out, who would maintain its operations?
 - Cost: What would it cost you to build a solution, including the maintenance of the solution?

Setting up ML model registry – build, maintain or buy?(3)

Maintaining a self-hosted model registry

- Another option to consider is to maintain an existing solution yourself
 - solution has been built out already, but might have to manage some features, such as object storage and database
 - most of these existing solutions are open-source solutions
- The following are the factors to consider:
 - Type of solution: Are you going to opt for an open-source solution with no license cost or a closed-source solution with license cost?
 - Operations: Who is going to manage the solution? Does the solution support consistent maintenance and software updates?
 - Cost: What is the cost of operating the solution in terms of the infrastructure to host it and the running cost?
 - Features: What features have already been implemented, and what features do you have to build and manage yourself? Is it worth adopting compared to building out your solution?
 - Support: What type of support is available in case things break during operations? Is there a community or dedicated customer support channel? For open-source solutions, while you might have a community, you will likely lack the necessary developer support required to fix things.
 - Accessibility: How easy is it to get started with the solution? Is the documentation comprehensive enough? Can everyone from the model reviewers to the model developers and software engineers intuitively use the solution?

Setting up ML model registry – build, maintain or buy?(4)

Purchase the license to a fully-managed solution

- The final option to consider is subscribing to a fully managed solution where the operations and management of the registry are handled by the solution vendor
 - do not have to worry about building or maintaining a solution
 - just have to ensure systems and services can integrate with the registry
- Here are the factors to consider:
- Industry type: What type of industry is the model built for? What sensitive information have the models learned?
Are there data privacy compliance measures? Is the model only allowed to stay on-premise?
- Features: Are the key features and functionalities of any model registry available in this solution? What extra features are available, and how relevant are they to your workflow?
- Cost: What's the cost of purchasing a license, and do the features justify the cost?
- Security: How secure is the platform hosting the solution? Is it resistant to third-party attacks?
- Performance: Is the registry highly performant? For situations where models are too large, can the registry provide models for services to consume at low latency?
- Availability: What's the uptime of the solution, and does it meet your required service level agreement (SLA)?
- Support: What level of support is available in case things go south?
- Accessibility: How easy is it to get started with the solution? Are the documentation and learning support decent enough?
What's the learning curve in terms of usage?

ML model registry solutions out there

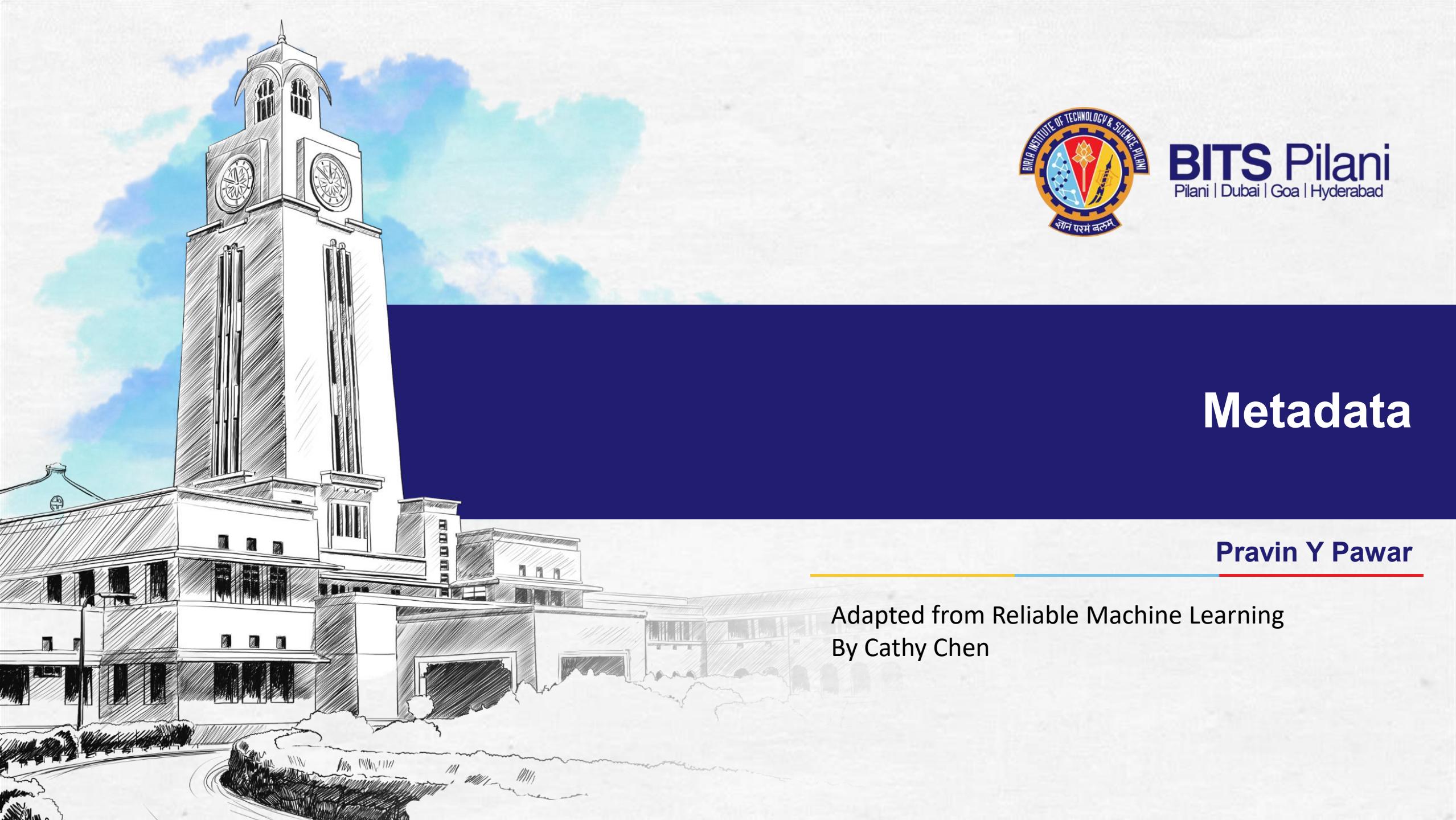
The most popular ones available out there that offer a host of Model Registry features.

Model Registry Features	Azure ML Studio	Neptune.ai	MLFlow	Verta.ai	AWS Sagemaker
Model Versioning	✓	✓	✓	✓	✗
CI/CD/CT compatibility	✓	✗	✓	✓	✗
Model packaging	✗	✗	✓	✓	✓
Model searching	✓	✓	✓	✓	✓
Access control, model review, and promoting models	✗	✓	✓	✓	✓
Model lineage and evaluation history	✓	✗	✗	✓	✓



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Metadata

Pravin Y Pawar

Adapted from Reliable Machine Learning
By Cathy Chen

Dataset Metadata

Metadata about features and labels

- Dataset provenance
 - Where did the data come from?
 - Depending on the source of data, might have a lookup table of logs from various systems, a key for an external data provider with data about when we downloaded the data, or even a reference to the code that generated the data
- Dataset location
 - For some datasets, will store raw, unprocessed data
 - should store a reference to where dataset is kept, as well as perhaps information about where we got it from
 - Some data is created for on an ongoing basis, such as logs from systems - should store the log or datastore reference where
 - that data is stored, or where we are permitted to read from it.
- Dataset responsible person or team
 - Should track which person or team is responsible for the dataset
 - In general, this is the team that chose to download or create the dataset, or the team that owns the system that produces the data
- Dataset creation date or version date
 - Often useful to know the first date a particular dataset was used
- Dataset use restrictions
 - Many datasets have restrictions on their use, either because of licensing or governance constraints
 - should document that in the metadata system for easy analysis and compliance later

Feature Metadata

Track of metadata about feature definitions

- Feature version definition
 - a reference to code or another durable description to what data the feature reads and how it processes the data to create the feature
 - should be updated for every updated version of the feature definition
 - versioning these definitions (and restricting the versions in use) will make the resulting codebase more predictable and maintainable
- Feature definition responsible person or team
 - There are two good use cases for storing this information:
 - figuring out what a feature is for
 - and finding someone who can help resolve an incident when the feature might be at fault
 - In both cases, it is useful to store authorship or maintainer information about the feature
- Feature definition creation date or current version date
 - useful to get a change history of when a feature was most recently updated and when it was originally created
- Feature use restrictions
 - important but trickier to store
 - Features may be restricted from use in some contexts.
 - For example, it may be illegal to use a particular feature in some jurisdictions.
 - Age and gender may be a reasonable predictor of automobile insurance risk models, but insurance is highly regulated
 - may not be permitted to take those fields into account

Label Metadata

Intended to help with the maintenance and development of the labeling system itself, but might also be used by the training system as it uses the labels

- Label definition version
 - Switching to metadata specific to labels and analogously to features,
 - must store the version of any label definitions to understand which labeling instructions the labels were made with
- Label set version
 - Changes to the labels may occur because of incorrect labels getting corrected or new labels being added
 - If the dataset is being used for comparison with an older model, using an older version of the labels may be desirable
- Label source
 - Although not typically needed for training, it is sometimes necessary to know the source of each label in a dataset
 - may be the source that particular label was licensed from,
 - the human who produced the label(along with any QA that was applied to the label),
 - or the algorithm that produced the label if an automated labeling approach was used
- Label confidence
 - Depending on how the labels are produced, might have different estimates of the confidence of correctness for different labels
 - Users of these labels might choose different thresholds to decide which labels to use in training their models

Pipeline Metadata

- Won't spend as much time on: metadata about the pipeline processes themselves
 - data about the intermediate artifacts we have,
 - which pipeline runs they came from,
 - and which binaries produced the
- Type of metadata is produced automatically by some ML training systems
 - for example, ML Metadata (MLMD) is automatically included in TensorFlow Extended (TFX),
 - which uses it to store artifacts about training runs
- Such systems are either integrated into training systems or are somewhat difficult to implement later!

Metadata Systems Overview

Feature systems and labeling systems both benefit from efficient tracking of metadata

- A metadata system is designed to keep track of what we're doing
 - In the case of features and labels, it should minimally keep track of the feature definitions and the versions used in each model's definitions and trained models
- Most organizations start building their data sciences and ML infrastructure without a solid metadata system,
 - only to regret it later!
- The next most common approach is to build several metadata systems, each targeted at solving a particular problem
 - make one for tracking feature definitions and mappings to feature stores
 - need a system for mapping model definitions to trained models, along with data about the engineers or teams responsible for those models
 - need a model serving system is also going to need to keep track of trained model versions, when they were put into production
 - need a model quality or fairness evaluation systems will need to be read from all of these systems in order to identify and track the likely contributing causes of changes in model quality or violations of our proposed fairness metrics

Choices for a metadata system

One system

- Build one system to track metadata from all of these sources
 - makes it simple to make correlations across multiple subsystems, simplifying analysis and reporting
- Such a large system is difficult to get right from a data schema perspective
 - will be constantly adding columns when we discover data we would like to keep track of (and backfilling those columns on existing data)
 - be difficult to stabilize such a system and make it reliable
- From a systems design perspective, should ensure that our metadata systems are never in the live path of either model training or model serving
 - But it's difficult to imagine how feature engineering or labeling can take place without the metadata system being functional
 - can still cause production problems for our humans working on those tasks
- If the needs are simple and well understood, prefer a single system.

Choices for a metadata system (2)

Multiple systems (that work together)

- Can build separate metadata systems for each task we identify
 - Perhaps would have one for features and labels, one for training, one for serving and one for quality monitoring
- Decoupling the system provides the standard advantages and costs that decoupling always does
 - allows to develop each system separately without concern for the others, making them nimbler and simpler to modify and extend
 - an outage of one metadata system has limited production impact on others
 - The cost, though, is the added difficulty of analysis and reporting across those systems
 - Will have processes that need to join data across the features, labeling, training, and serving systems
- The area is rapidly developing and teams expect to continue extending what they track and how they work, multiple systems will simplify the development over time.

Most Ignored!

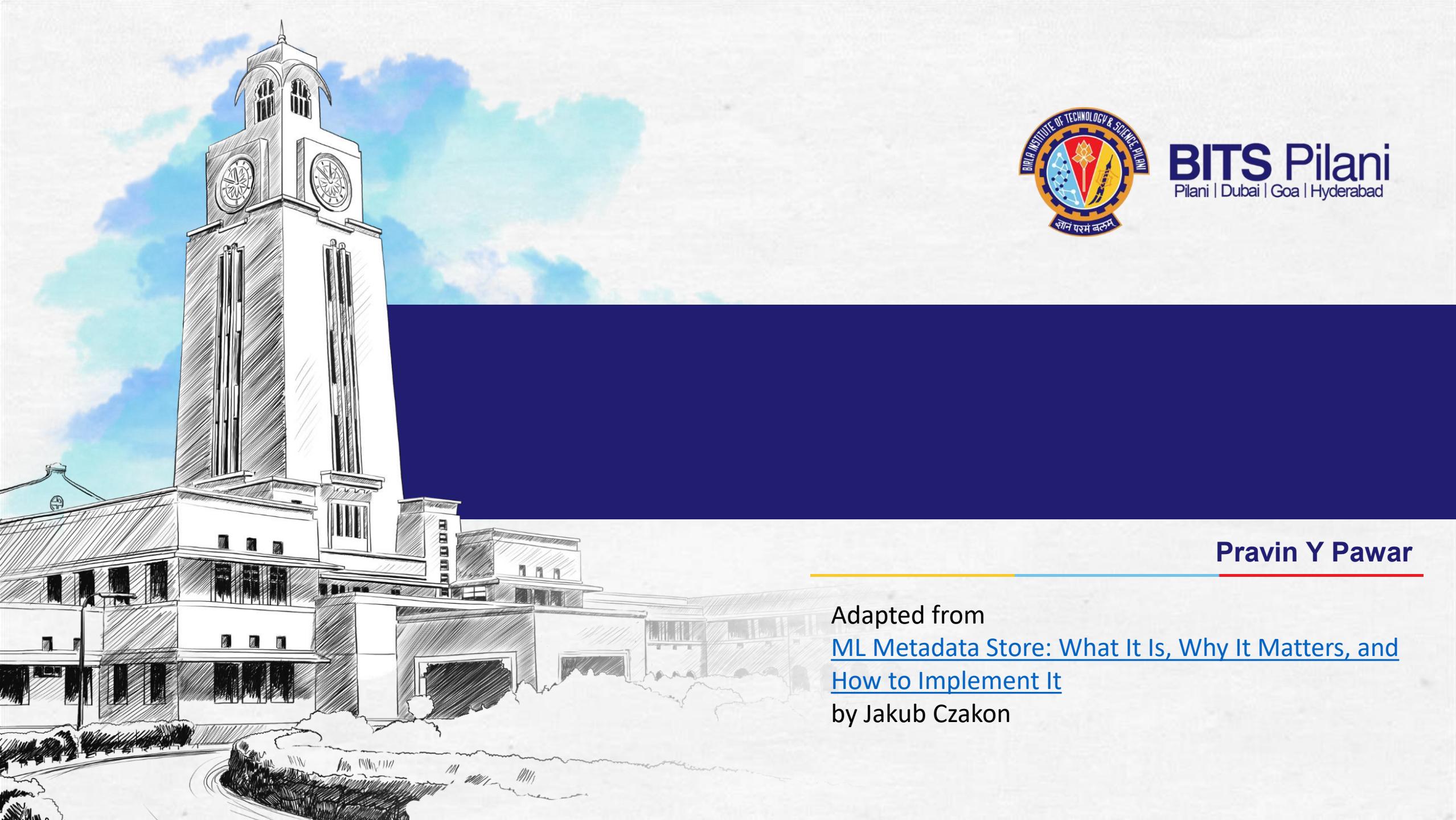
- More generally, metadata systems are often overlooked or deprioritized
 - They should not be
 - They are one of the most effective and direct contributors to productive use of the data in an ML system
 - Metadata unlocks value and should be prioritized





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Pravin Y Pawar

Adapted from
[ML Metadata Store: What It Is, Why It Matters, and How to Implement It](#)
by Jakub Czakon

Need for metadata

But the problems people have with storing and managing ML model metadata are different.

- For some, it is messy experimentation that is the issue
- Others have already deployed the first models to production, but they don't know how those models were created or which data was used
- Some people already have many models in production, but
 - orchestrating model A/B testing,
 - switching challengers and champions,
 - or triggering, testing, and monitoring re-training pipelines is not great.
- ML metadata store can help with all of those things and then some others as well.

What is ML metadata anyway?

- When you do machine learning, there is always a model involved. It is just what machine learning is.
- ML Model could be a classic, supervised model like a lightGBM classifier, a reinforcement learning agent, a bayesian optimization algorithm, or anything else really.
 - But it will take some data, run it through some numbers and output a decision.
 - ... and it takes a lot of work to deliver it into production.
- One has to:
 - train,
 - tune,
 - debug,
 - evaluate,
 - explain,
 - and compare it to baseline.
- If model makes it past the research phase, it also has to :
 - package,
 - deploy,
 - monitor,
 - and re-train it.

What is ML metadata anyway? (2)

- A lot of steps!
 - Don't go through those steps linearly, and you don't go through them once.
 - Takes a lot of iterations of fixing data, fixing model, fixing preprocessing, and all that good stuff.
 - **Each of those steps produces meta-information about the model, or as many people call it, ML model metadata.**
- Those could be:
 - training parameters,
 - evaluation metrics,
 - prediction examples,
 - dataset versions,
 - testing pipeline outputs,
 - references to model weights files,
 - and other model-related things.
- This information helps you know your models.
 - Where the particular version of the model is and quickly rollback to the previous version
 - How the model was built and who created it
 - Which data/parameters/code it used at training
 - How does the new experiment or model version compare to the baseline or previous production models
 - How did it perform at various evaluation stages
 - And when your models are important to someone, when they touch users or help clients make decisions, you better know a lot about them.



Metadata about experiments and model training runs

- During experimentation, you usually care about debugging, visualizing, monitoring your model training to get to the best model.
- To do that, it is a good practice to log anything that happens during the ML run, including:
 - data version: reference to the dataset, md5 hash, dataset sample to know which data was used to train the model
 - environment configuration: requirements.txt, conda.yml, Dockerfile, Makefile to know how to recreate the environment where the model was trained
 - code version: git SHA of a commit or an actual snapshot of code to know what code was used to build a model
 - hyperparameters: configuration of the feature preprocessing steps of the pipeline, model training, and inference to reproduce the process if needed
 - training metrics and losses: both single values and learning curves to see whether it makes sense to continue training
 - record of hardware metrics: CPU, GPU, TPU, Memory to see how much your model consumes during training/inference
 - evaluation and test metrics: f2, acc, roc on test and validation set to know how your model performs
 - performance visualizations: ROC curve, Confusion matrix, PR curve to understand the errors deeply
 - model predictions: to see the actual predictions and understand model performance beyond metrics
 - ...and about a million other things that are specific to your domain

Metadata about artifacts

- Apart from experiments and model training runs, there is one more concept used in ML projects: artifact
 - input or output of those runs can be used in many runs across the project
 - Artifacts can change during the project, and typically have many versions of the same artifact at some point in your ML lifecycle
 - Artifacts could be datasets, models, predictions, and other file-like objects.
- For artifacts you may want to log:
 - Reference: Paths to the dataset or model (s3 bucket, filesystem)
 - Version: Dataset or model md5 hash to let you quickly see the diff
 - Preview: Dataset/prediction preview (head of the table, snapshot of the image folder) to see what this dataset is about
 - Description: additional info about the artifact that will help you understand what it is. For example, you may want to log column names for a tabular dataset artifact
 - Authors: who created modified this artifact and when
 - And many other things that may be specific to your project as the size of the dataset, type of the ML model and others

Metadata about trained models

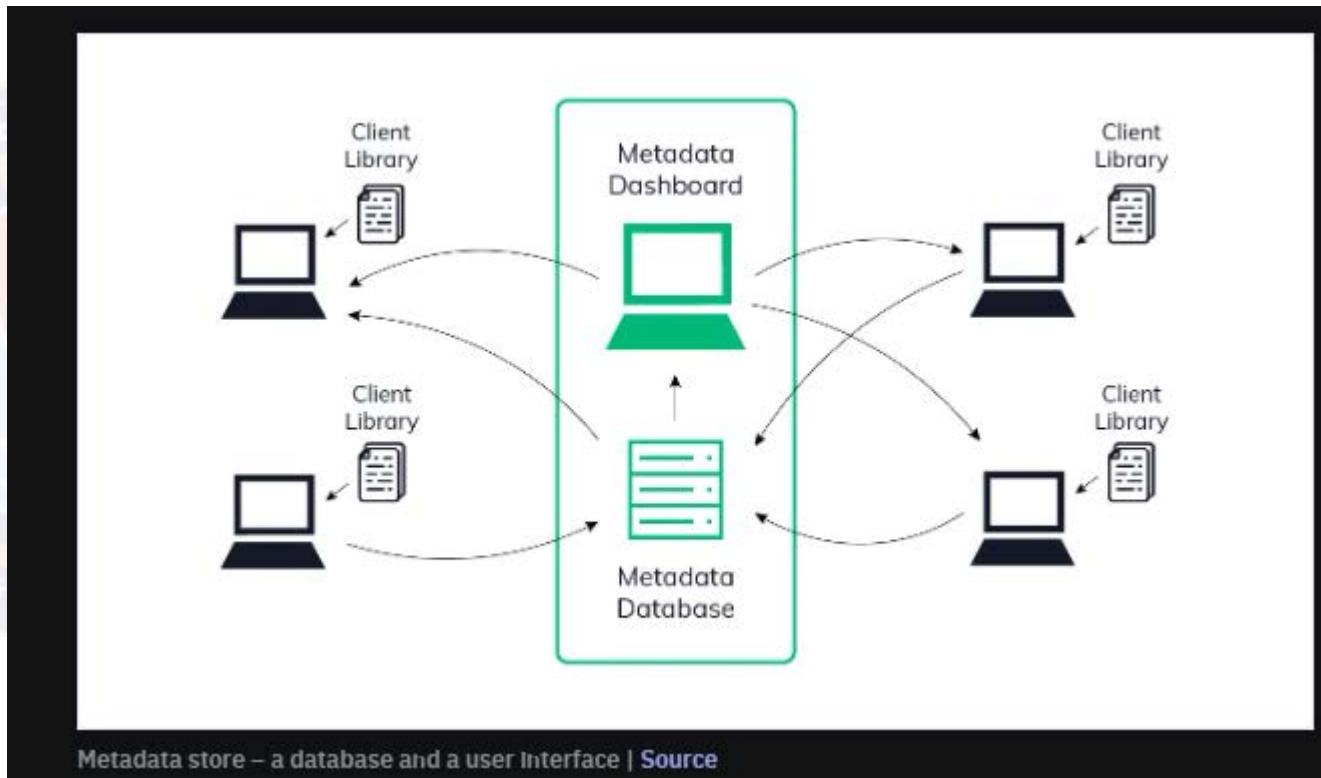
- Trained models are such an important type of artifact in ML projects
 - Once your model is trained and ready for production, needs change from debugging and visualization to knowing how to deploy a model package, version it, and monitor the performance on prod.
- So the ML metadata may want to log are:
 - Model package: Model binary or location to your model asset
 - Model version: code, dataset, hyperparameters, environment versions
 - Evaluation records: History record of all the evaluations on test/validation that happened over time
 - Experiment versions: Links to recorded model training (and re-training) runs and other experiments associated with this model version
 - Model creator/maintainer: who build this model, and who should you ask if/when things go wrong
 - Downstream datasets/artifacts: references of datasets, models, and other assets used downstream to build a model. This can be essential in some orgs for compliance.
 - Drift-related metrics: Data drift, concept drift, performance drift, for all the “live” production
 - Hardware monitoring: CPU/GPU/TPU/Memory that is consumed in production.
 - And anything else that will let you sleep at night while your model is sending predictions to the world

Metadata about pipeline

- You may be at a point where your ML models are trained in a pipeline that is triggered automatically:
 - When the performance drops below certain thresholds
 - When new labeled data arrives in the database
 - When a feature branch is merged to develop
 - Or simply every week
- Likely have some CI/CD workflow connected to a pipeline and orchestration tool like Airflow, Kubeflow, or Kedro
 - every trigger starts the execution of a computation DAG (directed acyclic graph) where every node produces metadata
- Need for the metadata is a bit different than for experiments or models
 - This metadata is needed to compute the pipeline (DAG) efficiently:
 - Input and output steps: information about what goes into a node and what goes out from a node and whether all the input steps are completed
 - Cached outputs: references to intermediate results from a pipeline so that you can resume calculations from a certain point in the graph

What is an ML metadata store?

- ML metadata store is a “store” for ML model-related metadata
 - place where one can get anything one need when it comes to any and every ML model you build and deploy.
- More specifically, ML metadata store lets you:
 - log,
 - store,
 - display,
 - monitor,
 - compare,
 - organize,
 - filter,
 - and query all model-related metadata.
- In short, it gives a single place to manage all the ML metadata about experiments, artifacts, models, and pipelines we have listed in the previous section.
- You can think of it as a database and a user interface built specifically to manage ML model metadata
- It typically comes with an API or an SDK (client library) to simplify logging and querying ML metadata.



Metadata store – a database and a user interface | [Source](#)

Repository vs Registry vs Store

- In the context of ML metadata,
 - all of those things are basically databases created to store metadata that have slight functional differences
- A metadata repository
 - place where store metadata objects along with all the relationships between those objects
 - can use GitHub as a repository where save all the evaluation metrics as a file created via post-commit hook or log parameters and losses during training to an experiment tracking tool which in this context is actually a repository
- A metadata registry
 - place where you “checkpoint” the important metadata
 - register something you care about and want to easily find and access it later
 - A registry is always for something specific. There are no general registries
 - For example, you may have a model registry that lists all the production models with references to the ML metadata repository where the actual model-related metadata is.
- A metadata store
 - place where you go “shopping” for metadata
 - For ML models, it is a central place where you can find all the model, experiment, artifact, and pipeline-related metadata
 - The more you need to come to the “shop”, search metadata “products”, compare them and “buy” them, the more it is a store rather than a repository.

Two flavors of ML metadata store

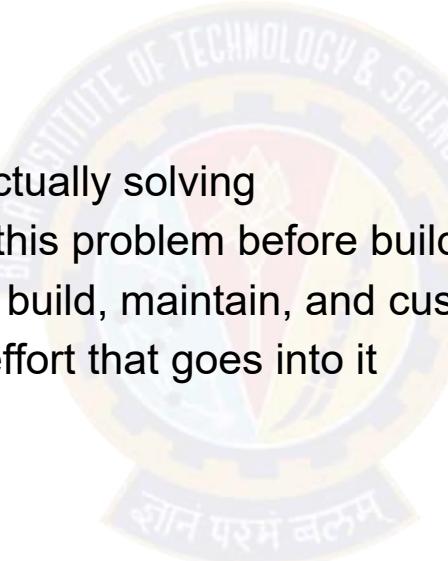
Pipeline-first vs model-first ML metadata store

- As your ML organization matures, you get to a point when training models happen in some automated, orchestrated pipelines
 - At that moment, running experiments, training, and re-training production models is always associated with executing a pipeline.
- ML metadata store can:
 - Treat pipelines as first-class objects and associate resulting models and experiments with them
 - Treat models and experiments as first-class objects and treat pipelines as a mode of execution
 - Depending on what you put in a center, your ML metadata store will do slightly different things
- If you focus on the process, the model building pipeline, you get something like MLMD, which powers pipeline and orchestration tools like Kubeflow and TensorFlow Extended (TFX).
- If you focus on the output, the model, you get something like MLflow or Neptune.

How do you set up an ML metadata management system?

Build vs maintain vs buy

- It's an old dilemma!
 - People who work in software face it many times in their careers.
- To make a decision, you should:
 - understand what problem you are actually solving
 - see if there are tools that can solve this problem before building it
 - assess how much time it will take to build, maintain, and customize it
 - don't underestimate the amount of effort that goes into it



How do you set up an ML metadata management system?(2)

Build a system yourself

Pros	Cons
No licence cost	You have to implement it
You can create it exactly for your use case	You have to set up and maintain the infrastructure
You can improve/change it however you like	You have to document it for other people
	You have to fix bugs
	You have to implement improvements and integrations with other tools
	When software doesn't work it is your fault

How do you set up an ML metadata management system?(3)

Maintain open source

Pros	Cons
No licence cost	You have to set up and maintain the infrastructure
You don't have to implement it	You have to adjust it to your needs
Documentation and community support is already there	You have to count on community support
Other people are creating improvements that you may care about	You can push for new features but may have to implement them yourself
	When software doesn't work it is your fault
	Project may get abandoned by the creators/maintainers/community

How do you set up an ML metadata management system?(4)

Buy a solution

Pros	Cons
You don't have to implement it	Licence fee
Documentation and support is provided by the vendor	You may not be able to fix bugs/create improvements yourself and the vendor may be slow to fix them
Infrastructure is already there (for hosted options)	Company may go out of the market
When you need features you can request them	
When software doesn't work it is their fault (and they have to fix it)	



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Model Packaging

Pravin Y Pawar

Adapted from “Engineering MLOps”
By Emmanuel Raj

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



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
 - where a piece of code or module is imported into the existing system and it works
- Engineering ML solutions is non-deterministic 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
 - are packaged into a file or a bunch of files or containers
- ML models need to be packaged for the following reasons:
 - Portability
 - Inference
 - Interoperability
 - Deployment agnosticity

Why package ML models?(2)

Portability and Inference

- Portability
 - Packaging ML models into software artifacts enables them to **be shipped or transported from one environment to another**
 - can be done **by shipping a file or bunch of files or a container**
 - can transport the artifacts and replicate the model in various setups
 - For example, a packaged model can be deployed **in a virtual machine or serverless setup**
- Inference
 - Enables to **serve the ML models in real time for ML inference**
 - ML inference - process that involves processing real-time data using ML models to calculate an output,
 - for example, a prediction or numerical score
- Effective ML model packaging - **a serialized model or container** - facilitate
 - **deployment and serve** the model to make predictions
 - analyze data in real time or in batches

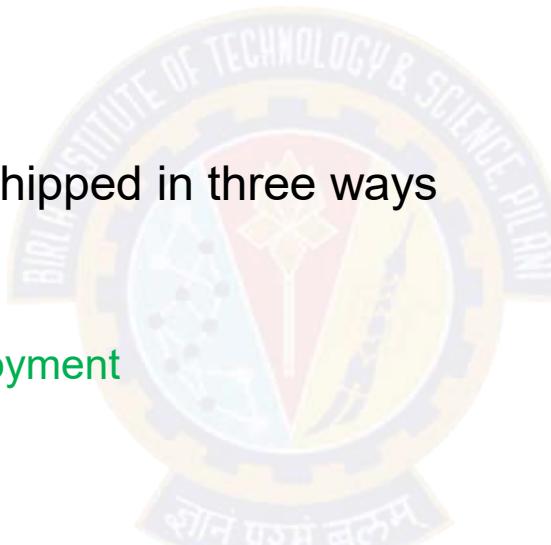
Why package ML models?(3)

Interoperability and Deployment agnosticity

- Interoperability
 - ability of two or more models or components to exchange information and to use exchanged information
 - in order to learn or fine-tune from each other and perform operations with efficiency
 - Exchanged information can be in the form of data or software artifacts or model parameters
 - enables models to fine-tune, retrain, or adapt to various environments
 - Packaging ML models is the foundation for enabling ML model interoperability
- Deployment agnosticity
 - Packaging ML models into software artifacts such as serialized files or containers
 - enables the models to be shipped and deployed in various runtime environments, such as in
 - a virtual machine,
 - a container serverless environment,
 - a streaming service,
 - microservices,
 - or batch services
 - Opens opportunities for portability and deployment agnosticity using the same software artifacts that an ML model is packaged in

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
 - as Serverless Functions



How to package ML models?(2)

Serialized files

- Serialization is a vital process for packaging an ML model
 - enables model portability, interoperability, and model inference
- Serialization is the method of converting an object or a data structure (for example, variables, arrays, and tuples)
 - into a storable artefact, for example, into a file or a memory buffer
 - that can be transported or transmitted (across computer networks)
 - only saves the data structure as it is in a storable artefact such as a file
- The main purpose of serialization is to reconstruct the serialized file into its previous data structure in a different environment
 - for example, a serialized file into an ML model variable
- A newly trained ML model can be
 - serialized into a file
 - exported into a new environment
 - can de-serialized back into an ML model variable or data structure for ML inferencing

How to package ML models?(3)

Serialized files Formats

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

Popular ML model serialization formats

How to package ML models?(4)

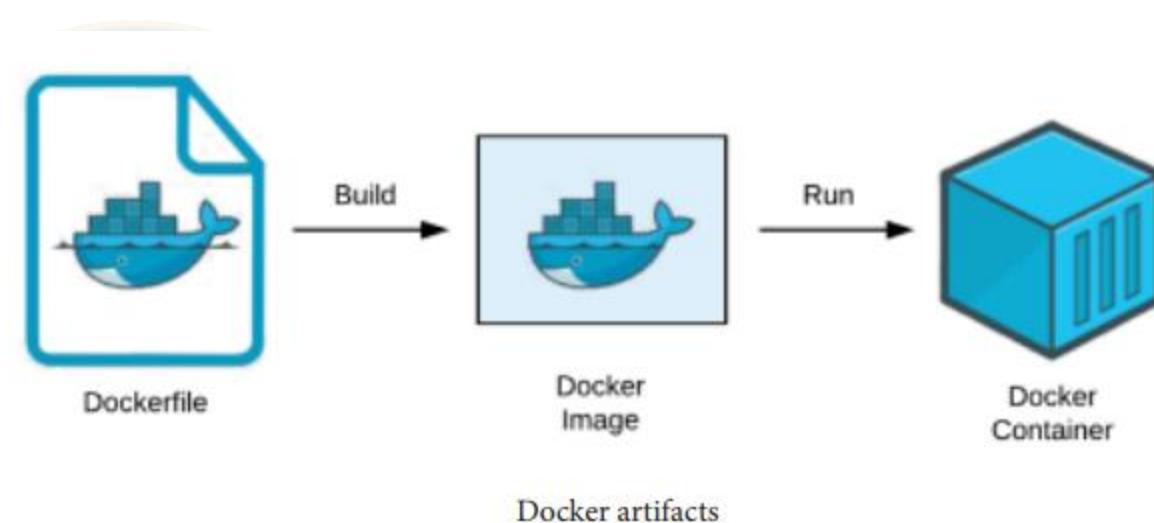
Packetizing or containerizing

- Every environment possesses different challenges when it comes to deploying ML models,
 - in terms of compatibility, robustness, and scalability
 - can be avoided by standardizing some processes or modules
 - containers are a great way to standardize ML models and software modules
- A container is a standard unit of software made up of code and all its dependencies
 - enables the quick and reliable operation of applications from one computing environment to another
 - enables the software to be environment- and deployment-agnostic
 - managed and orchestrated by Docker
- Docker - industry standard at developing and orchestrating containers
 - developed to make it convenient to build, deploy, and run applications by using containers

How to package ML models?(5)

Docker

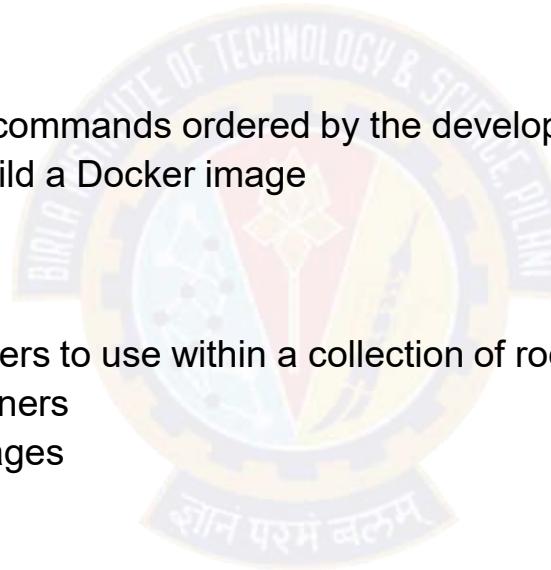
- By using containers, a developer can package an application with its components and modules,
 - such as files, libraries, and other dependencies, and deploy it as one package
- Docker containers are built using Dockerfiles, which are used to containerize an application
 - After building a Docker image, a Docker container is built
 - A Docker container is an application running with custom settings as orchestrated by the developer
 - A Dockerfile is built into a Docker image, which is then run as a Docker container



How to package ML models?(6)

Docker and ML Models

- Foundational components for building and running containers
 - A Dockerfile, Docker image, and a Docker container
- Dockerfile:
 - a text document containing a set of Docker commands ordered by the developer to build a Docker image
 - Docker is able to read the Dockerfile and build a Docker image
- Docker image:
 - a sequential collection of execution parameters to use within a collection of root filesystems within a container during runtime
 - Docker images are like a snapshot of containers
 - Containers are constructed from Docker images
- Docker container:
 - Containers are constructed from Docker images
 - A container is a runtime instance of a Docker image
- **ML models can be served in Docker containers for robustness, scalability, and deployment agnosticity**



How to package ML models?(7)

Microservices

- Microservices enable the collection of services that are independently deployable
- Each of these services is highly maintainable, testable, and loosely coupled
 - are orchestrated by architecture that is organized around business capabilities to enable a system to serve business needs
- Fragmented microservices are now developed by a dedicated team
 - Each microservice is isolated and less dependent on one another
 - easier to develop and maintain

How to package ML models?(9)

Microservices: Construction

- 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 and managing Docker images can be performed using container management tools
 - such as Kubernetes
- Docker enables extreme portability and interoperability
 - Docker images can be easily deployed to any popular cloud service,
 - such as Google Cloud, Azure, or AWS
- Docker images can be deployed and managed to any
 - Docker corporate server or data center or real-time environment as long as it supports Docker.

How to package ML models?(10)

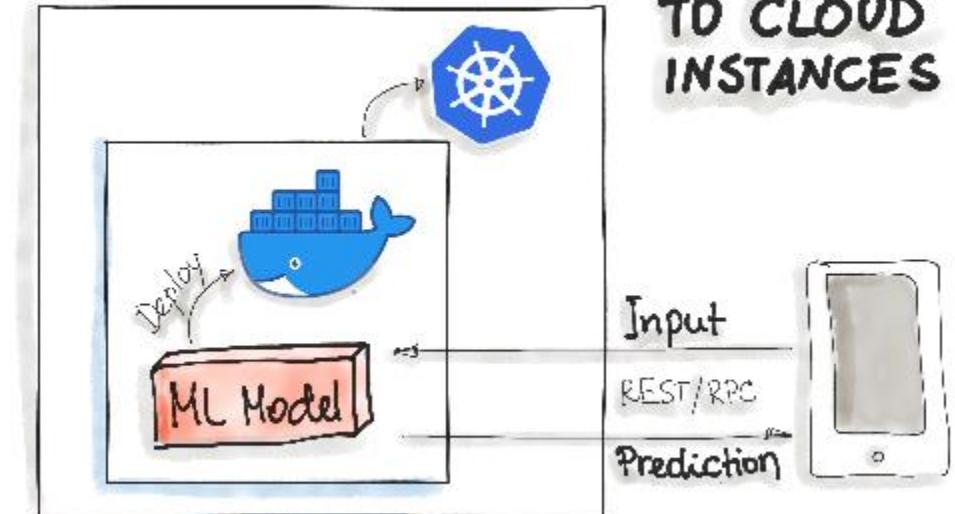
Microservices: Serving

- A popular way to serve ML models - Microservices can be served in a REST API format
- Python frameworks, such as Flask, Django, and FastAPI
 - Popular in enabling ML models to serve as REST API microservices
- Kubeflow - is a robust way of serving a models on scale
 - cloud-agnostic and can be run on-premises or on local machines
 - based on Kubernetes, but keeps the Kubernetes details and difficulties inside a box
 - To facilitate robust and scalable system operations, software developers can sync with Dockerized microservices via a REST API

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
 - means **deploy a container that wraps an ML model inference code**
- For on-premise, cloud, or hybrid deployments, **Docker** is considered to be de-facto standard containerization technology
 - One ubiquitous way is to **package the whole ML tech stack (dependencies) and the code for ML model prediction into a Docker container**
 - Then **Kubernetes** or an alternative (e.g. **AWS Fargate**) does the **orchestration**
 - The **ML model functionality**, such as prediction, is then available through a **REST API** (e.g. implemented as Flask application)

INFRASTRUCTURE: ML MODEL DEPLOYMENT TO CLOUD INSTANCES



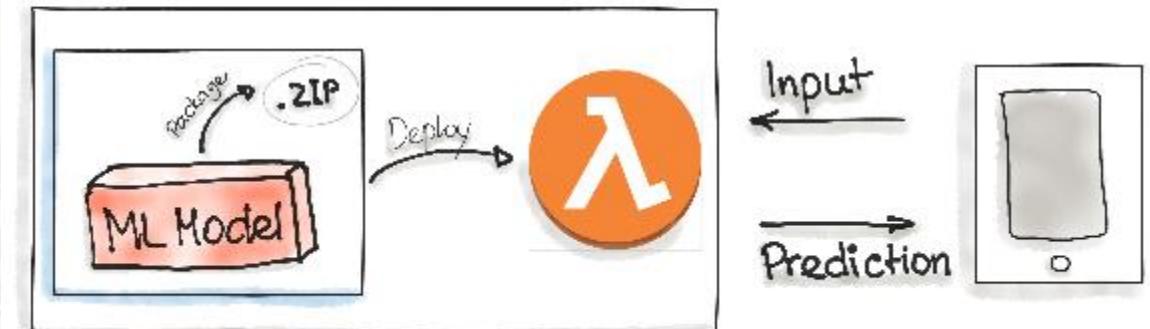
[three-levels-of-ml-software](#)

Deploying ML Models as Serverless Functions

- Various cloud vendors already provide machine-learning platforms where you can deploy model with their services
 - Amazon AWS Sagemaker,
 - Google Cloud AI Platform,
 - Azure Machine Learning Studio,
 - IBM Watson Machine Learning
- Commercial cloud services also provide **containerization** of ML models
 - AWS Lambda
 - Google App Engine servlet host
- In order to deploy an ML model as a serverless function,
 - the application code and dependencies are packaged into .zip files, with a single entry point function
 - function then could be managed by major cloud providers such as Azure Functions, AWS Lambda, or Google Cloud Functions
 - attention should be paid to possible constraints of the deployed artifacts such as the size of the artifact



INFRASTRUCTURE: ML MODEL DEPLOYMENT AS SERVERLESS FUNCTION



[three-levels-of-ml-software](#)



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Model serialization formats

Pravin Y Pawar

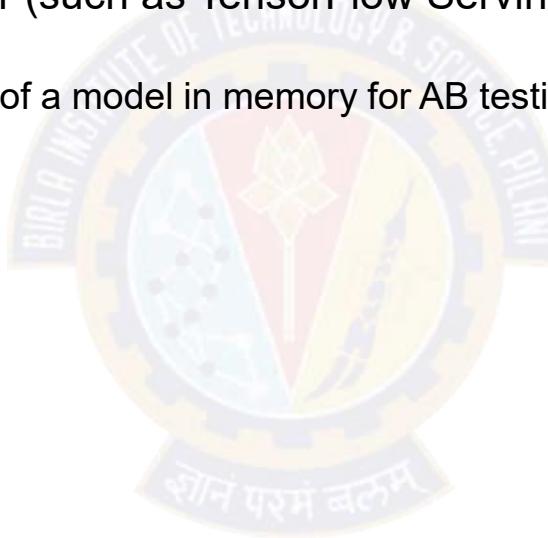
Adapted from "[Three Levels of ML Software](#)"

Model File Formats

- In supervised machine learning, the artefact created after training that is 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
- 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, with a .pb file extension
 - Keras saves models natively as .h5 file
 - Scikit-Learn saves models as pickled python objects, with a .pkl file extension
 - An older format for model serving based on XML, predictive model markup language (.pmml)
 - is still usable on some frameworks, such as Scikit-Learn.

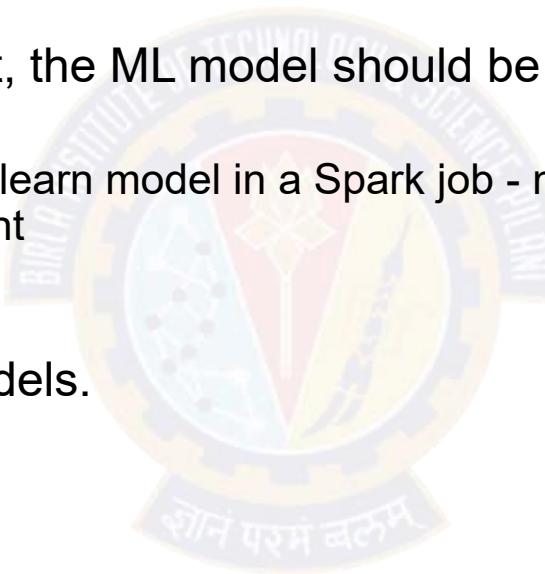
Model files

- Used to make predictions on new data by
 - either batch applications that typically read the model in as a file
 - or a real-time model serving server (such as TensorFlow Serving Server) that reads the model into memory,
 - may even have multiple versions of a model in memory for AB testing



ML Model serialization formats

- Various formats to distribute ML models
- In order to achieve a distributable format, the ML model should be present and should be executable as an independent asset
 - For example, might want to use a Scikit-learn model in a Spark job - 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 and PMML

- **Amalgamation**
 - simplest way to export an ML model - model and all necessary code to run are bundled as one package
 - usually, a single source code file that can be compiled on nearly any platform as a standalone program
 - straightforward concept, and the exported ML models are portable
- For example,
 - can create a standalone version of an ML model by using SKompiler
 - python package provides a tool for transforming trained Scikit-learn models into other forms, such as SQL queries, Excel formulas, Portable Format for Analytics (PFA) files, or SymPy expressions
- **PMML**
 - a format for model serving based on XML with the file extension .pmml
 - has been standardized by the Data Mining Group (DMG)
 - Basically, .pmml describes a model and pipeline in XML
 - does not support all of the ML algorithms
 - usage in open source-driven tools is limited due to licensing issues

Language-agnostic exchange formats(2)

PFA

- PFA (Portable Format for Analytics)
 - designed as a replacement for PMML
 - To run ML models as PFA files, will need a PFA-enabled environment
- From DMG:
 - “A PFA document is a string of JSON-formatted text that describes an executable called a scoring engine. Each engine has a well-defined input, a well-defined output, and functions for combining inputs to construct the output in an expression-centric syntax tree”.
- PFA capabilities include
 - control structures, such as conditionals, loops, and user-defined functions,
 - expressed within JSON, and can, therefore, be easily generated and manipulated by other programs,
 - fine-grained function library supporting extensibility callbacks

Language-agnostic exchange formats(3)

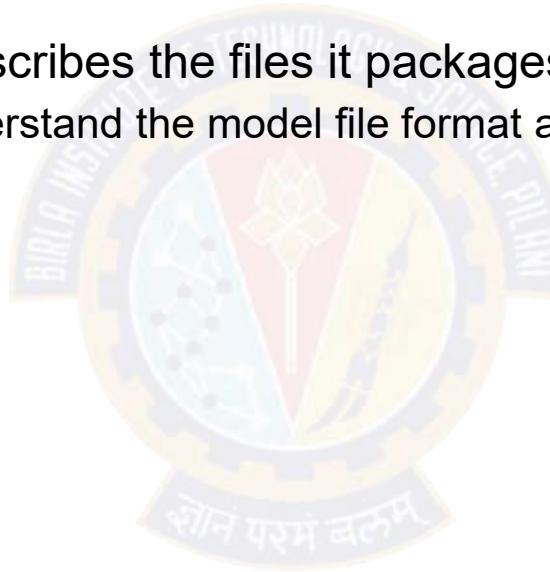
ONNX

- ONNX (Open Neural Network eXchange)
 - an ML framework independent file format
 - was created to allow any ML tool to share a single model format
 - is supported by many big tech companies such as Microsoft, Facebook, and Amazon
- Once the ML model is serialized in the ONNX format,
 - can be consumed by onnx-enabled runtime libraries (also called inference engines) and then make predictions
 - most deep learning tools have ONNX support
- A game changer for industrialized AI as models can be rendered interoperable and framework-independent

Language-agnostic exchange formats(4)

YAML

- YAML is used to package models as part of the MLFlow framework for ML pipelines on Spark
- MLFlow stores a YAML file that describes the files it packages for model serving
 - so that deployment tools can understand the model file format and know what files to deploy



Vendor-specific Exchange formats

- Scikit-Learn
 - saves models as pickled python objects, with a .pkl file extension
- H2O
 - allows to convert the models built to either POJO (Plain Old Java Object) or MOJO (Model Object, Optimized)
- SparkML
 - models can be saved in the MLeap file format and served in real-time using an MLeap model server
 - supports Spark, Scikit-learn, and Tensorflow for training pipelines and exporting them to an MLeap Bundle
 - The MLeap runtime is a JAR that can run in any Java application
- TensorFlow
 - saves models as .pb files; which is the protocol buffer files extension
- PyTorch
 - serves models by using their proprietary Torch Script as a .pt file
 - model format can be served from a C– application

Vendor-specific Exchange formats(2)

- Keras
 - saves a model as a .h5 file, which is known in the scientific community as a data file saved in the Hierarchical Data Format (HDF)
 - file contains multidimensional arrays of data
- Apple
 - has its proprietary file format with the extension .mlmodel to store models embedded in iOS applications
 - The Core ML framework has native support for Objective-C and Swift programming languages
 - Applications trained in other ML frameworks, such as TensorFlow, Scikit-Learn, and other frameworks need to use tools like coremltools and Tensorflow converter to translate their ML model files to the .mlmodel format for use on iOS

Summarizes of all ML model serialization formats

	Open-Format	Vendor	File Extension	License	ML Tools & Platforms Support	Human-readable	Compression
"almagi-nation"	-	-	-	-	-	-	✓
PMMI	✓	DMG	.pmml	AGPL	R, Python, Spark	✓ (XML)	X
PFA	✓	DMG	JSON		PFA-en-abled runtime	✓ (JSON)	X
ONNX	✓	SIG	.onnx		TF, CNTK, ONNX	-	✓

Summarizes of all ML model serialization formats(2)

	Open-Format	Vendor	File Extension	License	ML Tools & Platforms Support	Human-readable	Compression
TF Serving Format	✓	Google	.pb		TensorFlow	✗	g-zip
Pickle Format	✓		.pkl		scikit-learn	✗	g-zip
JAR/POJO	✓		.jar		H2O	✗	✓
HDF	✓		.h5		Keras	✗	✓
MLEAP	✓		.jar/ .zip		Spark, TF, scikit-learn	✗	g-zip
Torch Script	✗		.pt		PyTorch	✗	✓
Apple .mlmodel	✗	Apple	.mlmodel		TensorFlow, scikit-learn,	—	✓



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Model Deployment

Pravin Y Pawar

Adapted from “Designing Machine Learning Systems” and “Introducing MLOps”

Model Deployment

- “Deploy” is a loose term that generally means making model running and accessible
- To be deployed, model will have to leave the development environment
 - During model development, model usually runs in a development environment
 - can be deployed to a staging environment for testing
 - can be deployed 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 in the first scenario, production environment is similar to the development environment
- **If your work is closer to the second scenario, welcome to world of model deployment!**

Deployment is easy!

If you ignore all hard parts

- If want to deploy a model for friends to play with, all needs to be done is to
 - wrap predict function in a POST request endpoint using Flask or FastAPI,
 - put the dependencies this predict function needs to run in a container
 - push model and its associated container to a cloud service like AWS or GCP to expose the endpoint
- Can use this exposed endpoint for downstream applications:
 - e.g., when an application receives a prediction request from a user,
 - this request is sent to the exposed endpoint, which returns a prediction
- **If familiar with the necessary tools, can have a functional deployment in an hour!**

Machine Learning Deployment Myths

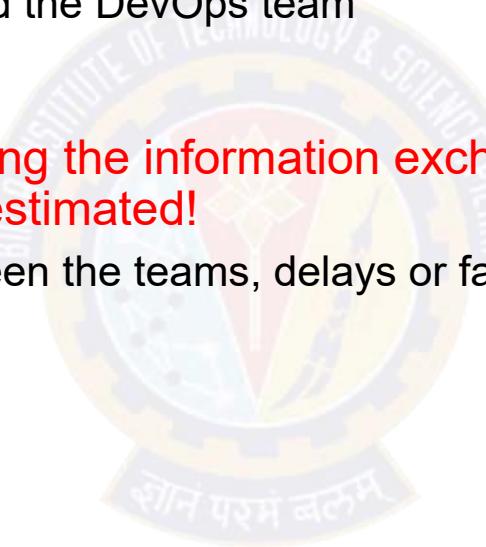
- Deploying an ML model can be very different from deploying a traditional software program
- This difference might cause people who have never deployed a model before
 - to either dread the process
 - or underestimate how much time and effort it will take
- Common myths about the deployment process
 - Myth 1: You Only Deploy One or Two ML Models at a Time
 - Myth 2: If We Don't Do Anything, Model Performance Remains the Same
 - Myth 3: You Won't Need to Update Your Models as Much
 - Myth 4: Most ML Engineers Don't Need to Worry About Scale

Deployment : Reality

- The hard parts include
 - making 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
 - 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
- In many other companies, once
 - a model is ready to be deployed, it will be exported and handed off to another team to deploy it
- However, this separation of responsibilities can cause high overhead communications across teams and make it slow to update model.
 - It also can make it hard to debug should something go wrong.

Productionalization and Deployment

- A key component of MLOps
 - presents an entirely different set of technical challenges than developing the model
 - 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

what exactly is going into production, and what does a model consist of?

- 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
 - (the means from which the API can access the resources it needs to perform the task)
 - that responds to requests in real time
- **Embedded model**
 - Here the model is packaged into an application, which is then published
 - A common example is an application that provides batch-scoring of requests
- What to-be-deployed models consist of depends, of course, on the technology chosen,
 - but typically they comprise a **set of code** (commonly Python, R, or Java) and **data artifacts**
 - 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

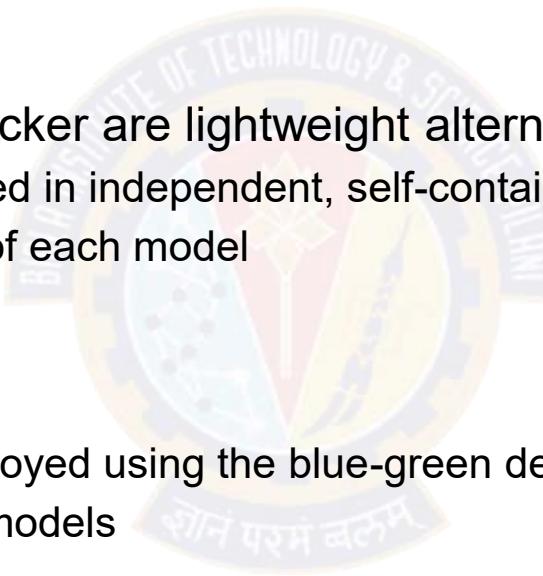
Model Export

- Can export the model to a portable format such as PMML, PFA, ONNX, or POJO
 - improves model portability between systems and simplify deployment
 - helps in reducing dependencies on the production environment
- Come at a cost
 - each format supports a limited range of algorithms,
 - sometimes the portable models behave in subtly different ways than the original
- Whether or not to use a portable format is a choice to be made based on a thorough understanding of the technological and business context.

Model Deployment: Dependency Management(2)

Containerization

- 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
- Also enable
 - new models to be seamlessly deployed using the blue-green deployment technique
 - scaling of compute resources for models
- Orchestrating many containers is the role of technologies such as Kubernetes
 - can be used both in the cloud and on-premise.



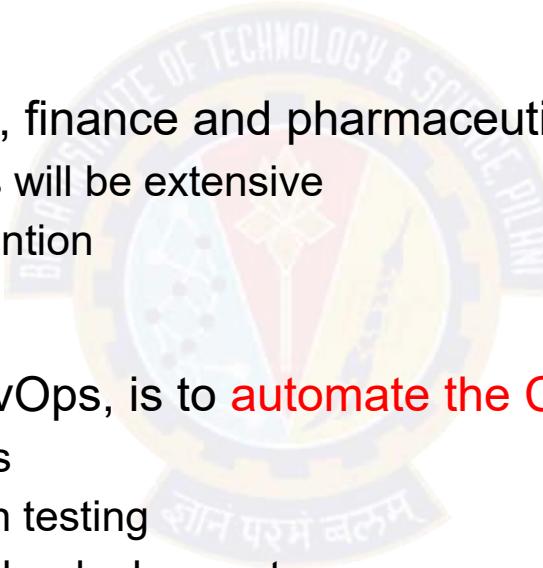
Model Deployment Requirements

What needs to be addressed – when moving from development to the production

- In customer-facing, mission-critical use cases, a more robust CI/CD pipeline is required
- Typically involves:
 - 1) Ensuring all coding, documentation and sign-off standards have been met
 - 2) Re-creating the model in something approaching the production environment
 - 3) Revalidating the model accuracy
 - 4) Performing explainability checks
 - 5) Ensuring all governance requirements have been met
 - 6) Checking the quality of any data artifacts
 - 7) Testing resource usage under load
 - 8) Embedding into a more complex application, including integration tests

Model Deployment Requirements(2)

- One thing is for sure is needed:
 - 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
 - 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
 - speeds up the deployment process
 - enables more extensive regression testing
 - reduces the likelihood of errors in the deployment





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

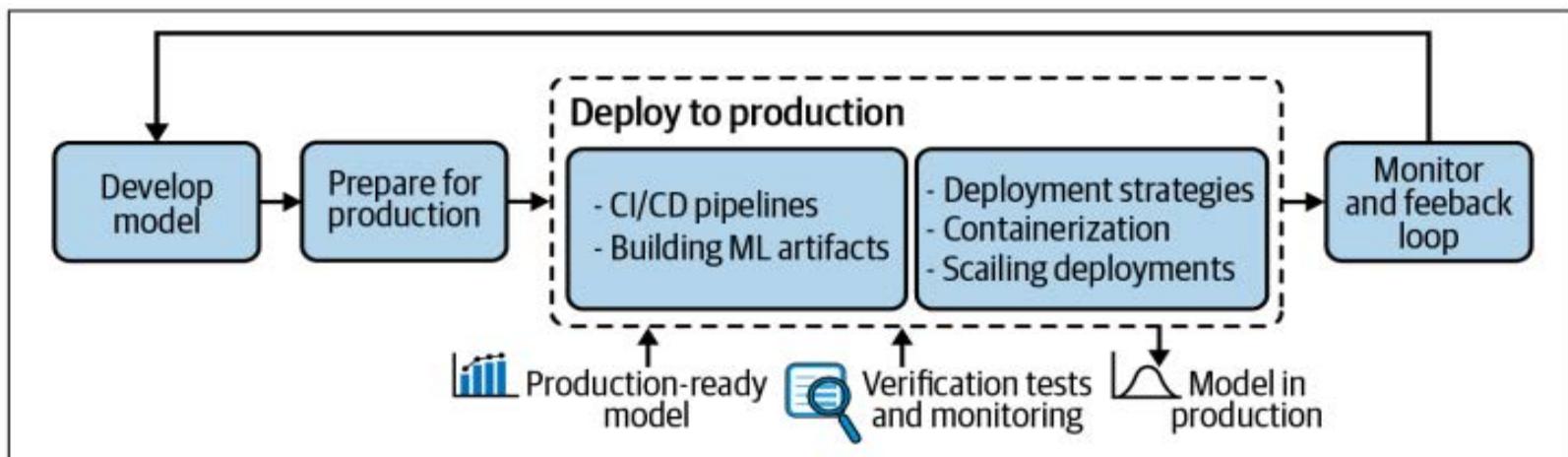
Deploying to Production

Pravin Y Pawar

Adapted from “Introducing MLOps”
By Mark Treveil and Dataiku Team

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
- Let's dive into the concepts and considerations when deploying machine learning models to production
 - that impact and drive—the way MLOps deployment processes are built



Deployment to production highlighted in the larger context of the ML project life cycle

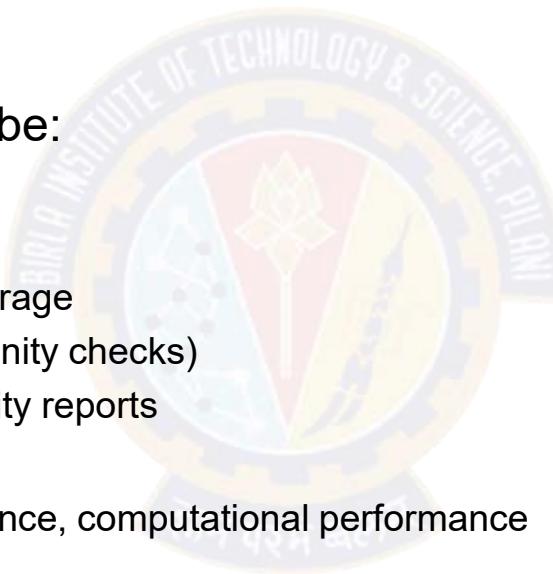
CI/CD Pipelines

A common acronym for continuous integration and continuous delivery (or put more simply, deployment)

- 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.

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



CI/CD for ML(2)

- 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

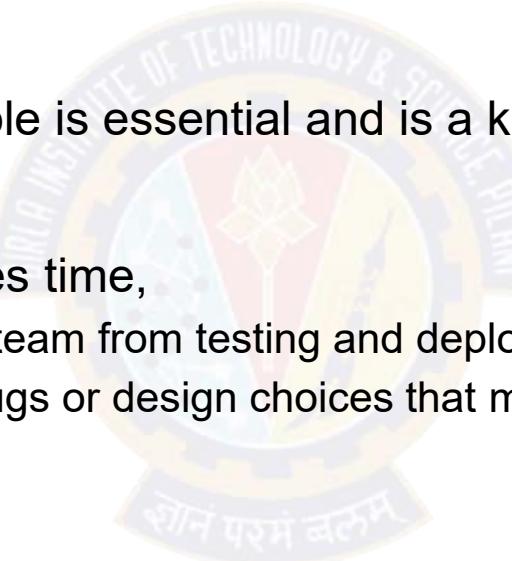
- The goal of a continuous integration pipeline is
 - to avoid unnecessary effort in merging the work from several contributors
 - also to detect bugs or development conflicts as soon as possible
- The very first step is using centralized version control systems
 - unfortunately, working for weeks on 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

Two ways to approach model deployment

- 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

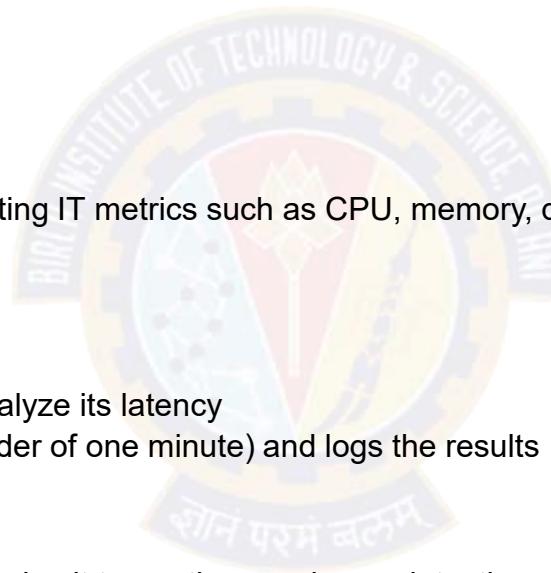
Considerations When Sending Models to Production

- 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

Once a model is released, it must be maintained

- 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





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

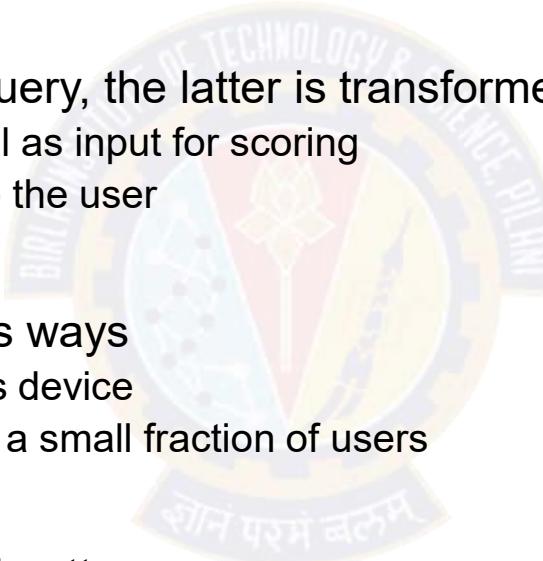
Deployment Patterns

Pravin Y Pawar

Adapted from “Machine Learning Engineering”
By Andriy Burkov

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.

Static Deployment(2)

Pros and Cons

- Many advantages:
 - software has direct access to the model, so the execution time is fast for the user
 - user data doesn't have to be uploaded to the server at the time of prediction
 - saves time and preserves privacy
 - model can be called when the user is offline
 - software vendor doesn't have to care about keeping the model operational
 - becomes the user's responsibility
- Several drawbacks:
 - The separation of concerns between the machine learning code and the application code isn't always obvious
 - makes it harder to upgrade the model without also having to upgrade the entire application
 - If the model has certain computational requirements for scoring (such as access to an accelerator or a GPU)
 - may add complexity and confusion as to where the static deployment can or cannot be used

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

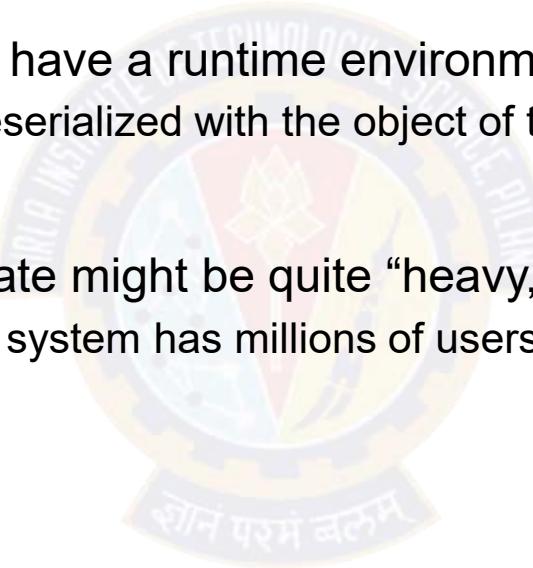
Deployment of Model Parameters

- In this deployment scenario, the model file only contains the learned parameters
 - user's device has installed a runtime environment for the model
- Some machine learning packages, like TensorFlow,
 - have a lightweight version that can run on mobile devices
- Alternatively, frameworks such as Apple's Core ML allow running models on Apple devices
 - created using popular packages, including scikit-learn, Keras, and XGBoost



Deployment of a Serialized Object

- Model file is a serialized object that the application would deserialize
- The advantage is that don't need to have a runtime environment for model on the user's device
 - all needed dependencies will be deserialized with the object of the model
- An evident drawback is that an update might be quite "heavy,"
 - which is a problem if your software system has millions of users



Deploying to Browser

- Most modern devices have access to a browser, either desktop or mobile
- Some machine learning frameworks, such as TensorFlow.js,
 - have versions that allow to train and run a model in a browser, by using JavaScript as a runtime
- Even possible to train a TensorFlow model in Python,
 - then deploy it to, and run it in the browser's JavaScript runtime environment
 - if a GPU (graphics processing unit) is available on the client's device, Tensorflow.js can leverage it

Dynamic Deployment on User's Device(2)

Advantages and Drawbacks

- Advantages
 - Calls to the model will be fast for the user
 - Reduces the impact on the organization's servers, as most computations are performed on the user's device
- If the model is deployed to the browser,
 - advantage is organization's infrastructure only needs to serve a web page that includes the model's parameters
 - A downside is bandwidth cost and application startup time might increase.
 - users must download the model's parameters each time they start the web application
 - as opposed to doing it only once when they install an application

Dynamic Deployment on User's Device(3)

Advantages and Drawbacks

- Monitoring
 - Deploying to a user's device makes it difficult to monitor the model performance
- Model updates
 - A serialized object can be quite voluminous
 - Some users may be offline during the update, or even turn off all future updates
 - may end up with different users using very different model versions
 - becomes difficult to upgrade the server-side part of the application
- Third-party analyses
 - Deploying models on the user's device means that the model easily becomes available for third-party analyses
 - may try to reverse-engineer the model to reproduce its behavior
 - may search for weaknesses by providing various inputs and observing the output
 - may adapt their data so the model predicts what they want

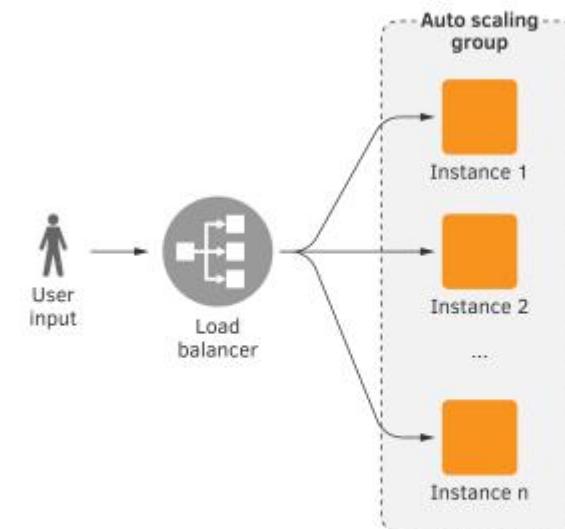
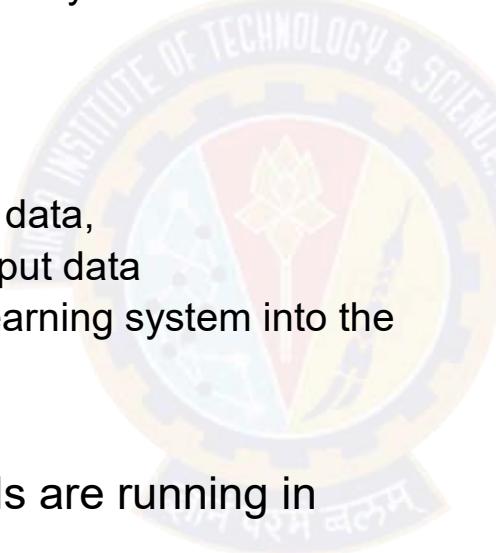
Dynamic Deployment on a Server

- 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
- Four common practices
 - Deployment on a Virtual Machine
 - Deployment in a Container
 - Serverless Deployment
 - Model Streaming



Deployment on a Virtual Machine(VM)

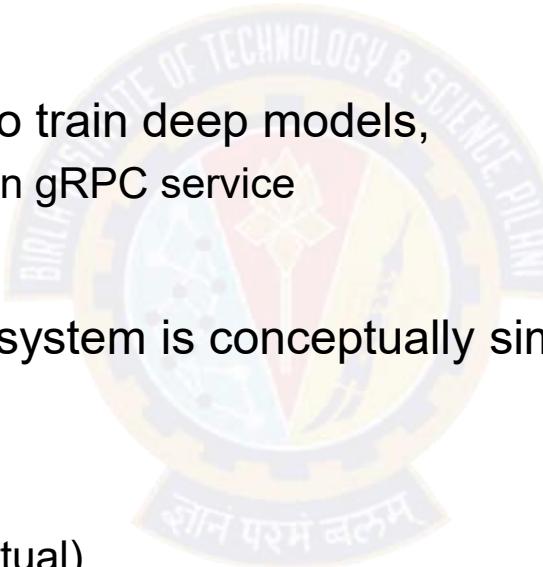
- In a typical web service architecture deployed in a cloud environment
 - predictions are served in response to canonically-formatted HTTP requests
- A web service running on a virtual machine
 - receives a user request containing the input data,
 - calls the machine learning system on that input data
 - then transforms the output of the machine learning system into the output JSON or XML string
- To cope with high load, several identical VMs are running in parallel
 - A load balancer dispatches the incoming requests to a specific virtual machine
 - VMs can be added and closed manually, or be a part of an autoscaling group that launches
 - VMs can be terminated virtual machines based on their usage



: Deploying a machine learning model as a web service on a virtual machine.

Deployment on a Virtual Machine(VM)2

- In Python, a REST API web service is usually implemented
 - using a web application framework such as Flask or FastAPI
- TensorFlow, a popular framework used to train deep models,
 - comes with TensorFlow Serving, a built-in gRPC service
- Advantage: Architecture of the software system is conceptually simple: a typical web or gRPC service
- Downsides
 - Need to maintain servers (physical or virtual)
 - If virtualization is used, there is an additional computational overhead due to virtualization and running multiple OS
 - Network latency, which can be a serious issue, depending on how fast you need to process scoring results
 - has a relatively higher cost, compared to deployment in a container, or a serverless deployment



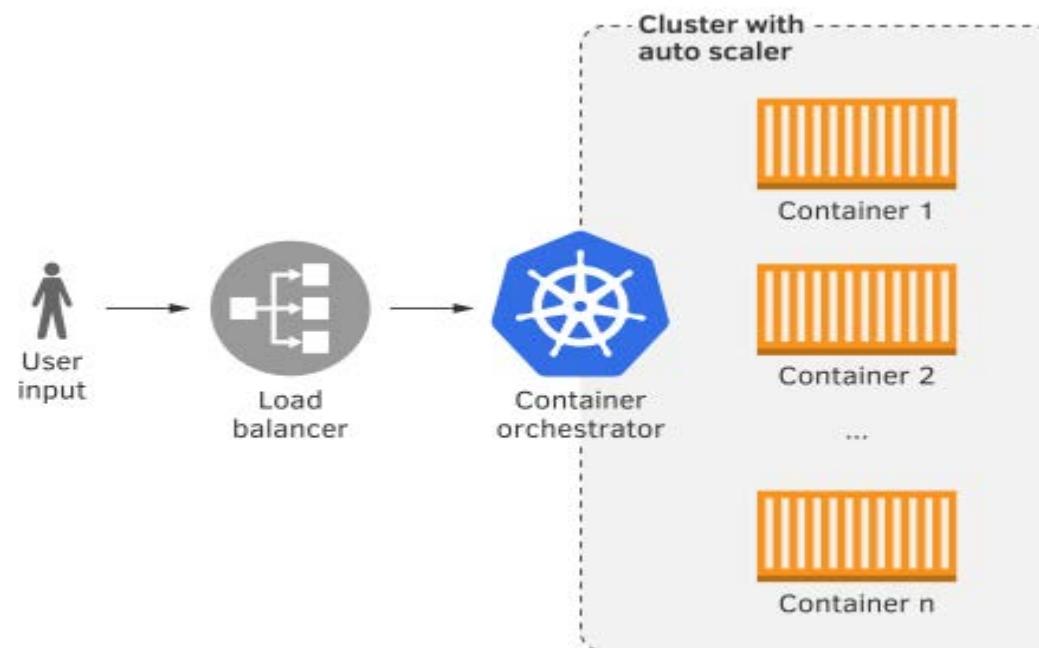
Deployment in a Container

- 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
- Deployment Process
 - The machine learning system and the web service are installed inside a container
 - Usually, a container is a Docker container, but there are alternatives
 - Then a container-orchestration system is used to run the containers on a cluster of physical or virtual servers
 - A typical choice of a container-orchestration system for running on-premises or in a cloud platform, is Kubernetes
 - Some cloud platforms provide both their own container-orchestration engine, such as AWS Fargate and Google Kubernetes Engine

Deployment in a Container(2)

Organization

- Virtual or physical machines are organized into a cluster,
 - whose resources are managed by the container orchestrator
- New virtual or physical machines can be manually added to the cluster, or closed
- If your software is deployed in a cloud environment,
 - a cluster autoscaler can launch (and add to the cluster) or terminate virtual machines
 - based on the usage of the cluster.



Deploying a model as a web service in a container running on a cluster.

Deployment in a Container(3)

- Advantage
 - More resource-efficient as compared to the deployment on a virtual machine
 - Allows the possibility to automatically scale with scoring requests
 - Allows us to scale-to-zero- reduced down to zero replicas when idle and brought back up if there is a request to serve
 - the resource consumption is low compared to always running services
 - leads to less power consumption and saves cost of cloud resources
- Drawback
 - Containerized deployment is generally seen as more complicated, and requires expertise

Serverless Deployment

- Several cloud services providers, including Amazon, Google, and Microsoft, offers 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
- The cloud platform provides an API to submit inputs to the serverless function
 - specifies its name, provides the payload, and yields the outputs
- The cloud platform takes care of
 - deploying the code and the model on an adequate computational resource,
 - executing the code,
 - routing the output back to the client

Serverless Deployment(2)

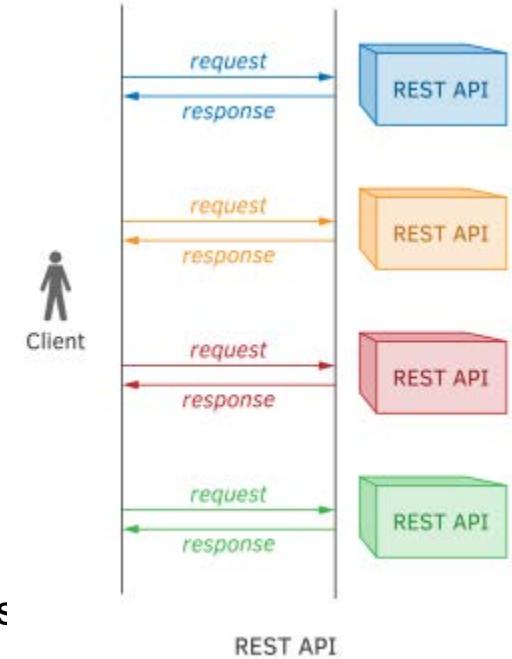
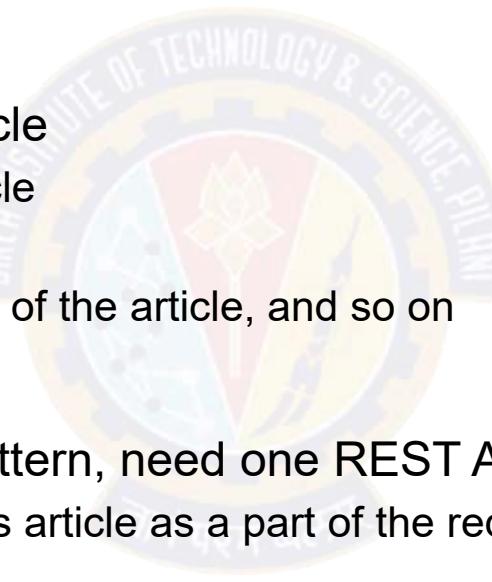
Advantages and Limitations

- Advantages to relying on serverless deployment
 - don't have to provision resources such as servers or virtual machines
 - don't have to install dependencies, maintain, or upgrade the system
 - highly scalable and can easily and effortlessly support thousands of requests per second
 - support both synchronous and asynchronous modes of operation
 - cost-efficient: only pay for compute-time
 - simplifies canary deployment, or canarying
 - Rollbacks are also very simple in the serverless deployment because it is easy to switch back to the previous version of the function

- Limitations
 - Restrictions by the cloud service provider
 - the function's execution time, zip file size, and amount of RAM available on the runtime
 - Zip file size limit can be a challenge - A typical model requires multiple heavyweight dependencies
 - Unavailability of GPU access can be a significant limitation for deploying deep models

Model Serving – REST API Revisited

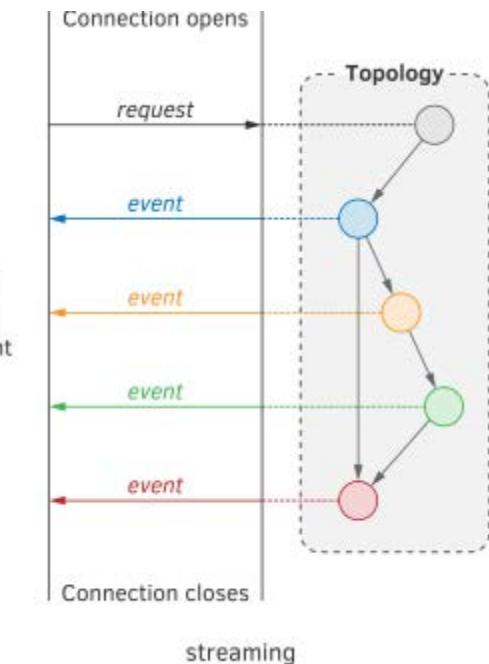
- In complex systems, there can be many models applied to the same input
 - a model can input a prediction from another model
- For example, the input may be a news article
 - One model can predict the topic of the article
 - Another model can extract named entities
 - Third model can generate a summarization of the article, and so on
- According to the REST API deployment pattern, need one REST API per model
 - client would call one API by sending a news article as a part of the request - get the topic as response
 - client calls another API by sending a news article, and gets the named entities as response; etc.



Model Streaming

Can be seen as an inverse to the REST API

- 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 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.
- In a stream-processing application, nodes transform their input in some way,
 - then either,
 - send the output to other nodes, or
 - send the output to the client, or
 - persist the output to the database or a filesystem.





Thank You!

In our next session:



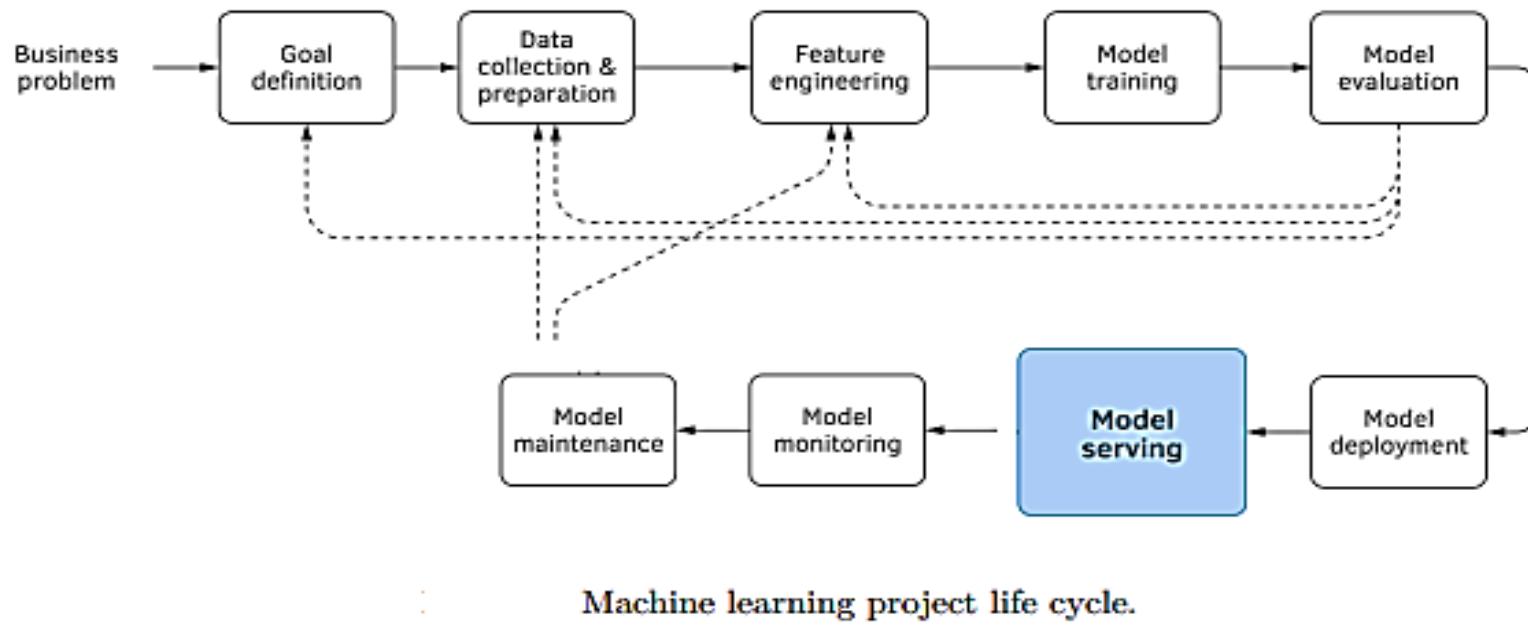
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Essentials of Model Serving

Pravin Y Pawar

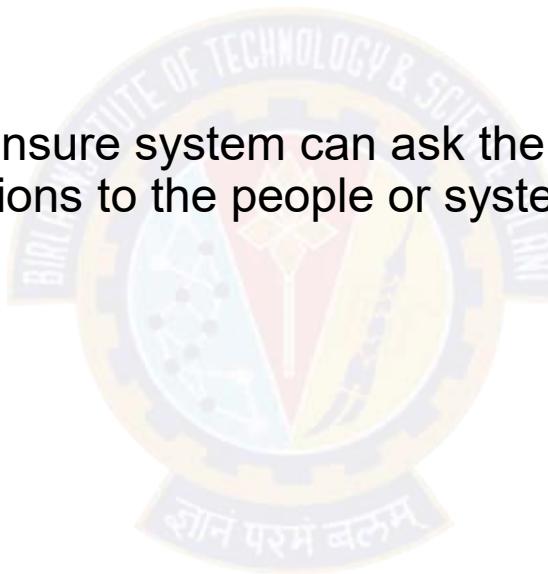
Adapted from “Machine Learning Engineering” and
“Reliable Machine Learning”

Model Serving in ML Lifecycle



Model Serving

- Model built in 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 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

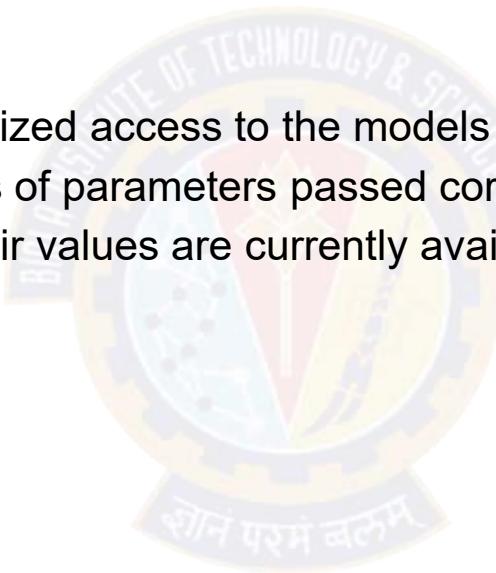
- The model serving runtime is the environment in which the model is applied to the input data
 - The runtime properties are dictated by the model deployment pattern
- However, an effective runtime will have several additional properties such as
 - Security and Correctness
 - Ease of Deployment
 - Guarantees of Model Validity
 - Ease of Recovery
 - Avoidance of Training/Serving Skew
 - Avoidance of Hidden Feedback Loops



Properties of the Model Serving Runtime(2)

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.



Properties of the Model Serving Runtime(3)

Ease of Deployment and Recovery

- 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.

Properties of the Model Serving Runtime(4)

Avoidance of Training/Serving Skew

- When it concerns feature extraction, 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 engineering team may reimplement the feature extractor code for production for many reasons
 - most common being data analyst's code is inefficient or incompatible with the production ecosystem
- Runtime should allow easy access to the feature extraction code for various needs,
 - including model retraining, ad-hoc model calls, and production.
- One way to implement it is by wrapping the feature extraction object into a separate web service
- If cannot avoid using two different codebases to generate features for training and production,
 - then the runtime should allow for the logging of feature values generated in the production environment
 - Those values should then be used as training values

Key questions for model serving

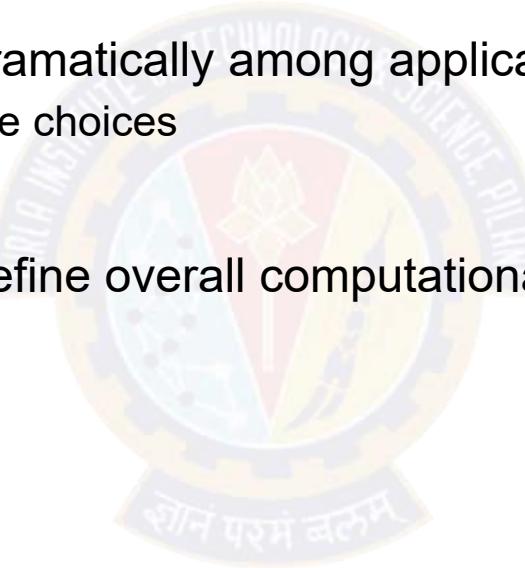
- Lot of ways to create structures around the model that support serving,
- Each with very different sets of trade-offs
- Useful to think through specific questions about needs of system
 - What will be the load to model?
 - What are the prediction latency needs of model?
 - Where does the model need to live?
 - What are the hardware needs for model?
 - How will the serving model be stored, loaded, versioned and updated?
 - What will feature pipeline for serving look like?

What will be the 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 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

What are the prediction latency needs of model?

- Prediction latency is time between the moment request is made and moment response is received
- Acceptable prediction latency can vary dramatically among applications
 - Is major determiner of serving architecture choices
- Taken together, latency and traffic load define overall computational needs of ML system
- If latency is too high, can be mitigated by
 - Using more powerful hardware
 - Making model less expensive to compute
- But creating a larger number of model replicas is not a solution for latency issue!



Where does the model need to live?

Model needs to be stored on physical device in specific location – home of model

- This choice has significant implications on overall serving architecture
- On a local machine
 - Not a practical solution – may be suitable for small batch predictions
 - Not recommended beyond small-scale prototyping or bespoke uses
- On servers owned or managed by our organization
 - Important when specific privacy or security concerns are in place
 - Right option if latency is hypercritical concern or if speciality hardware is needed to run models
 - Can limit flexibility in terms of scaling up/down, require special attention to monitoring
- In the cloud
 - Can allow easy scaling overall computation footprint up or down
 - Can be done by
 - running model servers on own virtual servers and controlling how many of them to be used
 - managed inference service
- On-device
 - Everything from mobile phones to smart watches, digital assistants, automobiles, printers etc.
 - Has strict constraints on model size, because of limited memory and power

What are the 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
 - Suitable for non-deep methods, non deep matrix multiplications
- Hardware accelerators – commonly GPU
 - Choice of serving deep learning models as they involve dense matrix multiplications
 - But has drawbacks
 - Specialized hardware – need to invest or use a cloud service – costly options
 - Not suited for operations not involving large amounts of dense matrix calculations

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

Model is physical object – has size and needs space

- Model serving in offline environment be stored on disk and loaded by specific binaries in batch jobs
 - Main requirement is disk space to store model and I/O capacity to load model from disk and RAM needed to load model into memory for use
- Model used in online serving needs to be stored in RAM in dedicated machine
 - For high-throughput services in latency critical settings, copies of these models likes to be stored and served in many replica machine in parallel
- Eventually these models needs to be updated by retraining
- means needs to swap version of model currently used in serving on a given machine with a new version
- 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?

Feature needs to be processed at serving time as well as at training time

- 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
 - Computational requirements for this may be considerable
- 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
 - Main source of classic training-serving skew and bugs notoriously difficult to detect and debug
- Promise is in 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
 - Means serving feature pipeline is far from an afterthought
 - But is indeed often most production-critical part of the entire serving stack



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

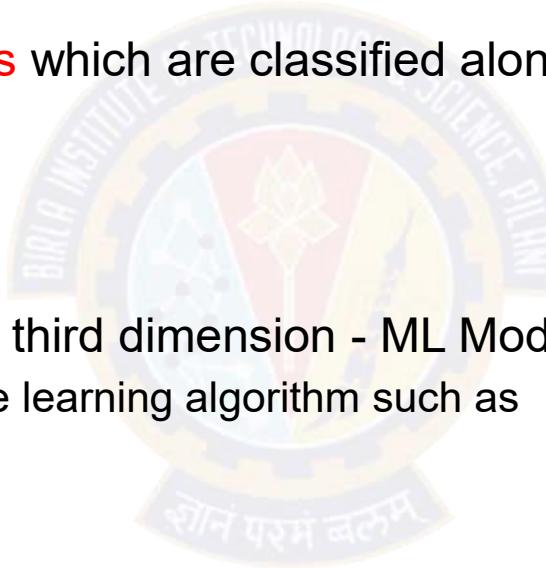
Different forms of ML workflows

Pravin Y Pawar

Extracted from
Three Levels of ML Software

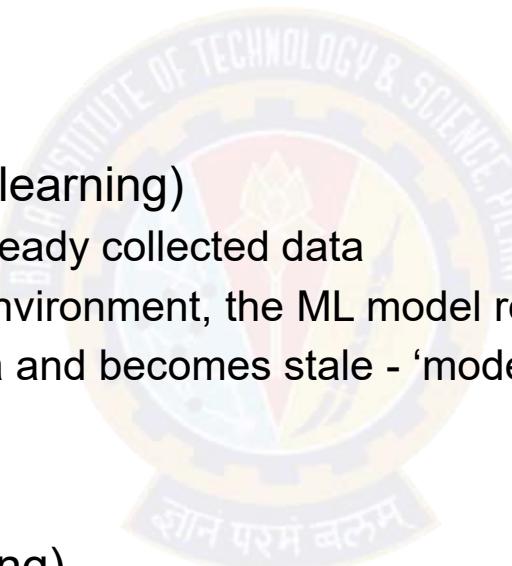
Different forms of ML workflows

- Operating an ML model might assume **several architectural styles**
- Primarily **Four architectural patterns** which are classified along two dimensions:
 - ML Model Training
 - ML Model Prediction
- For 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 re-trained
 - Model will see a lot of real-live data and becomes stale - ‘model decay’ and should be carefully monitored
- Online learning (aka dynamic learning)
 - The model is regularly being 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

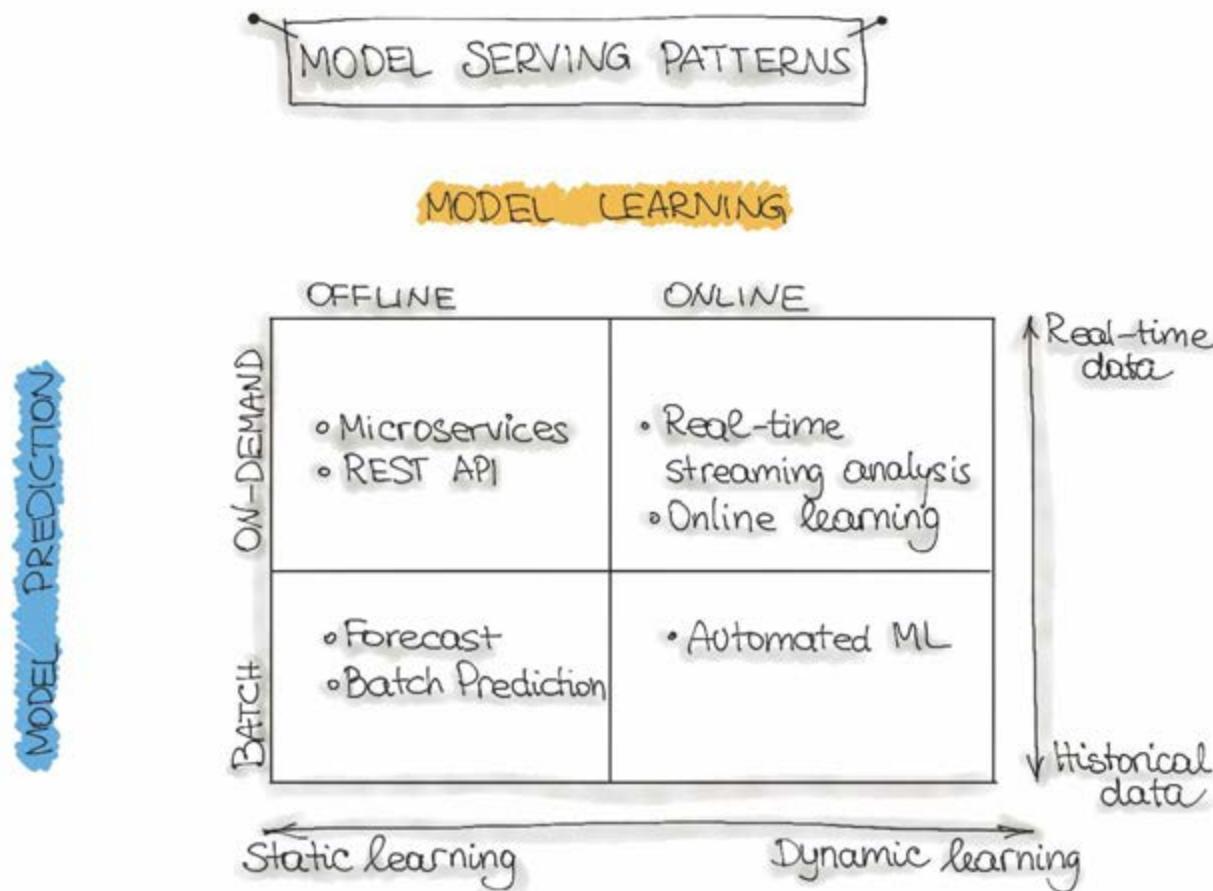


Model Prediction Patterns

- Two modes to denote the mechanics of the ML model to makes 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

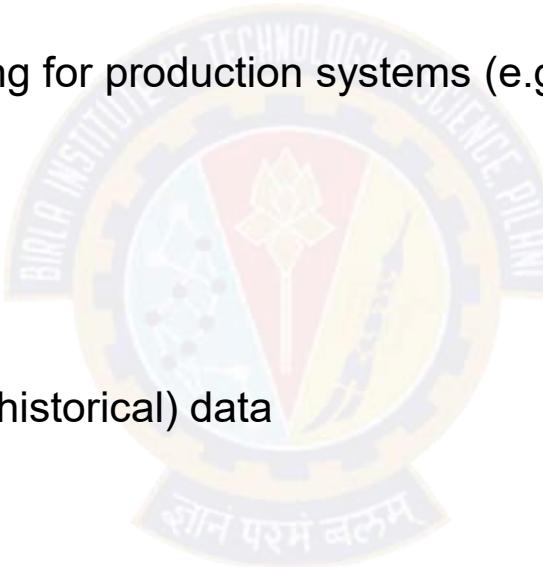
ML architecture patterns

Forecast, Web-Service, Online Learning, and AutoML



Forecast

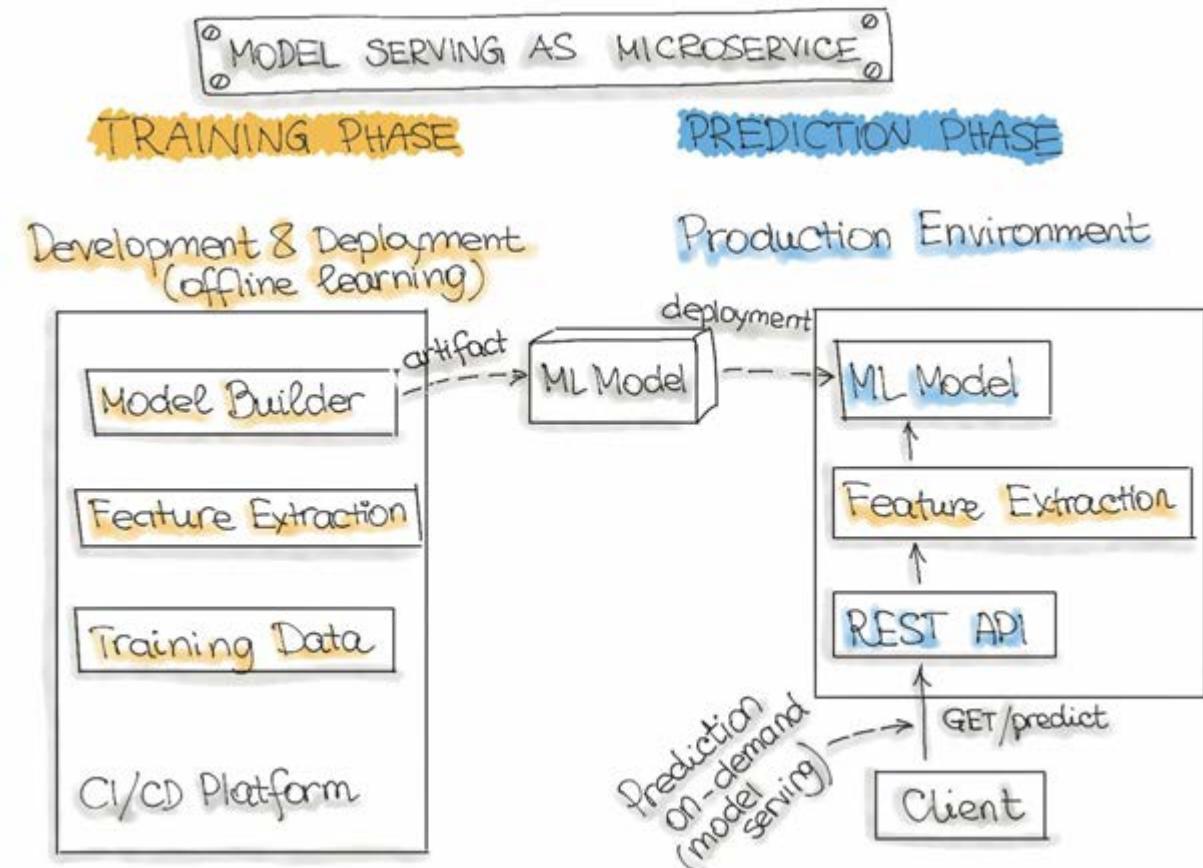
- Widely spread in academic research or data science education (e.g., Kaggle or DataCamp)
 - used to experiment with ML algorithms and data as it is the easiest way to create a machine learning system
 - not very useful in an industry setting for production systems (e.g. mobile applications)
- Usually involves steps
 - take an available dataset
 - train the ML model
 - run this model on another (mostly historical) data
 - makes predictions



Web-Service (or Microservices)

Architecture for wrapping trained models as deployable services

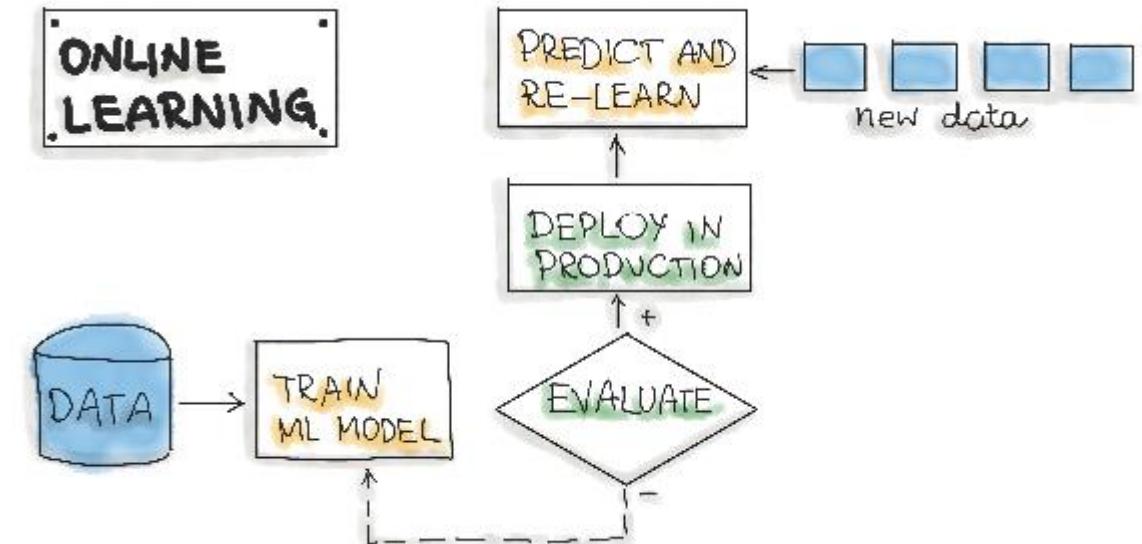
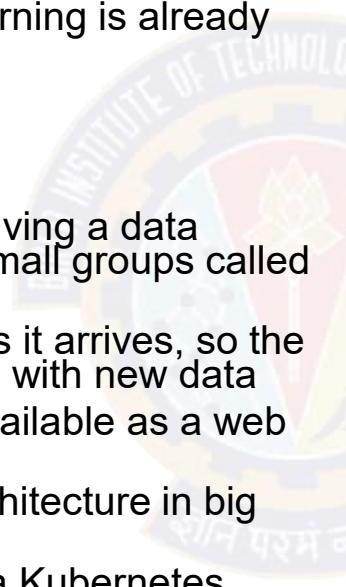
- The most commonly described deployment architecture for ML models
- The web service takes input data and outputs a prediction for the input data points
 - Model is trained offline on historical data, but it uses real-live data to make predictions
 - Model remains constant until it is re-trained and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



Online Learning (Real-time streaming analytics)

Most dynamic way to embed machine learning into a production system

- Confusing name because the core learning or ML model training is usually not performed on the live system
 - call it incremental learning- term online learning is already established within the ML community
- In this type of ML workflow
 - ML learning algorithm is continuously receiving a data stream, either as single data points or in small groups called mini-batches
 - System learns about new data on the fly as it arrives, so the ML model is incrementally being re-trained with new data
 - Continually re-trained model is instantly available as a web service
 - Technically works well with the lambda architecture in big data systems
 - Model would typically run as a service on a Kubernetes cluster or similar
- A big difficulty with the online learning system in production is that
 - if bad data is entering the system, the ML model, as well as the whole system performance, will increasingly decline



AutoML

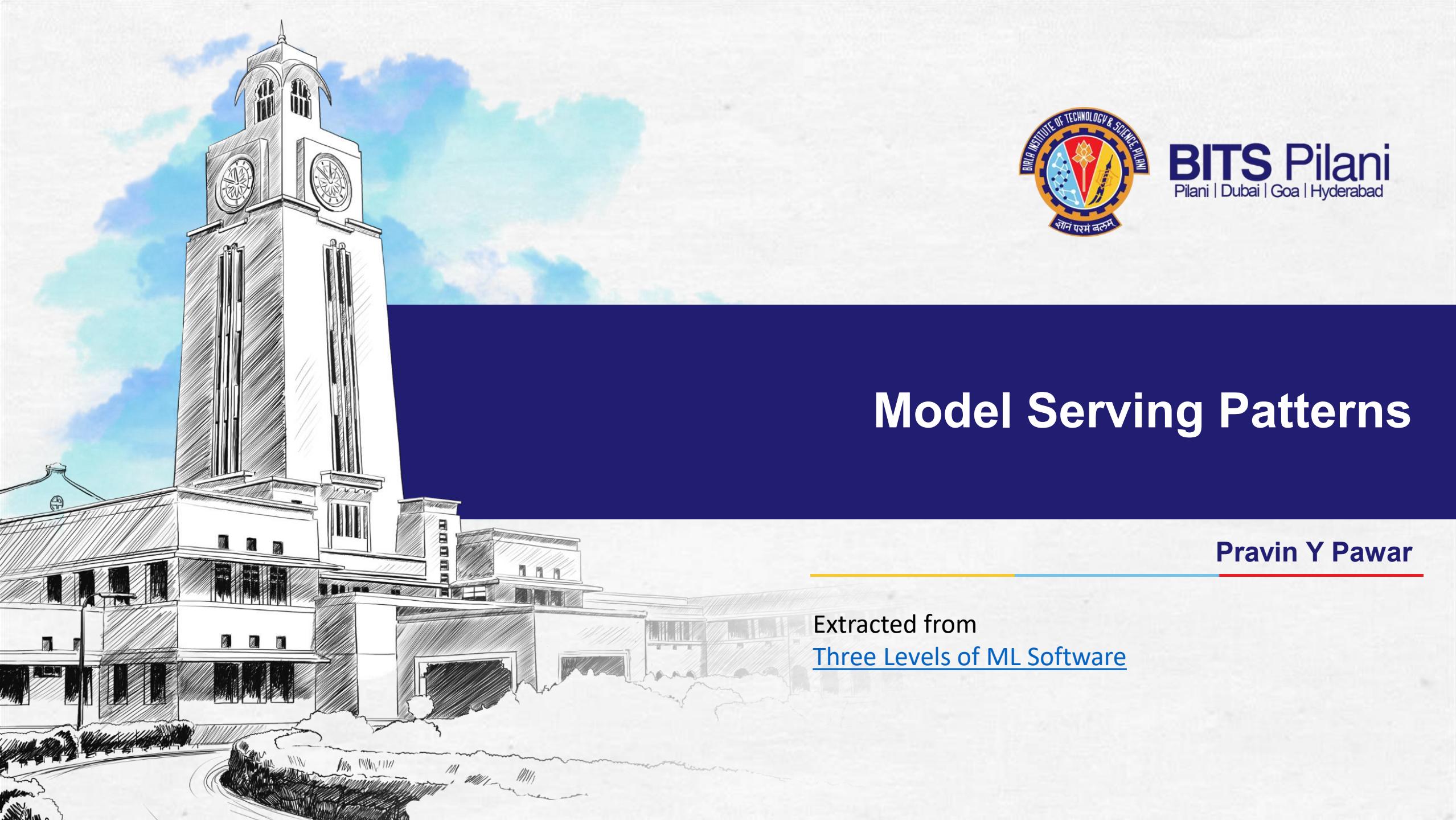
Sophisticated version of online learning

- 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
 - User needs to provide data, and the AutoML system automatically selects an ML algorithm and configures the selected algorithm
 - very experimental way to implement ML workflows
 - usually provided by big cloud providers, such as Google or MS Azure
 - Instead of updating the model, need to execute an entire ML model training pipeline in production that results in new models on the fly



Thank You!

In our next session:



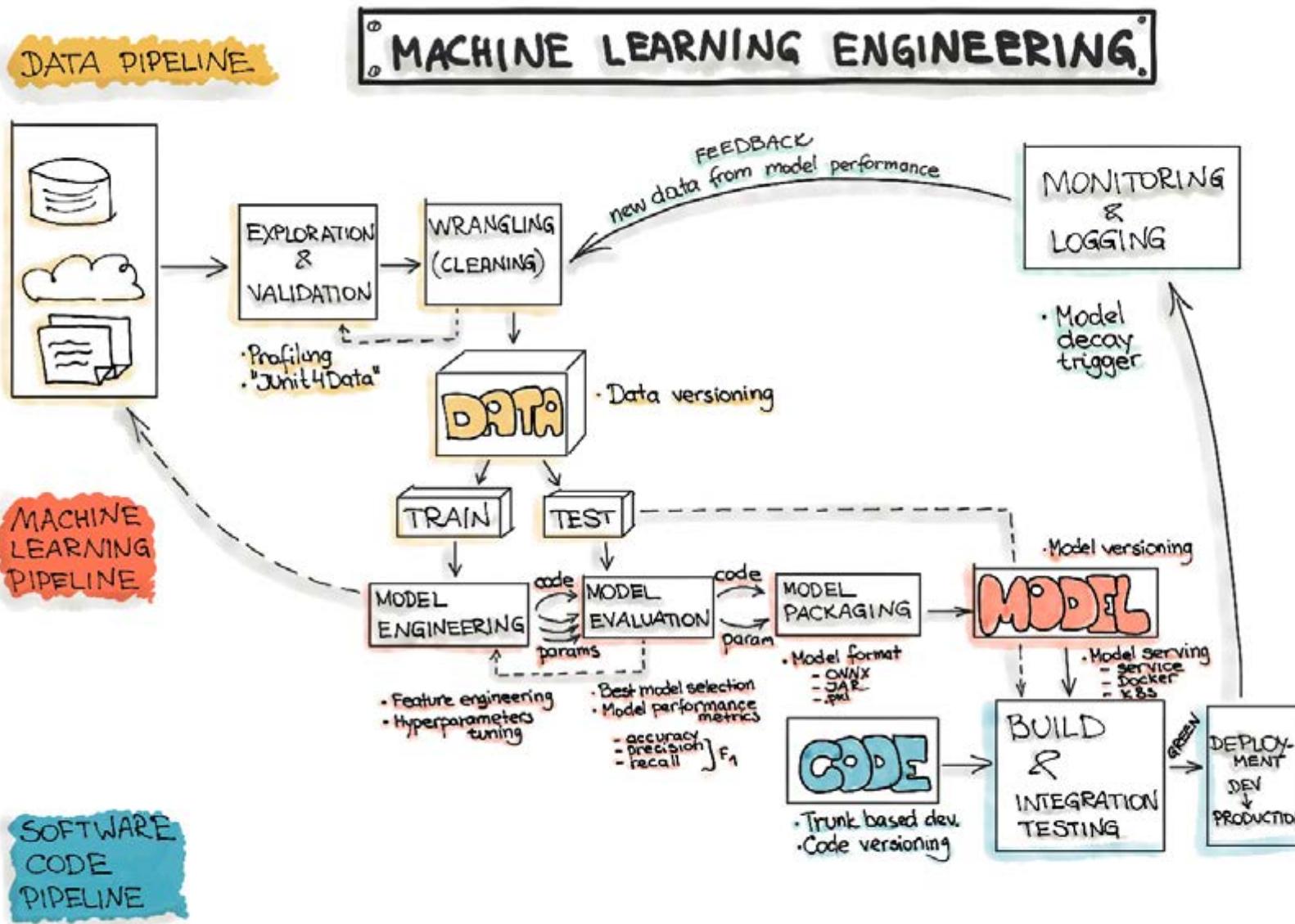
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Model Serving Patterns

Pravin Y Pawar

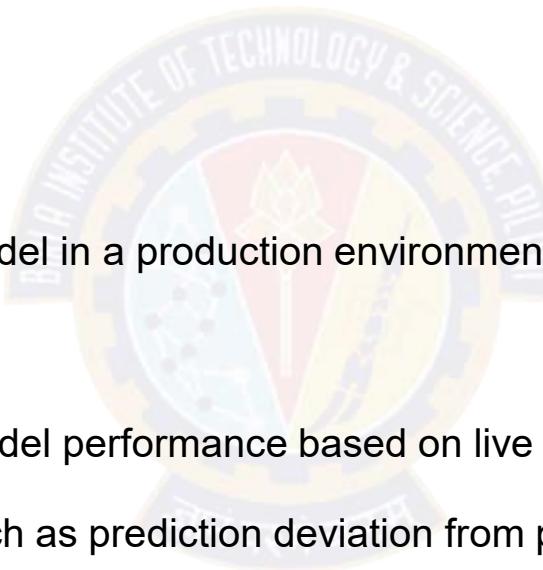
Extracted from
Three Levels of ML Software

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
 - 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 a log-record.



Model Serving Patterns

- Three components should be considered when ML model is served in a production environment
 - The inference is the process of getting data to be ingested by a model to compute predictions
 - requires a model, an interpreter for the execution, and input data
- Deploying an ML system to a production environment includes two aspects,
 - First deploying the pipeline for automated retraining and ML model deployment
 - Second, providing the API for prediction on unseen data
- Model serving is a way to integrate the ML model in a software system
- Five patterns to put the ML model in production:
 - Model-as-Service
 - Model-as-Dependency
 - Precompute
 - Model-on-Demand
 - and Hybrid-Serving

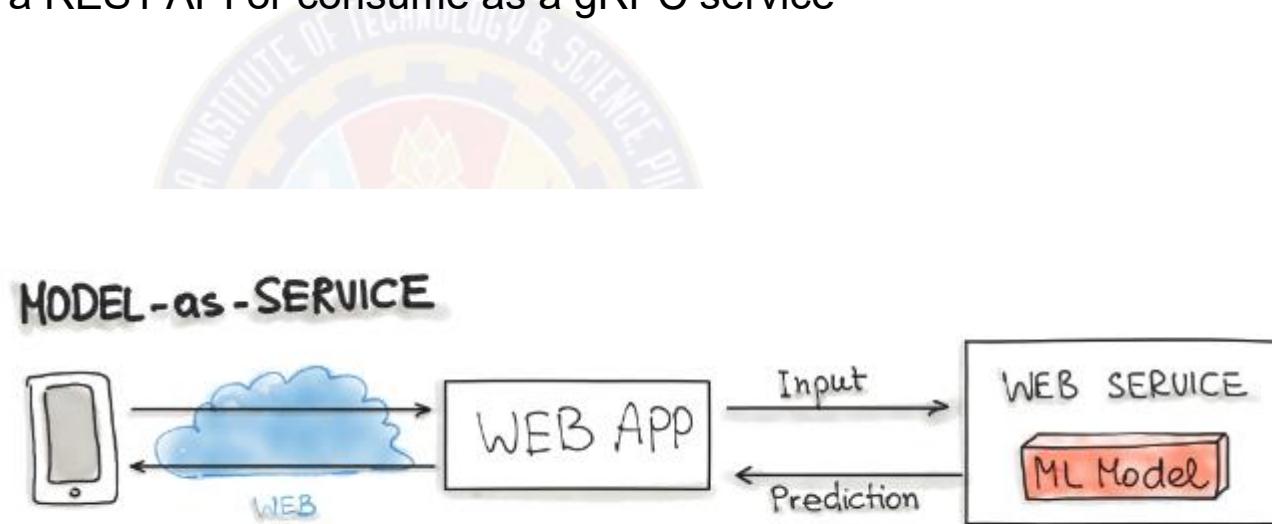
Model Serving Patterns(2)

Machine Learning Model Serving Taxonomy

Machine Learning Model Serving Taxonomy			
	ML Model		
Service & Versioning	Together with the consuming application	Independent from the consuming application	
Compile/ Runtime Availability	Build & runtime available	Available remotely through REST API/RPC	Available at the runtime scope
Serving Patterns	"Model-as-Dependency"	"Model-as-Service"	"Precompute" and "Model on Demand"
	Hybrid Model Serving (Federated Learning)		

Model-as-Service

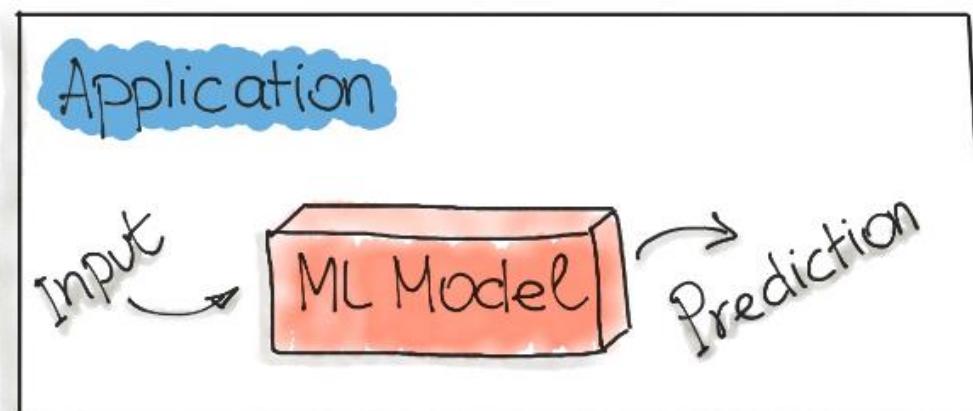
- A common pattern for wrapping an ML model as an independent service
 - can wrap the ML model and the interpreter within a dedicated web service
 - applications can request through a REST API or consume as a gRPC service
- Used for various ML workflows
 - Forecast
 - Web Service
 - Online Learning



Model-as-Dependency

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

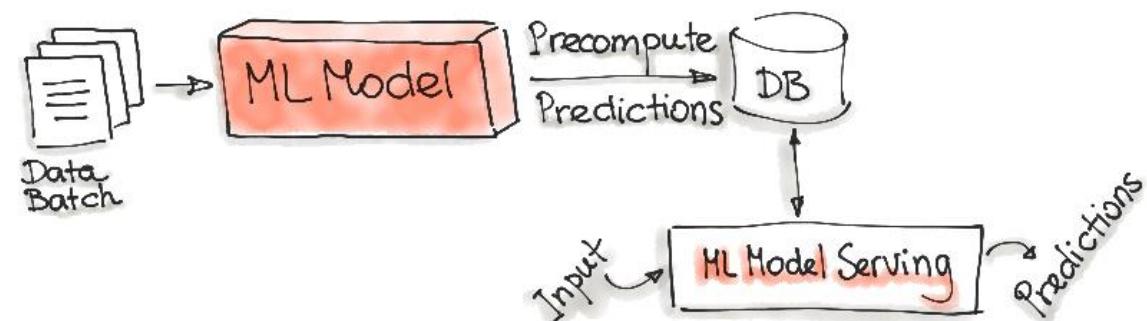
MODEL -as- DEPENDENCY



Precompute Serving Pattern

- Tightly related to the Forecast ML workflow
- Use an already trained ML model and precompute the predictions for the incoming batch of data
 - Resulting predictions are persisted in the database
 - For any input request, query the database to get the prediction result

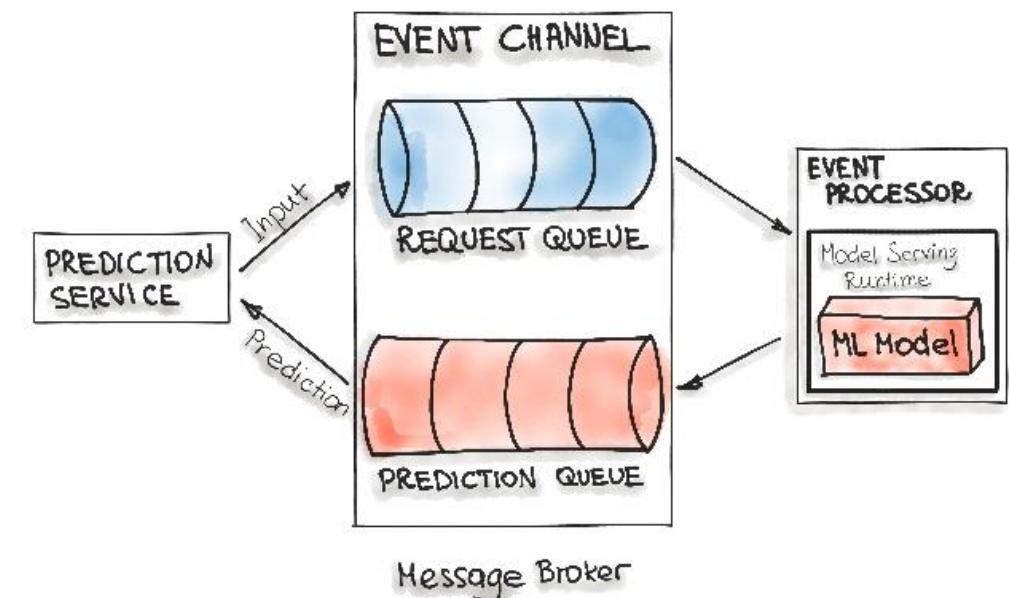
PRECOMPUTE SERVING PATTERN



Model-on-Demand

- Treats the ML model as a dependency that is available at runtime
 - contrary to the Model-as-Dependency pattern, has its own release cycle and is published independently
- Message-broker architecture is typically used for such on-demand model serving
 - contains two main types of architecture components:
 - a broker component
 - an event processor component
 - Broker component is the central part that contains the event channel that are utilised within the event flow
 - The event channels, which are enclosed in the broker component, are message queues
 - Message broker allows one process to write prediction-requests in a input queue
 - Event processor contains the model serving runtime and the ML model
 - connects to the broker, reads these requests in batch from the queue and sends them to the model to make the predictions
- The model serving process runs the prediction generation on the input data
 - writes the resulted predictions to the output queue
 - queued prediction results are pushed to the prediction service that initiated the prediction request

MODEL-ON-DEMAND

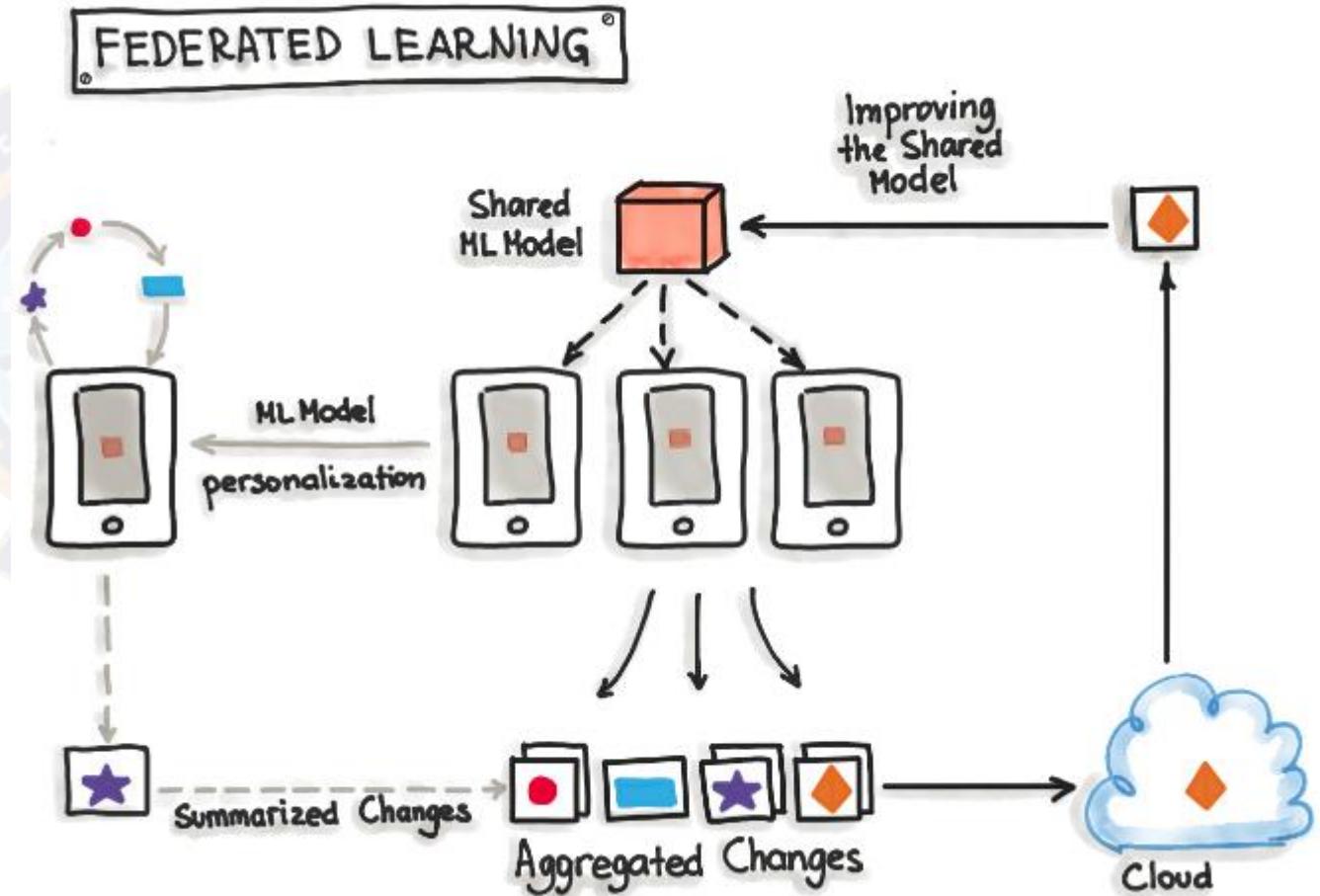


Hybrid-Serving (Federated Learning)

- Unique in the way it does, there is not only one model that predicts the outcome, but there are also lots of it
 - Exactly spoken there are as many models as users exist, in addition to the one that's held on a server
- Start with the unique model, the one on the server
 - Model on the server-side is trained only once with the real-world data
 - sets the initial model for each user
 - a relatively general trained model so it fits for the majority of users
- On the other side, there are the user-side models - really unique models
 - possible for the devices to train their own models due to hardware enhancements
 - devices will train their own highly specialized model for their own user
 - Once in a while, the devices send their already trained model data (not the personal data) to the server
- Then the server model will be adjusted
 - the actual trends of the whole user community will be covered by the model
 - this updated server model is set to be the new initial model that all devices are using

Hybrid-Serving (Federated Learning) 2

- Not having any downsides for the users, while the server model gets updated, this happens only when the device is idle, connected to WiFi and charging
 - testing is done on the devices, therefore the newly adopted model from the server is sent to the devices and tested for functionality
- Big benefit
 - data used for training and testing, which is highly personal, never leaves the devices while still capturing all data that is available
 - possible to train highly accurate models while not having to store tons of (probably personal) data in the cloud
- Constraints
 - mobile devices are less powerful
 - the training data is distributed across millions of devices and these are not always available for training
 - Exactly for this TensorFlow Federated (TFF) has been created - lightweight form of TensorFlow



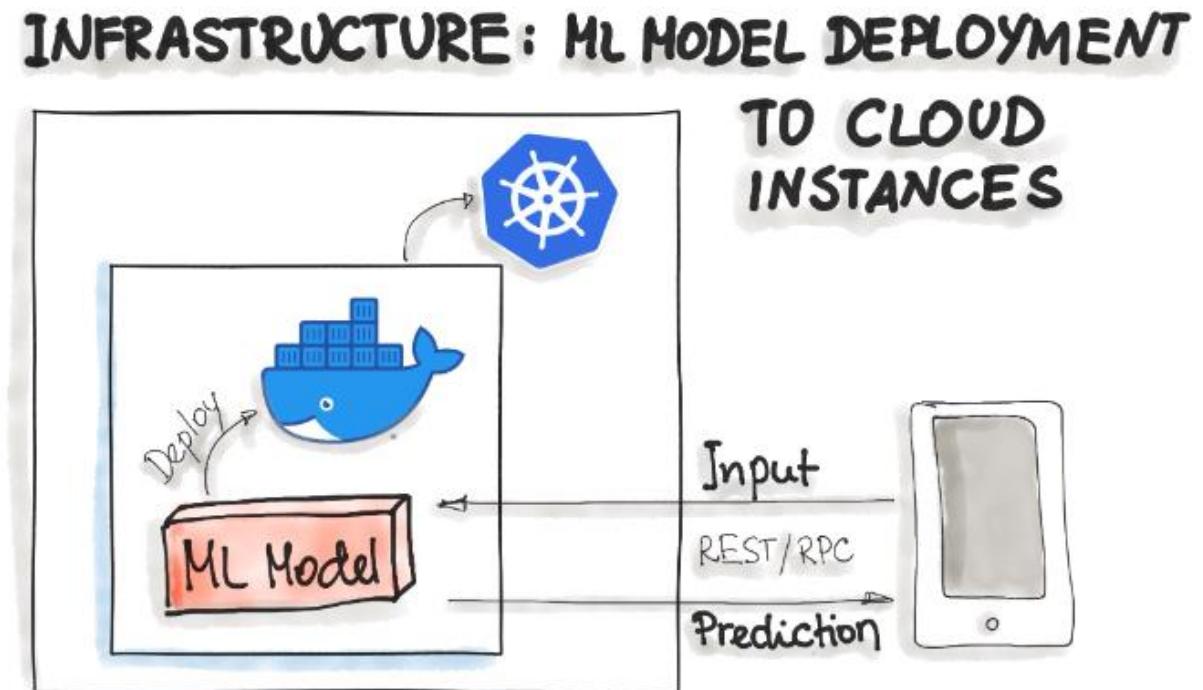
Deployment Strategies

- Common ways for wrapping trained models as deployable services, namely deploying ML models as
 - Docker Containers to Cloud Instances
 - Serverless Functions



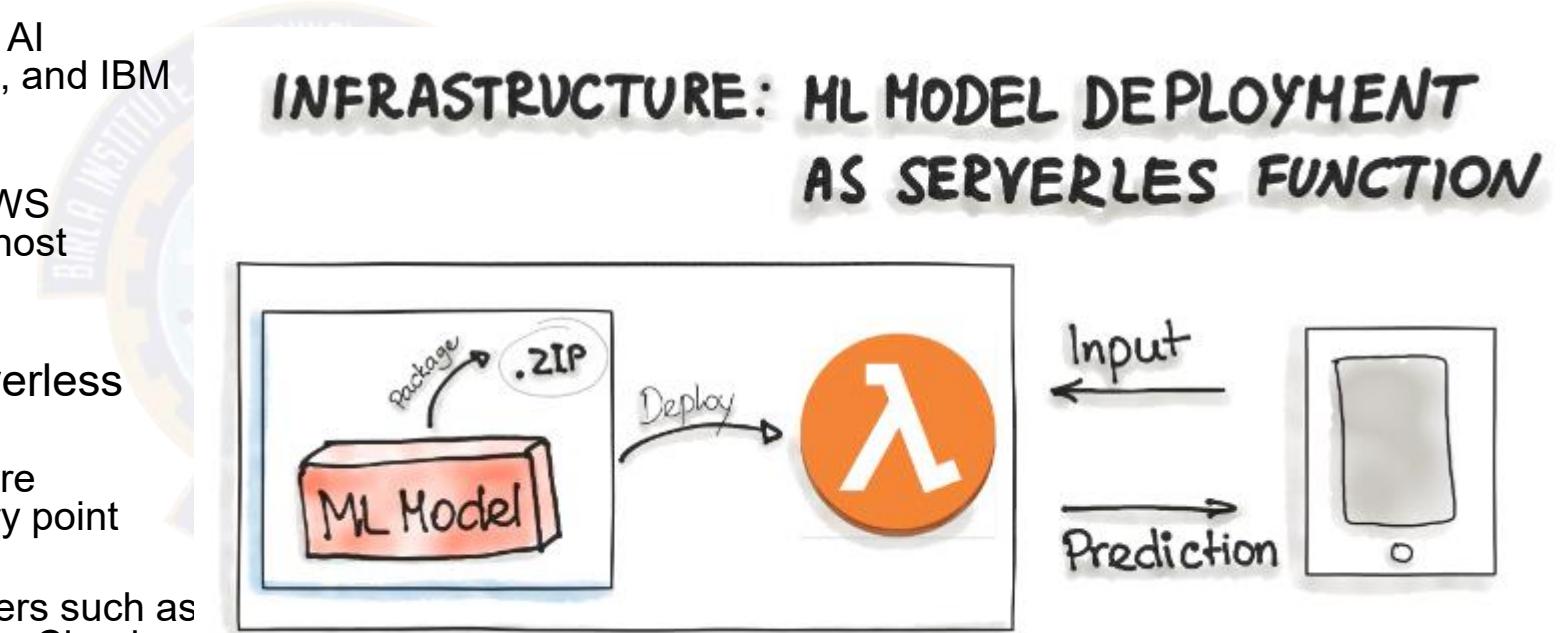
Deploying ML Models as Docker Containers

- No standard, open solution to ML model deployment!
- Containerization becomes the de-facto standard for delivery
 - as ML model inference being considered stateless, lightweight, and idempotent
 - means deploy a container that wraps an ML model inference code
- Docker is considered to be de-facto standard containerization technology
 - for on-premise, cloud, or hybrid deployments
- One ubiquitous way is to package the whole ML tech stack (dependencies) and the code for ML model prediction into a Docker container
 - Then Kubernetes or an alternative (e.g. AWS Fargate) does the orchestration
 - The ML model functionality, such as prediction, is then available through a REST API (e.g. implemented as Flask application)



Deploying ML Models as Serverless Functions

- Various cloud vendors already provide machine-learning platforms - can deploy model with their services
 - Amazon AWS Sagemaker, Google Cloud AI Platform, Azure Machine Learning Studio, and IBM Watson Machine Learning
 - Commercial cloud services also provide containerization of ML models such as AWS Lambda and Google App Engine servlet host
- In order to deploy an ML model as a serverless function,
 - the application code and dependencies are packaged into .zip files, with a single entry point function
 - could be managed by major cloud providers such as Azure Functions, AWS Lambda, or Google Cloud Functions
 - However, attention should be paid to possible constraints of the deployed artifacts such as the size of the artifacts





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Batch Prediction Versus Online Prediction

Pravin Y Pawar

Adapted from 'Designing Machine Learning Systems'

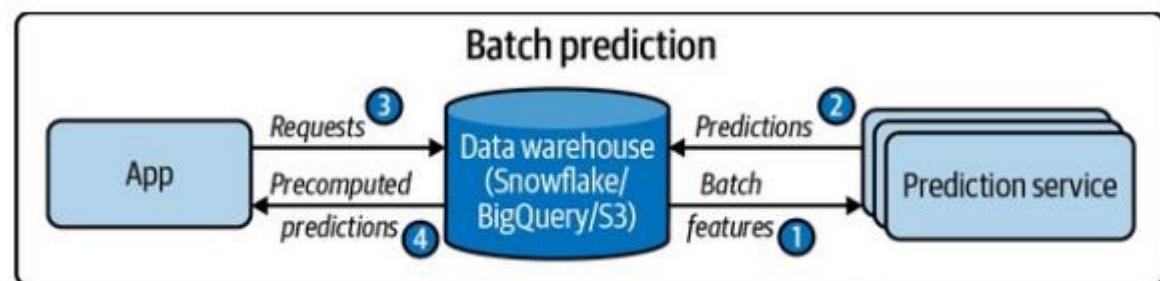
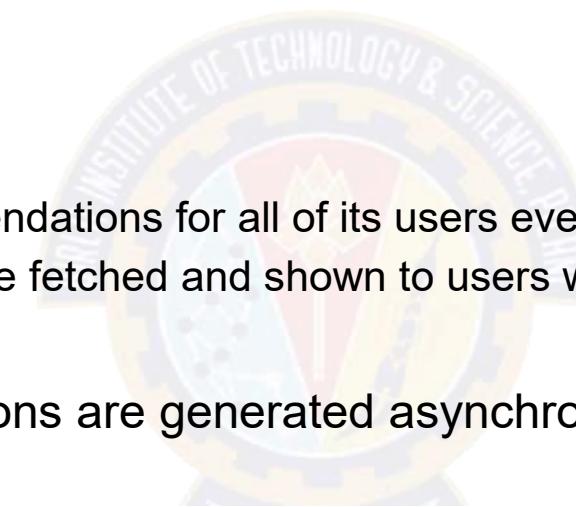
Three main modes of prediction

- Batch prediction, which uses only batch features
- Online prediction that uses only batch features (e.g., precomputed embeddings)
 - Aka on-demand prediction
- Online prediction that uses both batch features and streaming features
 - aka streaming prediction



Batch prediction

- When predictions are generated periodically or whenever triggered
 - Predictions are stored somewhere, such as in SQL tables or an in-memory database, and retrieved as needed
- For example,
 - Netflix might generate movie recommendations for all of its users every four hours,
 - the precomputed recommendations are fetched and shown to users when they log on to Netflix
- AKA asynchronous prediction: predictions are generated asynchronously with requests

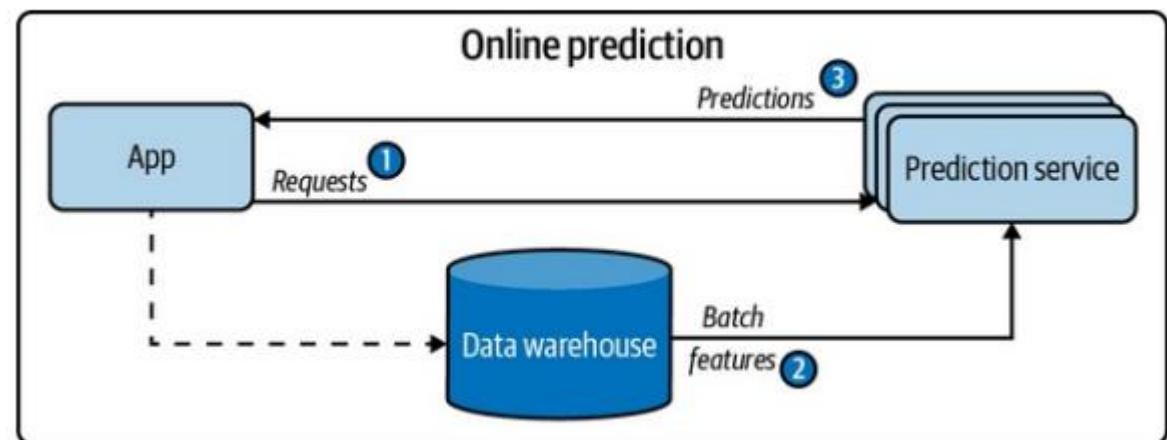
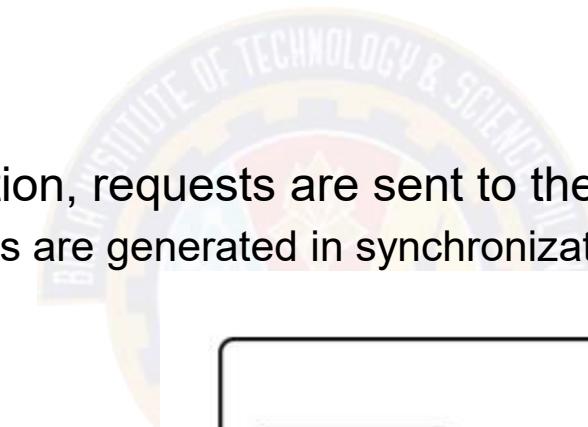


A simplified architecture for batch prediction

Online prediction

Using only batch features

- When predictions are generated and returned as soon as requests for these predictions arrive
 - For example, enter an English sentence into Google Translate and get back its French translation immediately
 - aka on-demand prediction
- Traditionally, when doing online prediction, requests are sent to the prediction service via RESTful APIs
 - aka synchronous prediction: predictions are generated in synchronization with requests



A simplified architecture for online prediction that uses only batch features

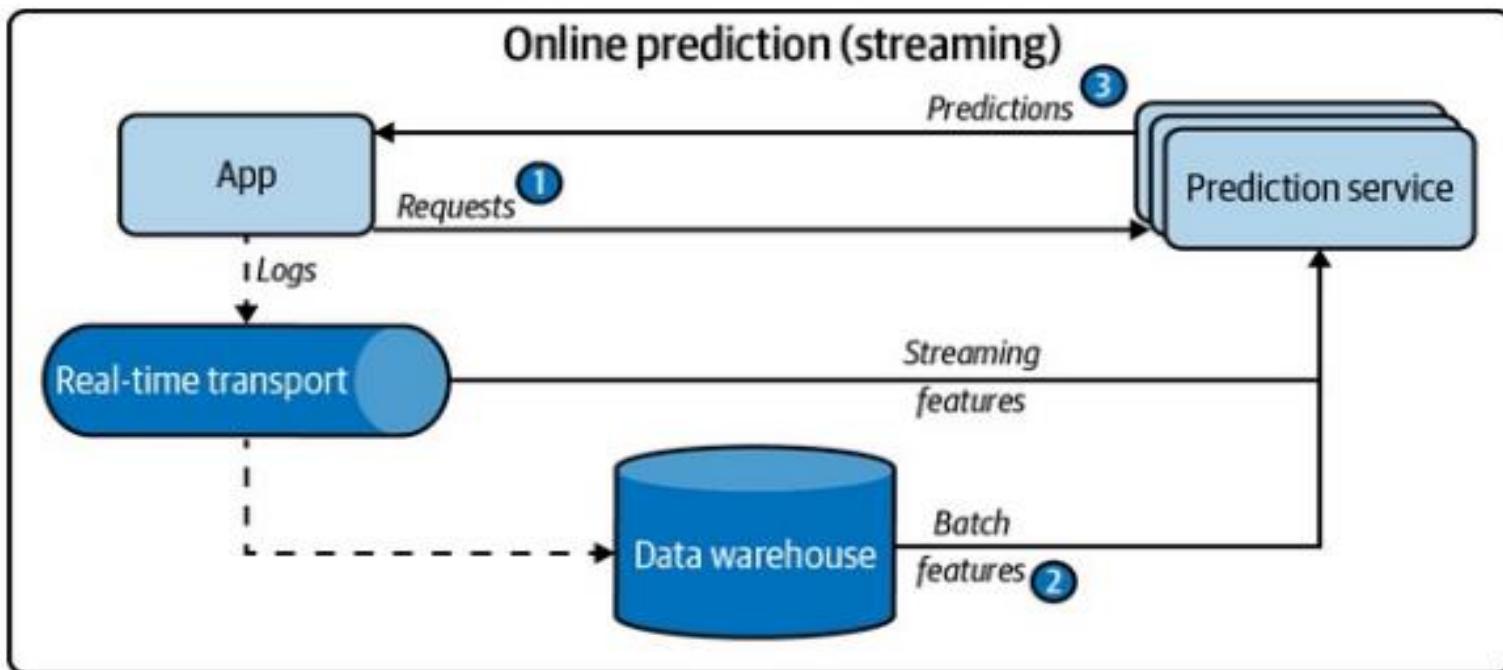
Online prediction

Using both batch and streaming features - “streaming prediction”

- Batch features
 - features computed from historical data, such as data in databases and data warehouses
 - for e.g. a restarents rating
- Streaming features
 - Features computed from streaming data — data in real-time transports
 - for e.g. estimation for delivery time
- In batch prediction, only batch features are used
- In online prediction, however, it's possible to use both batch features and streaming features.
 - For example, after a user puts in an order on DoorDash, they might need the following features to estimate the delivery time:
 - The mean preparation time of this restaurant in the past
 - In the last 10 minutes, how many other orders they have, and how many delivery people are available

Streaming prediction

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



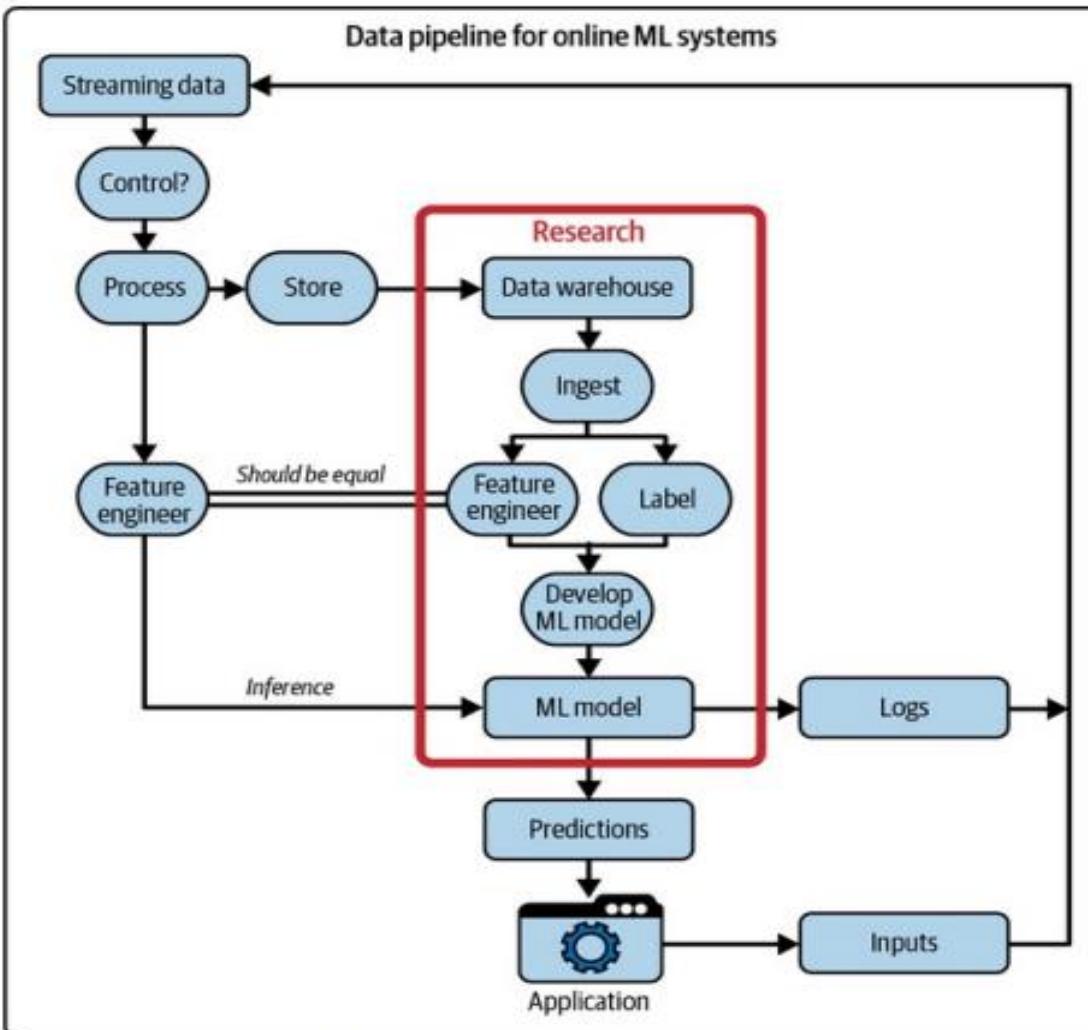
A simplified architecture for online prediction that uses both batch features and streaming features

Online vs Batch prediction

Some key differences between batch prediction and online prediction

	Batch prediction (asynchronous)	Online prediction (synchronous)
Frequency	Periodical, such as every four hours	As soon as requests come
Useful for	Processing accumulated data when you don't need immediate results (such as recommender systems)	When predictions are needed as soon as a data sample is generated (such as fraud detection)
Optimized for	High throughput	Low latency

Unifying Batch Pipeline and Streaming Pipeline





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

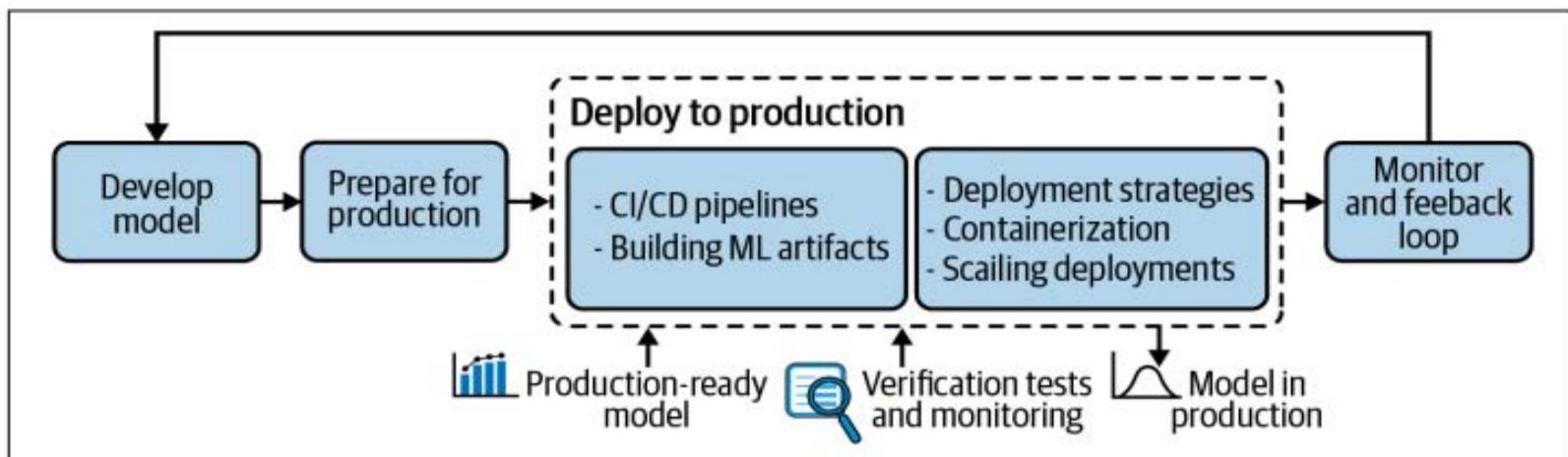
Deploying to Production

Pravin Y Pawar

Adapted from “Introducing MLOps”
By Mark Treveil and Dataiku Team

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



Deployment to production highlighted in the larger context of the ML project life cycle

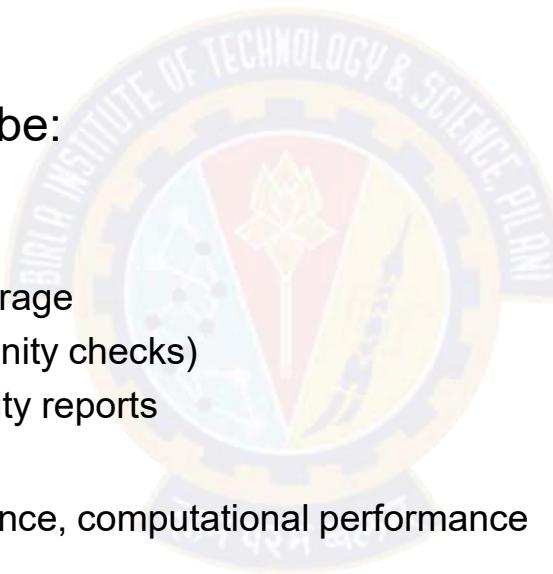
CI/CD Pipelines

A common acronym for continuous integration and continuous delivery (or put more simply, deployment)

- 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.

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



CI/CD for ML(2)

- 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

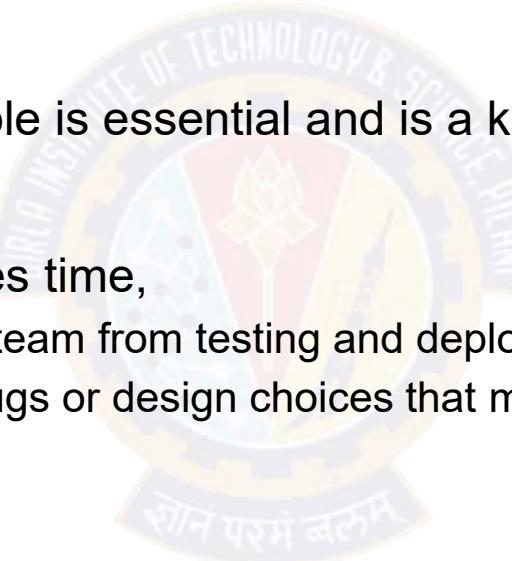
- The goal of a continuous integration pipeline is
 - to avoid unnecessary effort in merging the work from several contributors
 - also to detect bugs or development conflicts as soon as possible
- The very first step is using centralized version control systems
 - unfortunately, working for weeks on 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

Two ways to approach model deployment

- 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 both cases, multiple instances of the model can be deployed
 - to increase throughput and potentially lower latency

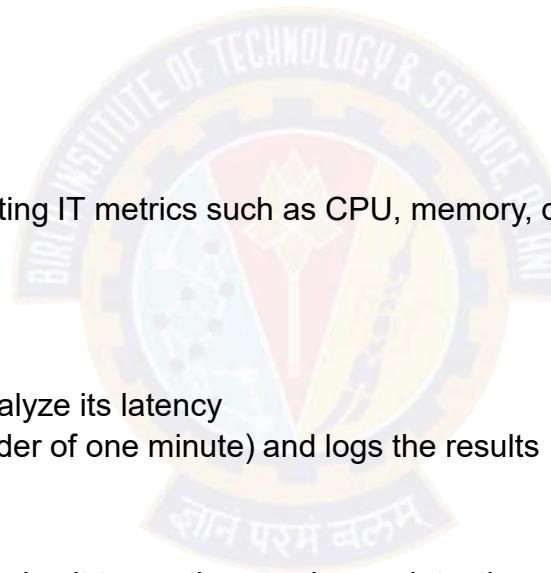
Considerations When Sending Models to Production

- 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

Once a model is released, it must be maintained

- 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





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Batch Prediction Versus Online Prediction

Pravin Y Pawar

Adapted from 'Designing Machine Learning Systems'

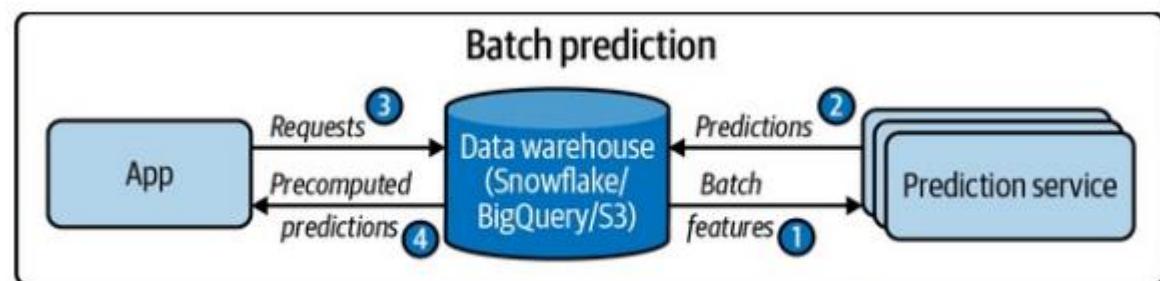
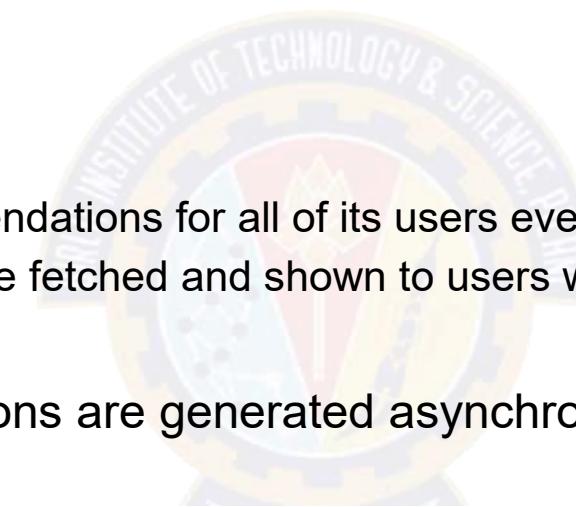
Three main modes of prediction

- Batch prediction, which uses only batch features
- Online prediction that uses only batch features (e.g., precomputed embeddings)
 - Aka on-demand prediction
- Online prediction that uses both batch features and streaming features
 - aka streaming prediction



Batch prediction

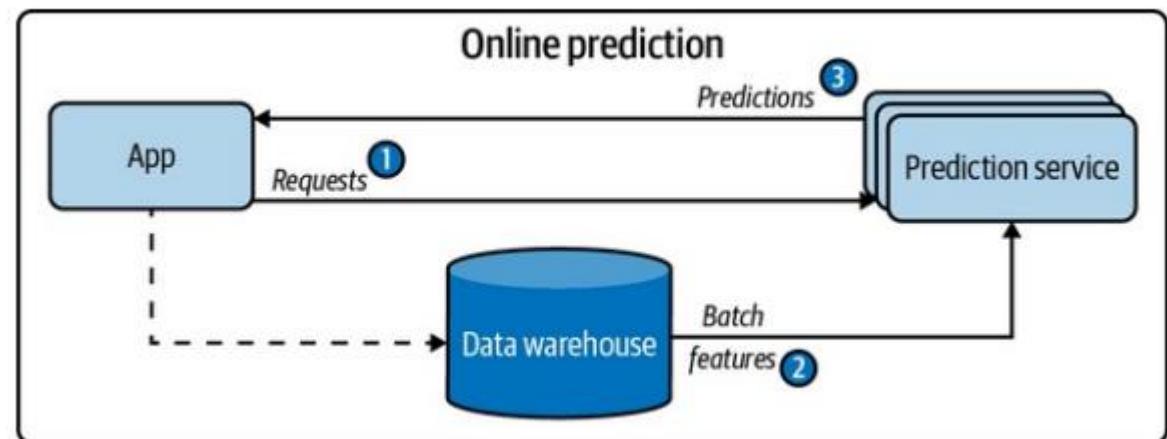
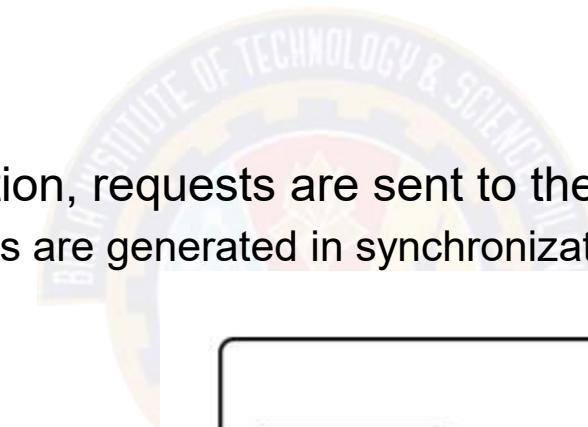
- When predictions are generated periodically or whenever triggered
 - Predictions are stored somewhere, such as in SQL tables or an in-memory database, and retrieved as needed
- For example,
 - Netflix might generate movie recommendations for all of its users every four hours,
 - the precomputed recommendations are fetched and shown to users when they log on to Netflix
- AKA asynchronous prediction: predictions are generated asynchronously with requests



Online prediction

Using only batch features

- When predictions are generated and returned as soon as requests for these predictions arrive
 - For example, enter an English sentence into Google Translate and get back its French translation immediately
 - aka on-demand prediction
- Traditionally, when doing online prediction, requests are sent to the prediction service via RESTful APIs
 - aka synchronous prediction: predictions are generated in synchronization with requests



A simplified architecture for online prediction that uses only batch features

Online vs Batch prediction

Some key differences between batch prediction and online prediction

	Batch prediction (asynchronous)	Online prediction (synchronous)
Frequency	Periodical, such as every four hours	As soon as requests come
Useful for	Processing accumulated data when you don't need immediate results (such as recommender systems)	When predictions are needed as soon as a data sample is generated (such as fraud detection)
Optimized for	High throughput	Low latency

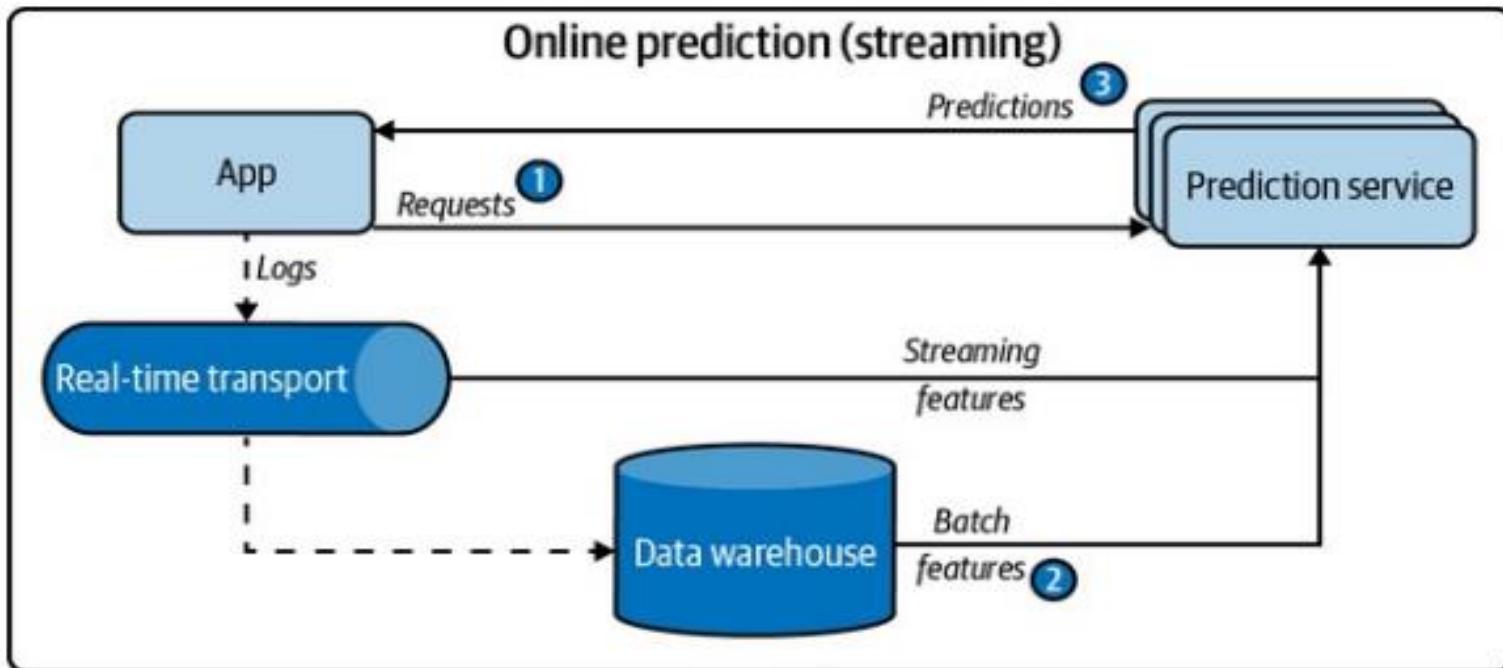
Online prediction

Using both batch and streaming features - “streaming prediction”

- Batch features
 - features computed from historical data, such as data in databases and data warehouses
 - for e.g. a restaurants rating
- Streaming features
 - Features computed from streaming data — data in real-time transports
 - for e.g. estimation for delivery time
- In batch prediction, only batch features are used
- In online prediction, however, it's possible to use both batch features and streaming features.
 - For example, after a user puts in an order on DoorDash, they might need the following features to estimate the delivery time:
 - The mean preparation time of this restaurant in the past
 - In the last 10 minutes, how many other orders they have, and how many delivery people are available

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



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Causes of ML System failure

Pravin Y Pawar

Adapted from “Designing Machine Learning Systems”
By Chip Huyen

ML system failure

- A failure happens when one or more expectations of the system is violated
- In traditional software, mostly care about a system's operational expectations:
 - whether the system executes its logic within the expected operational metrics,
 - e.g., latency and throughput
- For an ML system, care about both its operational metrics and its ML performance metrics
- For example, consider an English-French machine translation system
 - Operational expectation might be that, given an English sentence, the system returns a French translation within a one-second latency
 - ML performance expectation is that the returned translation is an accurate translation of the original English sentence 99% of the time

ML system failure types

- Operational expectation violations are easier to detect,
 - as usually accompanied by an operational breakage
 - such as a timeout, a 404 error on a webpage, an out-of-memory error, or a segmentation fault
- However, ML performance expectation violations are harder to detect
 - as doing so requires measuring and monitoring the performance of ML models in production
- In the English-French machine translation system, detecting whether the returned translations are correct 99% of the time is difficult
 - if don't know what the correct translations are supposed to be
- To effectively detect and fix ML system failures in production,
 - it's useful to understand why a model, after proving to work well during development, would fail in production
- Two types of failures: software system failures and ML-specific failures

Software System Failures

failures that would have happened to non-ML systems

- Dependency failure
 - A software package or a codebase that system depends on breaks, which leads system to break
 - common when the dependency is maintained by a third party
- Deployment failure
 - failures caused by deployment errors,
 - such as when accidentally deploy the binaries of an older version of model instead of the current version,
 - or when systems don't have the right permissions to read or write certain files
- Hardware failures
 - When the hardware that is used to deploy model, such as CPUs or GPUs, doesn't behave the way it should
 - For example, the CPUs used might overheat and break down
- Downtime or crashing
 - If a component of system runs from a server somewhere, such as AWS or a hosted service, and that server is down, system will also be down

Software System Failures(2)

- Addressing software system failures requires not ML skills, but traditional software engineering skills!
- Because of the importance of traditional software engineering skills in deploying ML systems,
 - ML engineering is mostly engineering, not ML!
- The reasons for the prevalence of software system failures:
 - ML adoption in the industry is still nascent,
 - tooling around ML production is limited
 - and best practices are not yet well developed or standardized
- However, as tooling's and best practices for ML production mature,
 - there are reasons to believe that the proportion of software system failures will decrease
 - and the proportion of ML-specific failures will increase

ML-Specific Failures

failures specific to ML systems

- Examples include
 - data collection and processing problems,
 - poor hyper parameters,
 - changes in the training pipeline not correctly replicated in the inference pipeline and vice versa,
 - data distribution shifts that cause a model's performance to deteriorate over time,
 - edge cases,
 - and degenerate feedback loops
- Even though they account for a small portion of failures, they can be more dangerous than non-ML failures
 - as they're hard to detect and fix, and they can prevent ML systems from being used altogether

Production data differing from training data

- ML model learns from the training data, means that the model learns the underlying distribution of the training data
 - with the goal of leveraging this learned distribution to generate accurate predictions for unseen data
 - when the model is able to generate accurate predictions for unseen data - model “generalizes to unseen data”
- The assumption
 - unseen data comes from a stationary distribution that is the same as the training data distribution
- This assumption is **incorrect** in most cases for two reasons!
- First, the **underlying distribution of the real-world data is unlikely to be the same as the underlying distribution of the training data**
 - Real-world data is multifaceted and, in many cases, virtually infinite,
 - whereas training data is finite and constrained by the time, compute, and human resources available during the dataset creation and processing
 - divergence leads to a common failure mode known as the train-serving skew: a model that does great in development but performs poorly when deployed
- Second, **the real world isn't stationary. Things change. Data distributions shift.**

Edge cases

- Edge cases are the data samples so extreme that they cause the model to make catastrophic mistakes
- An ML model that performs well on most cases but fails on a small number of cases
 - might not be usable if these failures cause catastrophic consequences
 - major self-driving car companies are focusing on making their systems work on edge cases
- Also true for any safety-critical application such as medical diagnosis, traffic control, e-discovery, etc.
- Can also be true for non-safety-critical applications
 - Imagine a customer service chatbot that gives reasonable responses to most of the requests,
 - but sometimes, it spits out outrageously racist or sexist content
 - will be a brand risk for any company that wants to use it, thus rendering it unusable

Degenerate feedback loops

- Feedback loop as the time it takes from when a prediction is shown until the time feedback on the prediction is provided
 - feedback can be used to
 - extract natural labels to evaluate the model's performance
 - train the next iteration of the model
- A degenerate feedback loop can happen when the predictions themselves influence the feedback,
 - which, in turn, influences the next iteration of the model
- A degenerate feedback loop is created when a system's outputs are used to generate the system's future inputs,
 - which, in turn, influence the system's future outputs
- Degenerate feedback loops are especially common in tasks with natural labels from users,
 - such as recommender systems and ads click-through-rate prediction
- Imagine building a system to recommend to users songs that they might like
 - songs that are ranked high by the system are shown first to users
 - because they are shown first, users click on them more,
 - which makes the system more confident that these recommendations are good



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Monitoring and Feedback Loop

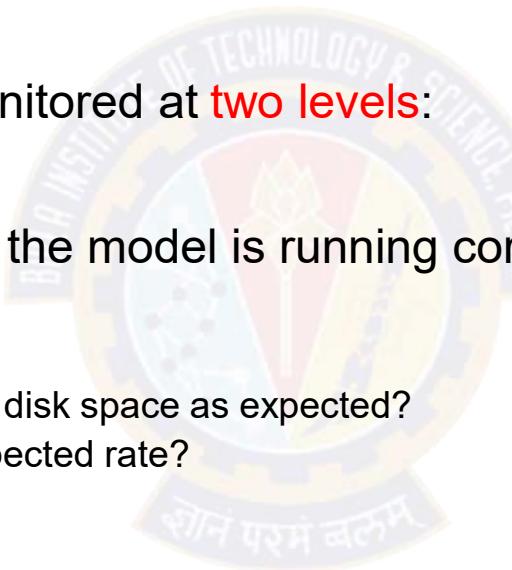
Pravin Y Pawar

Adapted from “Introducing MLOps”
By Treveil et al

Model monitoring

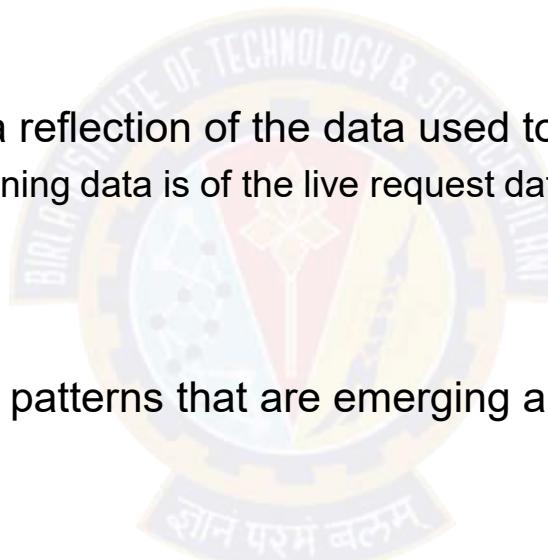
Model monitoring is a crucial step in the ML model life cycle and a critical piece of MLOps

- When a machine learning model is deployed in production
 - it can start degrading in quality fast—and without warning—until it's too late
- Machine learning models need to be monitored at **two levels**:
- At the **resource level**, including ensuring the model is running correctly in the production environment.
 - Key questions include:
 - Is the system alive?
 - Are the CPU, RAM, network usage, and disk space as expected?
 - Are requests being processed at the expected rate?
- At the **performance level**, meaning monitoring the pertinence of the model over time
 - Key questions include:
 - 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 performance monitoring

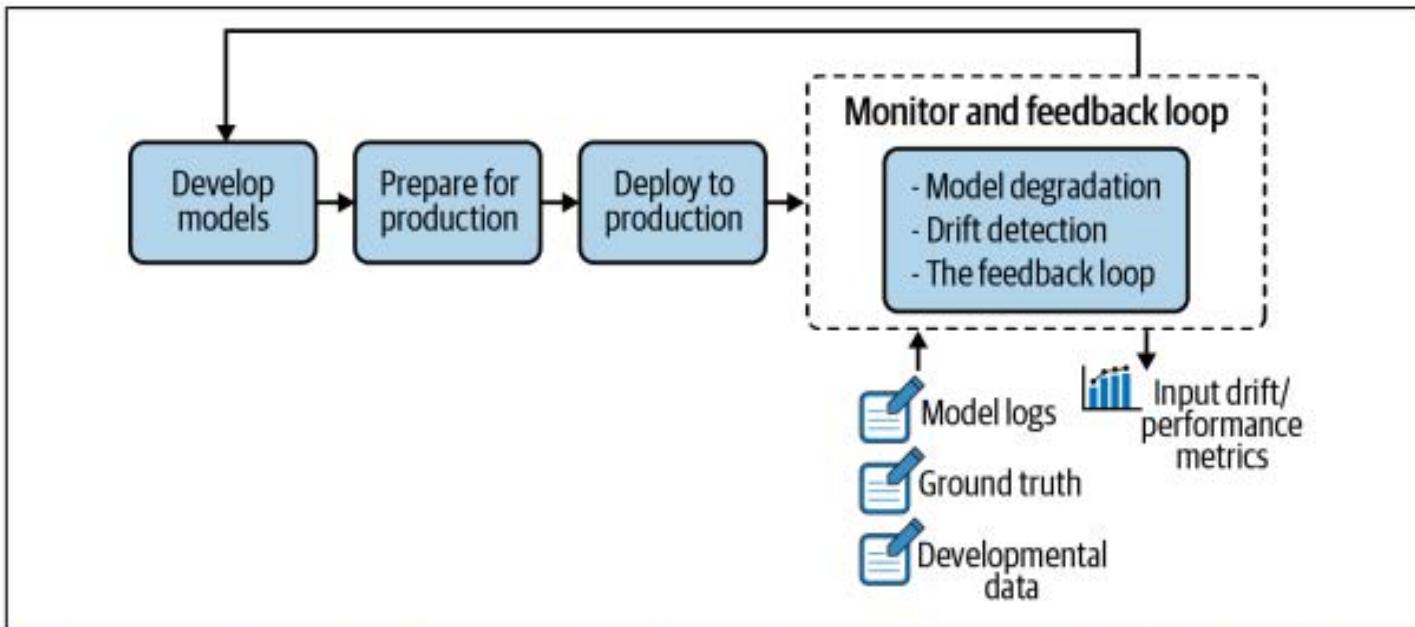
- The first level is a traditional DevOps topic
- The latter is more complicated. Why?
 - Because how well a model performs is a reflection of the data used to train it
 - in particular, how representative that training data is of the live request data
- As the world is constantly changing,
 - a static model cannot catch up with new patterns that are emerging and evolving without a constant source of new 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
- Ideal scenario would be a pipeline that automatically triggers checks for degradation of model performance
 - 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
- 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

Monitoring and feedback loop



Monitoring and feedback loop highlighted in the larger context of the ML project life cycle

Model Degradation

- Once a machine learning model is trained and deployed in production
- Two approaches to monitor its performance degradation:
 - ground truth evaluation
 - input drift detection



Ground Truth Evaluation

Defined

- Ground truth retraining requires waiting for the label event
 - In a fraud detection model, the ground truth would be whether or not a specific transaction was actually fraudulent
 - For a recommendation engine, it would be whether or not the customer clicked on—or ultimately bought—one of the recommended products
- With the new ground truth collected,
 - next step is to compute the performance of model based on ground truth
 - compare it with registered metrics in the training phase
- When the difference surpasses a threshold, the model can be deemed as outdated, and it should be retrained
- The metrics to be monitored can be of two varieties:
 - Statistical metrics like accuracy, ROC AUC, log loss, etc.
 - domain agnostic
 - drawback is that the drop may be statistically significant without having any noticeable impact
 - cost of retraining and risk associated with a redeployment may be higher than expected benefits
 - Business metrics, like cost-benefit assessment such as the credit scoring
 - far more interesting because they ordinarily have a monetary value
 - enabling subject matter experts to better handle the cost-benefit trade-off of the retraining decision

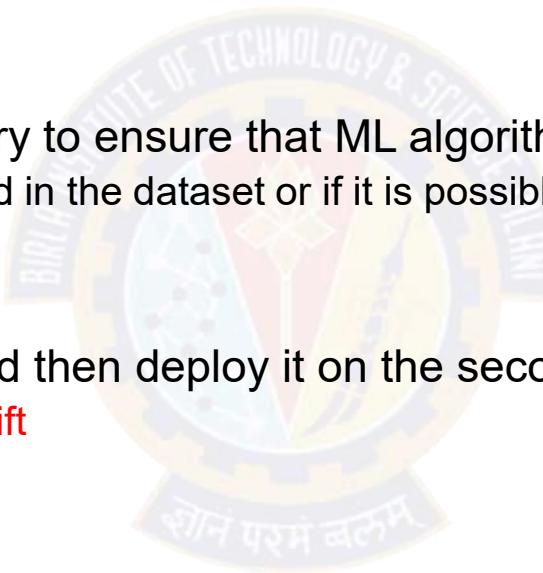
Ground Truth Evaluation(2)

Challenges

- Ground truth is not always immediately, or even imminently, available
 - For some types of models, teams need to wait months (or longer) for ground truth labels to be available,
 - which can mean significant economic loss if the model is degrading quickly
 - Deploying a model for which the drift is faster than the lag is risky
- Ground truth and prediction are decoupled
 - To compute the performance of the deployed model on new data, it's necessary to be able to match ground truth with the corresponding observation
 - In many production environments, this is a challenging task because these two pieces of information are generated and stored in different systems and at different timestamps
- Ground truth is only partially available
 - In some situations, it is extremely expensive to retrieve the ground truth for all the observations,
 - which means choosing which samples to label and thus inadvertently introducing bias into the system

Input Drift

- Mathematically speaking, the samples of each dataset cannot be assumed to be drawn from the same distribution
 - i.e., they are not “identically distributed”
- Another mathematical property is necessary to ensure that ML algorithms perform as expected: independence
 - property is broken if samples are duplicated in the dataset or if it is possible to forecast the “next” sample given the previous one
- If train the algorithm on the first dataset and then deploy it on the second one
 - The **resulting distribution shift is called a drift**
- In wine quality prediction exercise,
- Called a **feature drift**
 - if the alcohol level is one of the features used by the ML model
 - or if the alcohol level is correlated with other features used by the model
- Called as a **concept drift** if it is not



Drift Detection in Practice

- To be able to react in a timely manner, model behavior should be monitored solely based on the feature values of the incoming data,
 - without waiting for the ground truth to be available
- The logic is that if the data distribution (e.g., mean, sd, correlations between features) diverges
 - between the training and testing phases on one side and the development phase on the other,
 - **a strong signal that the model's performance won't be the same!**
- Not the perfect mitigation measure, as retraining on the drifted dataset will not be an option,
 - but it can be part of mitigation measures (e.g., reverting to a simpler model, reweighting)

Causes of Data Drift

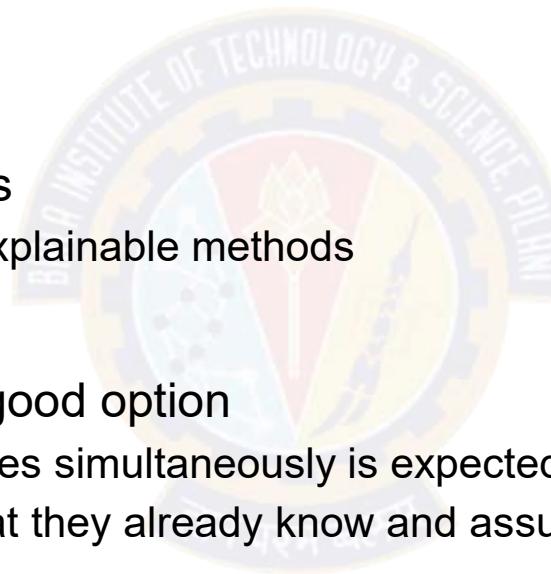
Two frequent root causes of data drift

- Sample selection bias
 - the training sample is not representative of the population
 - often stems from the data collection pipeline itself
 - For instance, building a model to assess the effectiveness of a discount program will be biased
 - if the best discounts are proposed for the best clients
- Non-stationary environment
 - where training data collected from the source population does not represent the target population
 - Often happens for time dependent tasks — such as forecasting with strong seasonality effects,
 - where learning a model over a given month won't generalize to another month

Input Drift Detection Techniques

Choice depends on the expected level of interpretability

- Two approaches
 - Univariate statistical tests
 - Domain classifier
- Should prefer univariate statistical tests
 - Organizations that need proven and explainable methods
- Domain classifier approach may be a good option
 - if complex drift involving several features simultaneously is expected
 - if the data scientists want to reuse what they already know and assuming the organization doesn't dread the black box effect



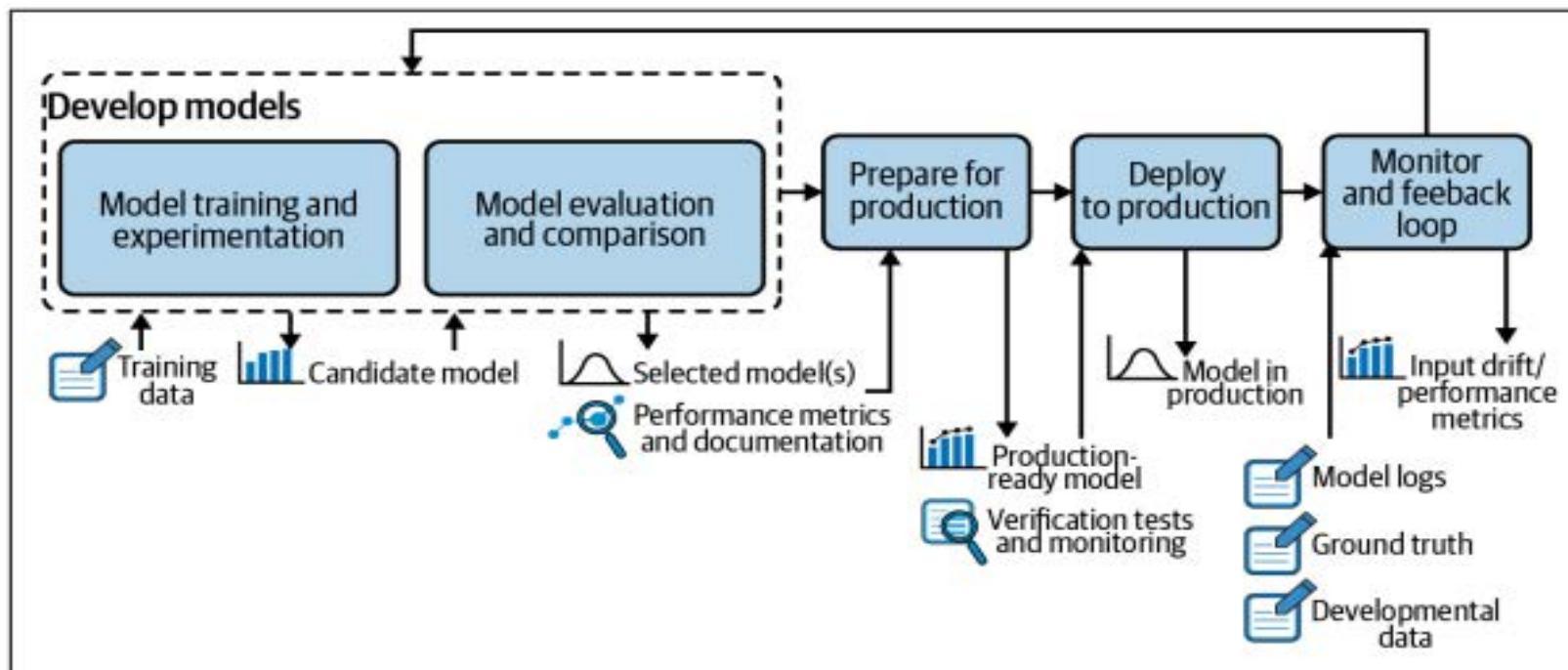
Univariate statistical tests

- Requires applying a statistical test on data from the source distribution and the target distribution for each feature
 - A warning will be raised when the results of those tests are significant
- Basic approaches rely on two tests:
 - For continuous features, the Kolmogorov-Smirnov test is a nonparametric hypothesis test that is used to check whether two samples come from the same distribution
 - For categorical features, the Chi-squared test is a practical choice that checks whether the observed frequencies for a categorical feature in the target datamatch the expected frequencies seen from the source data
- The main **advantage** of p-values is that they help detect drift as quickly as possible
- The main **drawback** is that they do not quantify the level of the effect
- If development datasets are very large, it is necessary to complement p-values with business-significant metrics
 - For example, on a sufficiently large dataset, the average age may have significantly drifted from a statistical perspective,
 - but if the drift is only a few months, this is probably an insignificant value for many business use cases

Domain classifier

- Data scientists train a model that tries to discriminate between the original dataset (input features and, optionally, predicted target) and the development dataset
 - stack the two datasets and train a classifier that aims at predicting the data's origin
- The performance of the model (its accuracy, for example) can then be considered as a metric for the drift level
 - If this model is successful in its task, and thus has a high drift score,
 - implies that the data used at training time and the new data can be distinguished
 - fair to say that the new data has drifted
- To gain more insights, in particular to identify the features that are responsible for the drift
 - one can use the feature importance of the trained model

Continuous delivery for end-to-end machine learning process

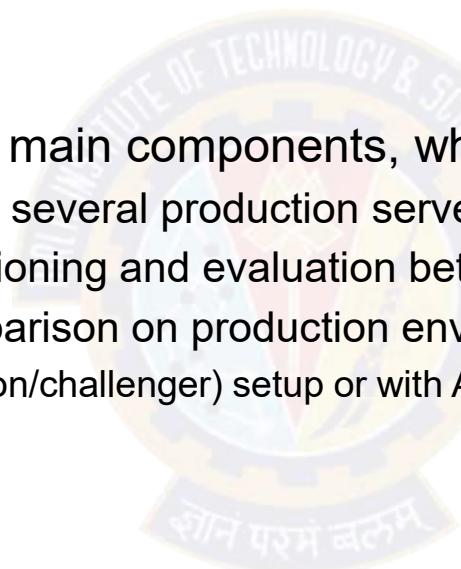


The Feedback Loop

- All effective machine learning projects implement a form of data feedback loop
 - information from the production environment flows back to the model prototyping environment for further improvement
- Continuous delivery for end-to-end machine learning process
 - 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 it is, no action is required
 - If the model's performance is degrading, an update will be triggered, either automatically or manually by the data scientist
- In practice, usually means either retraining the model with new labeled data or developing a new model with additional features

The Feedback Loop(2)

- Goal of retraining is to be able to capture the emerging patterns and make sure that the business is not negatively impacted
- This infrastructure is comprised of three main components, which are critical to robust MLOps capabilities:
 - A logging system that collects data from several production servers
 - A model evaluation store that does versioning and evaluation between different model versions
 - An online system that does model comparison on production environments,
 - either with the shadow scoring (champion/challenger) setup or with A/B testing



Logging

- Monitoring a live system means collecting and aggregating data about its states
- Nowadays, as production infrastructures are getting more and more complex,
 - with several models deployed simultaneously across several servers,
 - **an effective logging system is more important than ever!**
- Data from these environments needs to be centralized to be analyzed and monitored,
 - either automatically or manually
 - will enable continuous improvement of the ML system
- **An event log of a machine learning system is a record with a timestamp and information such as**
 - **Model metadata** - Identification of the model and the version
 - **Model inputs** - Feature values of new observations
 - **Model outputs** - Predictions made by the model that
 - **System action** - an action taken based on model prediction
 - **Model explanation** - an explanation of prediction (i.e., which features have the most influence on the prediction)

Model Evaluation

- If model performance is degrading, after review, data scientists decide to improve the model by retraining it,
 - With several trained candidate models, the next step is to compare them with the deployed model
 - means evaluating all the models (the candidates as well as the deployed model) on the same dataset
- If one of the candidate models outperforms the deployed model, there are two ways to proceed:
 - either update the model on the production environment
 - or move to an online evaluation via a champion/challenger or A/B testing setup
- In a nutshell, this is the **notion of model store**
- A structure that allows data scientists to:
 - Compare multiple, newly trained model versions against existing deployed versions
 - Compare completely new models against versions of other models on labeled data
 - Track model performance over time

Model evaluation store

- Formally, the model evaluation store serves as a structure that centralizes the data related to model life cycle to allow comparisons
- Two main tasks of a model evaluation store are:
 - Versioning the evolution of a logical model through time
 - Each logged version of the logical model must come with all the essential information concerning its training phase, including:
 - The list of features used
 - The preprocessing techniques that are applied to each feature
 - The algorithm used, along with the chosen hyperparameters
 - The training dataset
 - The test dataset used to evaluate the trained model (this is necessary for the version comparison phase)
 - Evaluation metrics
 - Comparing the performance between different versions of a logical model

Online Evaluation

- Online evaluation of models in production is critical from a business perspective,
 - but can be challenging from a technical perspective
- Two main modes of online evaluation:
 - Champion/challenger (otherwise known as shadow testing)
 - the candidate model shadows the deployed model and scores the same live requests
 - A/B testing
 - the candidate model scores a portion of the live requests and the deployed model scores the others
- Both cases require ground truth
 - evaluation will necessarily take longer than the lag between prediction and ground truth obtention
- Whenever shadow testing is possible, it should be used over A/B testing
 - because it is far simpler to understand and set up, and it detects differences more quickly



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Production Monitoring

Pravin Y Pawar

Adapted from “Reliable Machine Learning”
By Cathy Chen

The basics

- Monitoring, at the most basic level, provides data about how your systems are performing
 - data is made storable, accessible, and displayable in some reasonable way
- Observability is an attribute of software, meaning that when correctly written, the emitted monitoring data
 - usually extended or expanded in some way, with labeling or tagging
 - can be used to correctly infer behavior of system
- Obviously, monitoring is hugely important in and of itself,
 - but an offshoot of monitoring is absolutely crucial: alerting
 - A useful simplification is that when things go wrong, humans are alerted to fix them
 - defining the conditions for “things going wrong,” and being able to reliably notify the responsible folks that

What Does It Look Like?

- To do monitoring,
 - must have a monitoring system
 - as well as systems to be monitored (called the target systems)
- Today, target systems
 - emit metrics a series, typically of numbers, with an identifying name
 - which are then collected by the monitoring system
 - and transformed in various ways,
 - often via aggregation (producing a sum or a rate across multiple instances or machines)
 - or decoration (adding, say, event details onto the same data)
 - aggregated metrics are used for system analysis, debugging, and the alerting

Example

Web Server

- Web server emits a metric of the total number of requests it received
 - The monitoring system will obtain these metrics, usually via push or pull,
 - which refers to whether the metrics get pulled from the target systems or get pushed from them
 - These metrics are then collated, stored,
 - and perhaps processed in some way, generally as a time series
- Different monitoring systems will make different choices about how to receive, store, process,
 - but the data is generally queryable and often there's a graphical way to plot the monitoring data
 - to take advantage of our visual comparison hardware (eyes, retinas, optic nerves, and so on) to figure out what's actually happening

Problems with ML Production Monitoring

- ML model development is still in its infancy!
 - tools are immature, conceptual frameworks are underdeveloped
 - discipline is in short supply, as everyone scrambles to get some kind of model—any kind of model!
- The pressure to ship is real and has real effects!
- In particular, model development,
 - which is inherently hard because it involves reconciling a wide array of conflicting concerns
 - gets harder because that urgency forces developers and data science folks to focus on those hard problems
 - ignore the wider picture
- That wider picture often involves questions around monitoring and observability!

Difficulties of Development Versus Serving

The first problem is that effectively simulating production in development is extremely hard

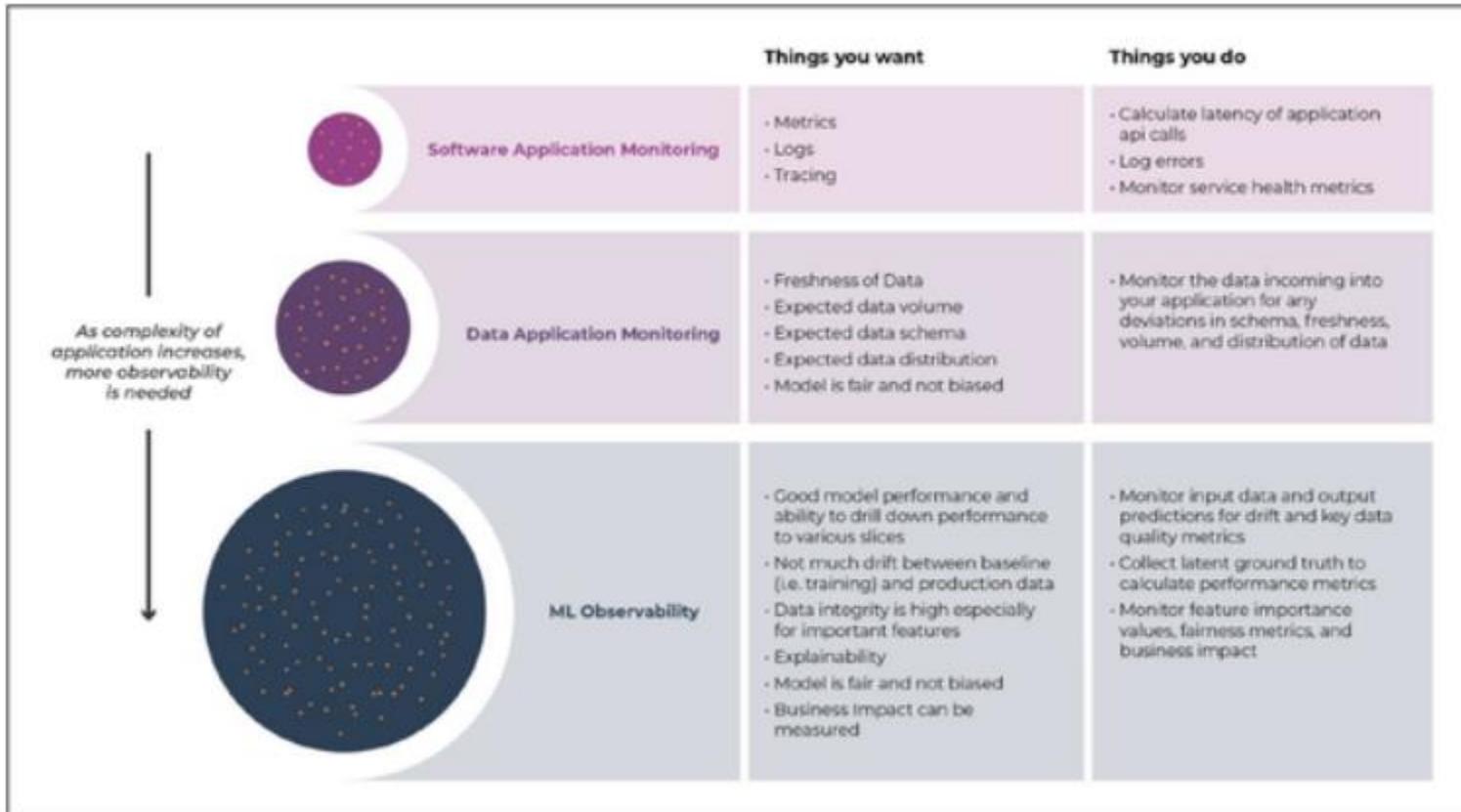
- Even if separate environments dedicated to that task (like test, staging, and so on.)
- Primarily Because of
 - the wide variety of possible serving architectures
 - model pools, shared libraries, edge devices, etc., with the associated infrastructure running on)
 - the invocation of the predictions
 - in development often invoke prediction methods directly or with a relatively small amount of code between developer and the model for velocity reasons
 - Running in production also generally means don't have the ability to manipulate input, logging level, processing, and so on
 - leading to huge difficulties in debugging, reproducing problematic configurations, etc.
 - data in testing is not necessarily distributed like the data the model encounters in production
 - as always for ML, data distribution really matters

Difficulties of Development Versus Serving(2)

The second problem is about mature practices

- In conventional software delivery,
 - the industry has a good handle on work practices to improve throughput, reliability, and developmental velocity
 - most important is grouped concepts of continuous integration / continuous deployment (CI/CD), unit tests, small changes
- Unfortunately, today w\missing this equivalent of CI/CD for model development
 - not yet converged onto a good set of (telemetry-related, or otherwise) tools for model training and validation
- Expect this will improve over time as
 - existing tools (such as MLflow and Kubeflow) gain traction
 - vendors incorporate more of these concerns into their platforms
 - and the mindset of holistic, or whole-lifecycle monitoring gains more acceptance

Observability layers and system requirements



Observability layers and system requirements

Reasons for Continual ML Observability—in Production

- Observability data from models is absolutely fundamental to business
 - both tactical operations and strategic insights
- One example - connection between latency and online sales
 - In 2008, Amazon discovered that each additional 100 ms of latency lost 1% of sales, and also the converse
 - Similar results have been confirmed by Akamai Technologies, Google, Zalando, and others
- Without observability, there would be no way to have discovered this effect,
 - and certainly no way to know for sure that either making it better or worse!
- Ultimately, observability data is business outcome data
 - In the era of ML, this happily allows not just to detect and respond to outages,
 - but also to understand incredibly important things that are happening to business

ML observability in context

System/Infra Observability

- Infra/App timing as the base of monitoring
- App & system response time issues
- Tracing & troubleshooting response time



dynatrace

Data Observability

- Tables as the base of monitoring
- Monitoring data changes
- Schema monitoring

MONTE CARLO

Bigeye

ML Observability

- Models are the base of monitoring
- Distributions, vs baselines, model version, SHAP analysis and performance
- Deep model performance analysis vs data

arize



Software/DevOps



Data Eng



ML Engineer & DS

[arize](#)

Key Components of Observability

System vs Machine Learning

System Observability	ML Observability
Logs <ul style="list-style-type: none">Records of an event that happened within an application.Typically not mutable by an event ID.Searchable by tags and unstructured indexes.	Inference Store <ul style="list-style-type: none">Records of ML prediction events that are logged from the model.Raw prediction events that hold granular context about the model's predictions.Mutable by prediction ID and dataset.
Metrics <ul style="list-style-type: none">Measured values of system performance.Metrics comprise a set of attributes (i.e. value, label, and timestamp) that convey information about SLAs, SLOs, and SLIs.	Model Metrics <ul style="list-style-type: none">Calculated metrics on the prediction events.Provides ways to determine model health over time – this includes drift, performance, and data quality metrics.Metrics can be monitored.Metrics can be aggregate or slice-level.
Tracing <ul style="list-style-type: none">Provides context for the other components of observability (logs, metrics).Follows the entire lifecycle of a request or action across distributed systems.	ML Performance Tracing <ul style="list-style-type: none">ML performance tracing is the methodology for pinpointing the source of a model performance problem.Involves mapping back to the data that caused the problem.Necessarily a distinct discipline because logs and metrics are rarely helpful for debugging model performance.



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Monitoring - Metrics and Toolbox

Pravin Y Pawar

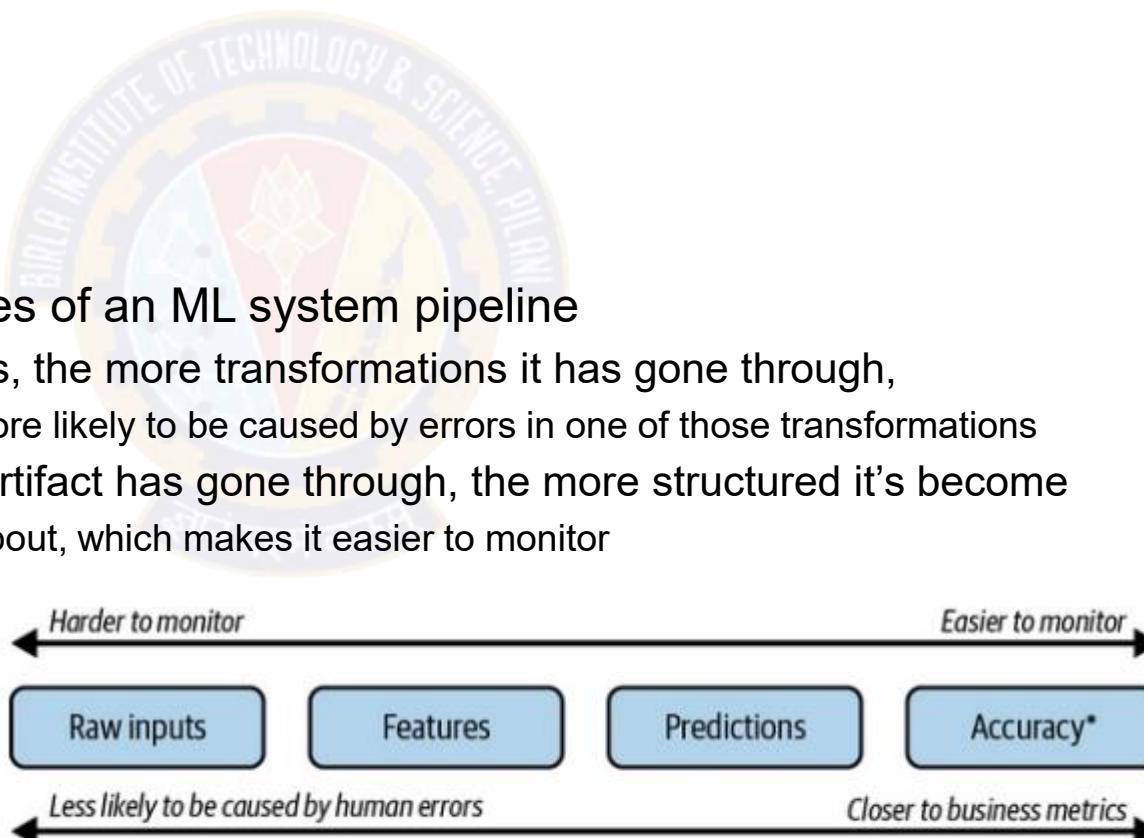
Adapted from “Designing Machine Learning Systems”
By Chip Huyen

ML Monitoring

- As the industry realized that many things can go wrong with an ML system,
 - many companies started investing in monitoring and observability for their ML systems in production
- Monitoring refers to the act of tracking, measuring, and logging different metrics that can help us determine when something goes wrong
- Observability means setting up our system in a way that gives us visibility into system to help us investigate what went wrong
- Monitoring is all about metrics!
 - Because ML systems are software systems, the first class of metrics need to monitor are the operational metrics
 - designed to convey the health of systems
- Generally divided into three levels:
 - the network the system is run on,
 - the machine the system is run on,
 - and the application that the system runs
- Examples
 - latency; throughput; the number of prediction requests model receives in the last minute, hour, day;
 - the percentage of requests that return with a 2xx code;
 - CPU/GPU utilization; memory utilization; etc.

ML-Specific Metrics

- Within ML-specific metrics, there are generally four artifacts to monitor:
 - a model's accuracy-related metrics,
 - predictions,
 - features,
 - and raw inputs
- Artifacts generated at four different stages of an ML system pipeline
 - The deeper into the pipeline an artifact is, the more transformations it has gone through,
 - which makes a change in that artifact more likely to be caused by errors in one of those transformations
 - However, the more transformations an artifact has gone through, the more structured it's become
 - closer it is to the metrics actually care about, which makes it easier to monitor



Monitoring accuracy-related metrics

- Accuracy-related metrics are the most direct metrics to help you decide whether a model's performance has degraded
- If system receives any type of user feedback for the predictions it makes
 - click, hide, purchase, upvote, downvote, favorite, bookmark, share, etc.
 - should definitely log and track it
- Some feedback can be used to infer natural labels,
 - which can then be used to calculate model's accuracy-related metrics
- Feedback can be used to detect changes in ML model's performance
 - For example, when building a system to recommend to users what videos to watch next on YouTube,
 - want to track not only whether the users click on a recommended video (click-through rate),
 - but also the duration of time users spend on that video and whether they complete watching it (completion rate)
 - If, over time, the clickthrough rate remains the same but the completion rate drops, it might mean that recommender system is getting worse
- Possible to engineer system to collect users' feedback
 - For example, Google Translate has the option for users to upvote or downvote a translation

Monitoring predictions

- Prediction is the most common artifact to monitor
 - If it's a regression task, each prediction is a continuous value (e.g., the predicted price of a house)
 - If it's a classification task, each prediction is a discrete value corresponding to the predicted category
- Each prediction is usually just a number (low dimension), predictions are easy to visualize
 - summary statistics are straightforward to compute and interpret
- Can monitor predictions for distribution shifts as they are low dimensional
 - Easier to compute two-sample tests to detect whether the prediction distribution has shifted
 - Prediction distribution shifts are also a proxy for input distribution shifts
- Can also monitor predictions for anything odd happening
 - such as predicting an unusual number of False in a row

Monitoring features

- ML monitoring solutions in the industry focus on tracking changes in features, both
 - the features that a model uses as inputs
 - the intermediate transformations from raw inputs into final features
- Feature monitoring is appealing because
 - compared to raw input data, features are well structured following a predefined schema
- The first step of feature monitoring is feature validation:
 - ensuring that features follow an expected schema
- Things can be checked for a given feature:
 - If the min, max, or median values of a feature are within an acceptable range
 - If the values of a feature satisfy a regular expression format
 - If all the values of a feature belong to a predefined set
 - If the values of a feature are always greater than the values of another feature
- Many open source libraries that help in basic feature validation,
 - Two most common are Great Expectations and Deequ, which is by AWS

Monitoring features(2)

Four major concerns when doing feature monitoring

- A company might have hundreds of models in production, and each model uses hundreds, if not thousands, of features.
 - Even something as simple as computing summary statistics for all these features every hour can be expensive,
 - not only in terms of compute required but also memory used
 - Tracking, i.e., constantly computing, too many metrics can also slow down system
 - increase both the latency that users experience
- While tracking features is useful for debugging purposes, it's not very useful for detecting model performance degradation
 - In practice, an individual feature's minor changes might not harm the model's performance at all
 - Feature distributions shift all the time, and most of these changes are benign
 - If want to be alerted whenever a feature seems to have drifted, might soon be overwhelmed by alerts
 - realize that most of these alerts are false positives - "alert fatigue"
- Feature extraction is often done in multiple steps (such as filling missing values and standardization),
 - using multiple libraries (such as pandas, Spark),
 - on multiple services (such as BigQuery or Snowflake).
- Even if its detected a harmful change in a feature, it might be impossible to detect whether this change is
 - caused by a change in the underlying input distribution or whether it's caused by an error in one of the multiple processing steps
- The schema that features follow can change over time.
 - If don't have a way to version schemas and map each of features to its expected schema,
 - the cause of the reported alert might be due to the mismatched schema rather than a change in the data

Monitoring raw inputs

- A change in the features might be caused by problems in processing steps and not by changes in data
 - can monitor the raw inputs before they are processed
 - not be easier to monitor, as it can come from multiple sources in different formats, following multiple structures
- The way many ML workflows are set up today also makes it impossible for ML engineers to get direct access to raw input data,
 - as the raw input data is often managed by a data platform team who processes and moves the data to a location like a data warehouse,
- ML engineers can only query for data from that data warehouse where the data is already partially processed
- Monitoring raw inputs is often a responsibility of the data platform team, not the data science or ML team

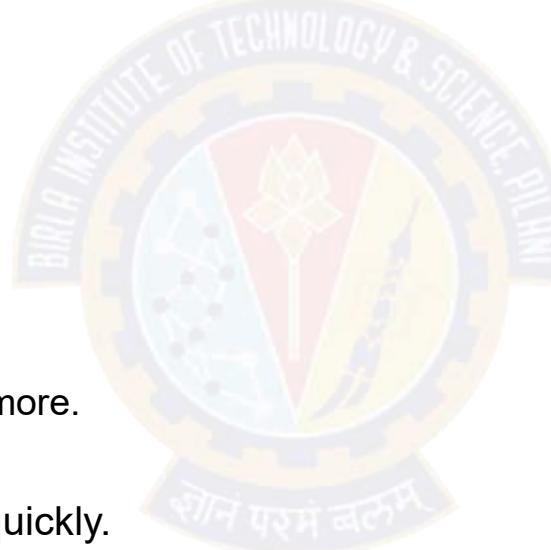
Monitoring Toolbox

- Measuring, tracking, and interpreting metrics for complex systems is a nontrivial task
 - engineers rely on a set of tools to help them do so
- Common for the industry to herald **metrics, logs, and traces** as the three pillars of monitoring
- Seem to be generated from the perspective of people who develop monitoring systems:
 - traces are a form of logs and metrics can be computed from logs
- Focus on the set of tools from the perspective of users of the monitoring systems:
 - logs,
 - dashboards,
 - alerts

Monitoring Toolbox(2)

Logs

- Traditional software systems rely on logs to record events produced at runtime
 - An event is anything that can be of interest to the system developers,
 - either at the time the event happens or later for debugging and analysis purposes
- Examples of events
 - a container starts,
 - the amount of memory it takes,
 - when a function is called,
 - when that function finishes running,
 - the other functions that this function calls,
 - the input and output of that function,
 - log crashes, stack traces, error codes, and more.
- The number of logs can grow very large very quickly.
 - need to query your logs for the sequence of events that caused it, a process that can feel like searching for a needle in a haystack
- Because logs have grown so large and so difficult to manage,
 - there have been many tools developed to help companies manage and analyze logs
 - The log management market is estimated to be worth USD 2.3 billion in 2021, and it's expected to grow to USD 4.1 billion by 2026



Monitoring Toolbox(3)

Dashboards

- A picture is worth a thousand words!
 - A series of numbers might mean nothing, but visualizing them on a graph might reveal the relationships among these numbers
 - Dashboards to visualize metrics are critical for monitoring
- Another use of dashboards is to make monitoring accessible to non-engineers.
 - Monitoring isn't just for the developers of a system, but also for non-engineering stakeholders
 - including product managers and business developers
- Dashboard rot
 - Excessive metrics on a dashboard can also be counterproductive
 - important to pick the right metrics
 - abstract out lower-level metrics to compute higher-level signals that make better sense for specific tasks

Monitoring Toolbox(4)

Alerts

- When monitoring system detects something suspicious, it's necessary to alert the right people about it
- An alert consists of the following three components:
 - An alert policy
 - describes the condition for an alert
 - might want to create an alert when a metric breaches a threshold, optionally over a certain duration
 - Notification channels
 - describe who is to be notified when the condition is met
 - A description of the alert
 - helps the alerted person understand what's going on with a detailed description
 - often necessary to make the alert actionable
 - by providing mitigation instructions or a runbook, a compilation of routine procedures and operations that might help with handling the alert
- Alert fatigue is a real phenomenon!
 - can be demoralizing—nobody likes to be awakened in the middle of the night for something outside of their responsibilities
 - can be dangerous —being exposed to trivial alerts can desensitize people to critical alerts
 - important to set meaningful conditions so that only critical alerts are sent out



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

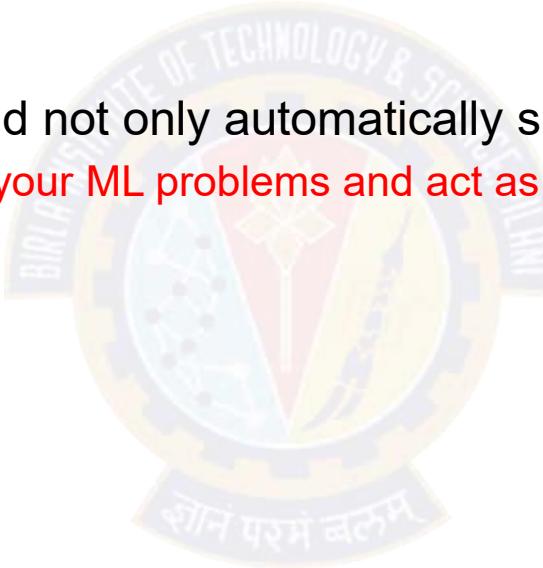
Machine Learning Observability

Pravin Y Pawar

Adapted from "[Machine Learning Observability](#)"

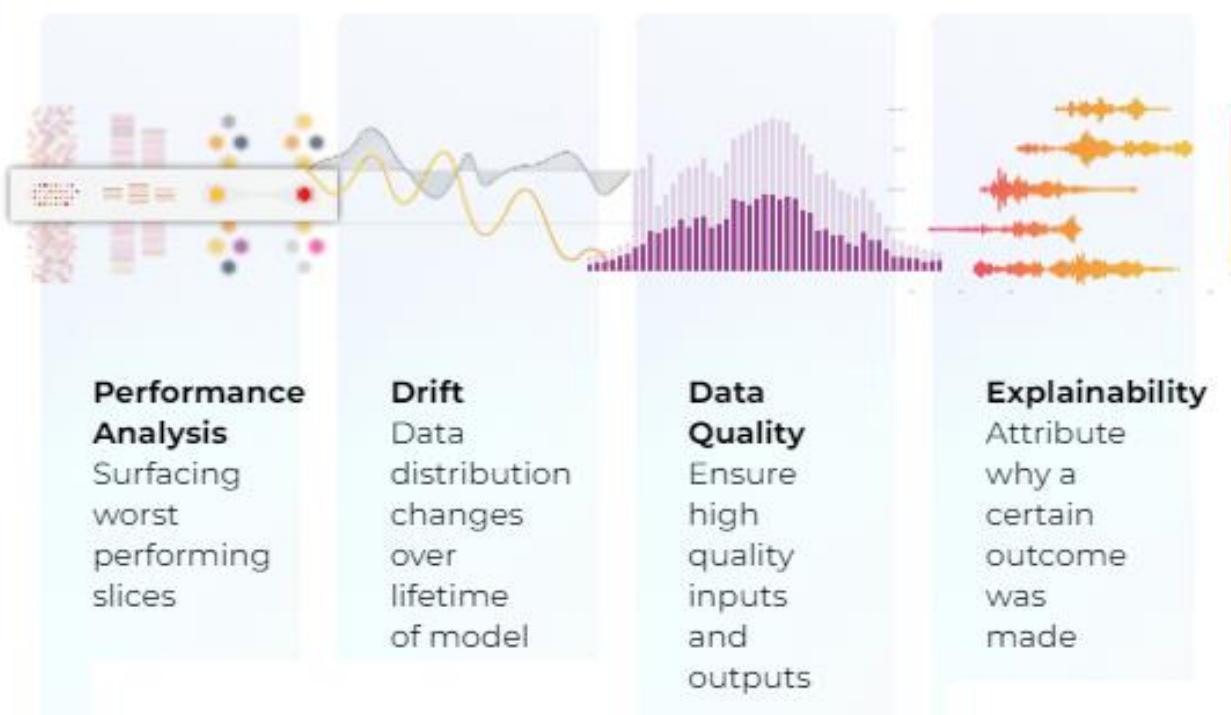
What is ML Observability?

- ML Observability is a tool used to monitor, troubleshoot, and explain machine learning models
 - as they move from research to production environments
- An effective observability tool should not only automatically surface issues,
 - but drill down to the root cause of your ML problems and act as a guardrail for models in production



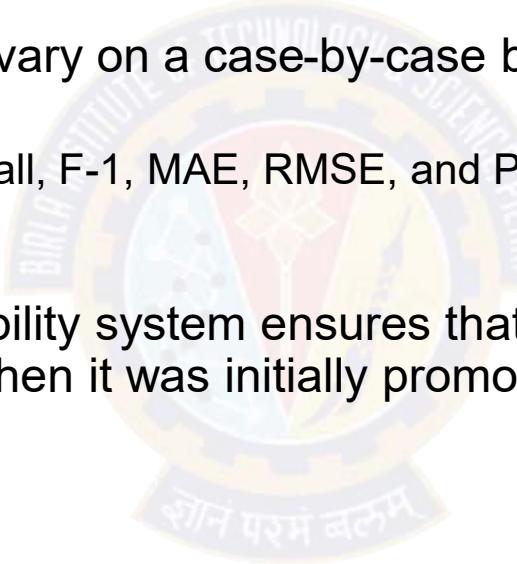
4 Pillars of ML Observability

ML Observability in practice



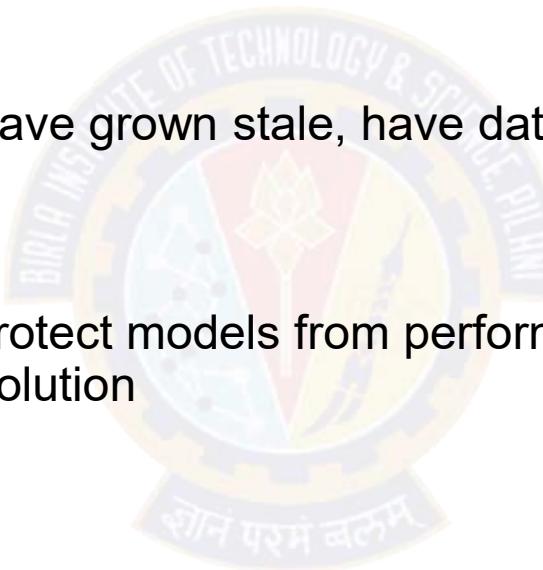
Performance Analysis

- ML observability enables fast actionable performance information on models deployed in production
- While performance analysis techniques vary on a case-by-case basis depending on model type and its use case in the real world,
 - common metrics include: Accuracy, Recall, F-1, MAE, RMSE, and Precision
- Performance analysis in an ML observability system ensures that performance has not degraded drastically from when it was trained or when it was initially promoted to production.



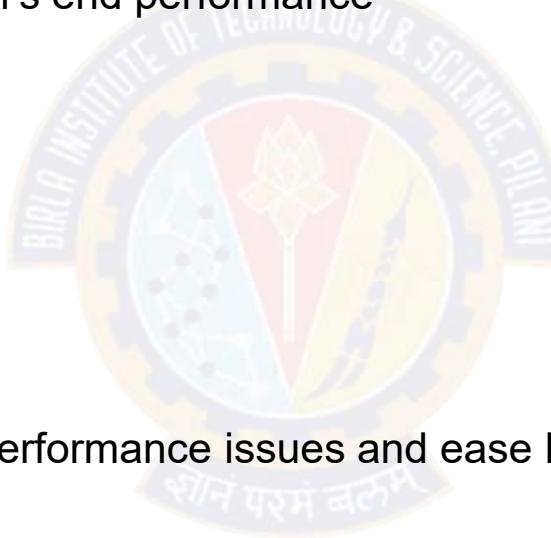
Drift

- ML observability encompasses drift to monitor for a change in distribution over time,
 - measured for model inputs, outputs, and actuals of a model
- Measure drift to identify if models have grown stale, have data quality issues, or if there are adversarial inputs in model
- Detecting drift in models will help protect models from performance degradation and allow to better understand how to begin resolution



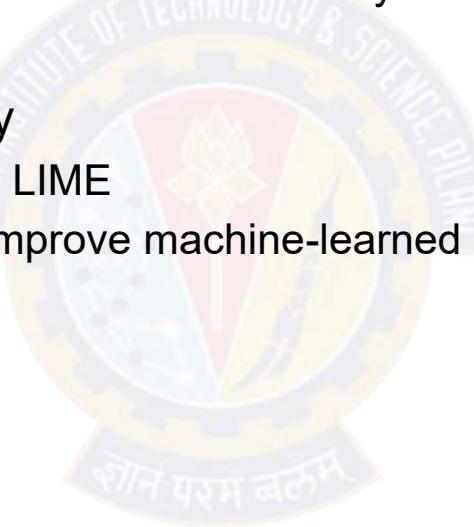
Data Quality

- Data quality checks in an ML observability system identify hard failures
 - within data pipelines between training and production
 - that can negatively impact a model's end performance
- Data quality includes
 - monitoring for cardinality shifts,
 - missing data,
 - data type mismatch,
 - out-of-range,
 - and more to better gauge model performance issues and ease RCA



Explainability

- Explainability in ML observability uncovers feature importance
 - across training, validation, and production environments
 - which provides the ability to introspect and understand why a model made a particular prediction
- Explainability is commonly achieved by
 - calculating metrics such as SHAP and LIME
 - to build confidence and continuously improve machine-learned models



ML observability in context

System/Infra Observability

- Infra/App timing as the base of monitoring
- App & system response time issues
- Tracing & troubleshooting response time



dynatrace

Data Observability

- Tables as the base of monitoring
- Monitoring data changes
- Schema monitoring

MONTE CARLO

Bigeye

ML Observability

- Models are the base of monitoring
- Distributions, vs baselines, model version, SHAP analysis and performance
- Deep model performance analysis vs data



Software/DevOps



Data Eng



ML Engineer & DS

[arize](#)

How Can I Achieve ML Observability?

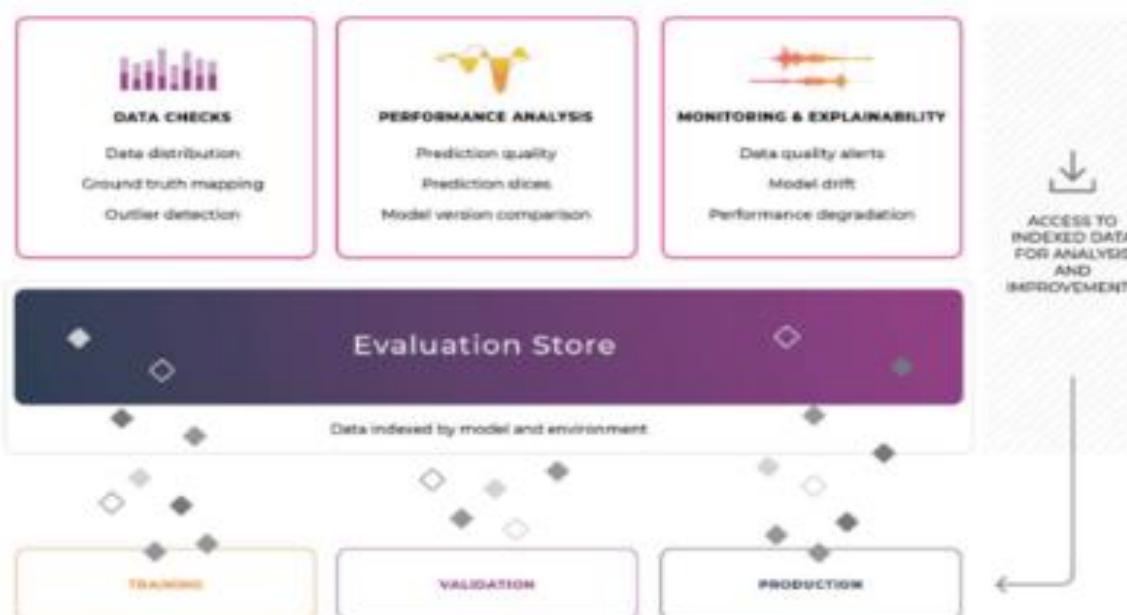
ML Observability with an Evaluation Store

Monitor drift, data quality issues, or anomalous performance degradations using baselines

Analyze performance metrics in aggregate (or slice) for any model, in any environment — production, validation, training

Root Cause Analysis to connect changes in performance to why they occurred

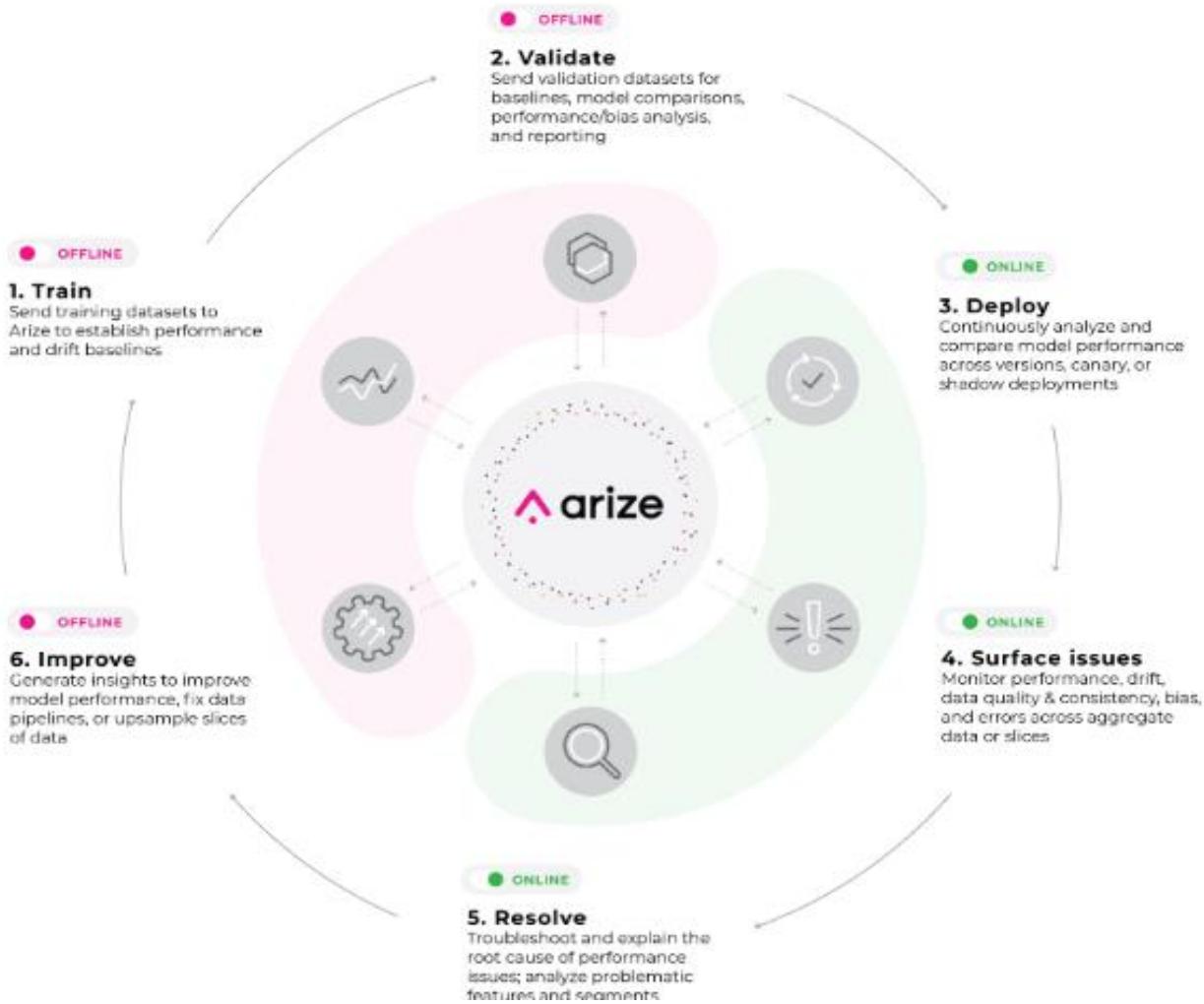
Enable feedback loop to **actively improve** model performance



arize

ML Observability with Arize

- ML Observability is the practice of obtaining a deep understanding into model's data and performance across its lifecycle
 - Observability doesn't just stop at surfacing a red or green light,
 - but enables ML practitioners to root cause/explain why a model is behaving a certain way in order to improve it





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

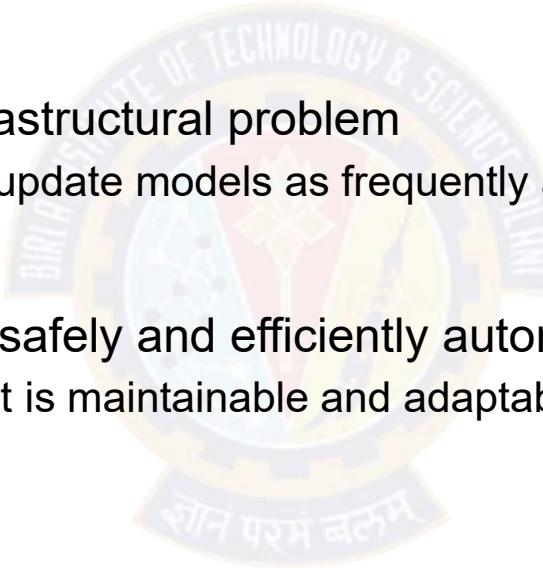
Continual Learning

Pravin Y Pawar

Adapted from “Designing Machine Learning Systems”
By Chip Huyen

Continual Learning

- How do adapt models to data distribution shifts?
 - The answer is by continually updating our ML models
- Continual learning is largely an infrastructural problem
 - Setting up infrastructure allows to update models as frequently as required
- The goal of continual learning is to safely and efficiently automate the update
 - allows to design an ML system that is maintainable and adaptable to changing environments



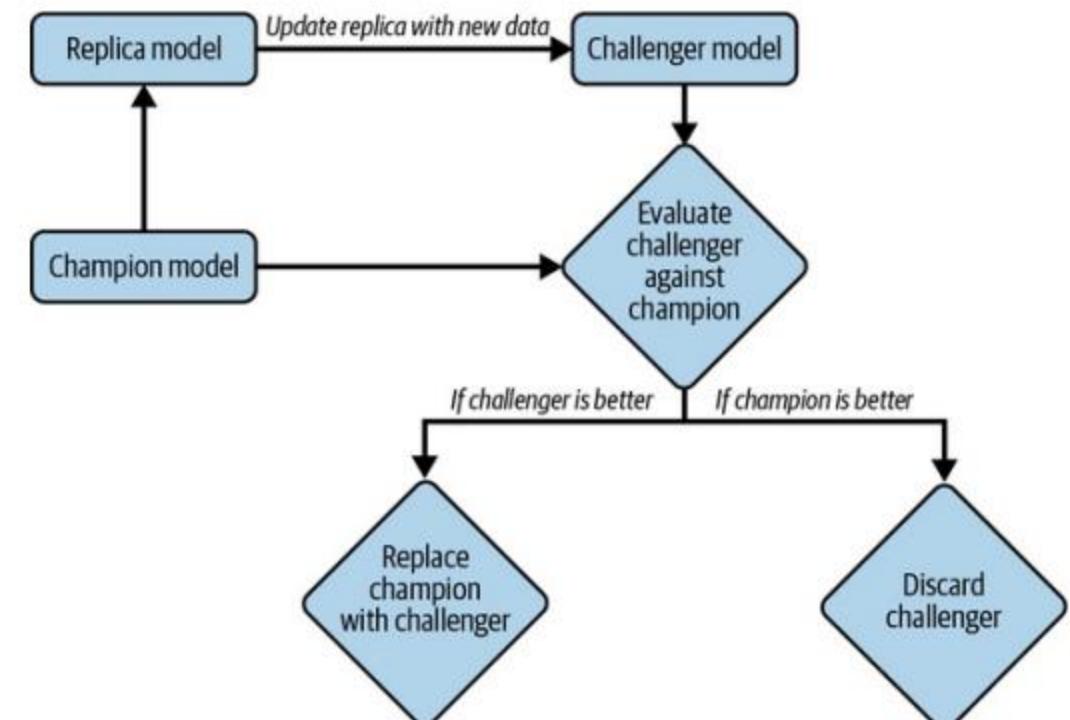
What is continual learning?

- Many people think of the training paradigm where a model updates itself with every incoming sample in production
 - people imagine updating models very frequently, such as every 5 or 10 minutes
- Very few companies actually do that!
 - makes model susceptible to catastrophic forgetting - the tendency of a neural network to completely and abruptly forget previously learned information upon learning new information
 - make training more expensive - most hardware backends today were designed for batch processing, so processing only one sample at a time causes a huge waste of compute power and is unable to exploit data parallelism
- Most companies don't need to update their models that frequently because of two reasons
 - First, they don't have enough traffic (i.e., enough new data) for that retraining schedule to make sense
 - Second, their models don't decay that fast
 - If changing retraining schedule from a week to a day gives no return and causes more overhead, there's no need to do it.

A simplification of how continual learning work in production

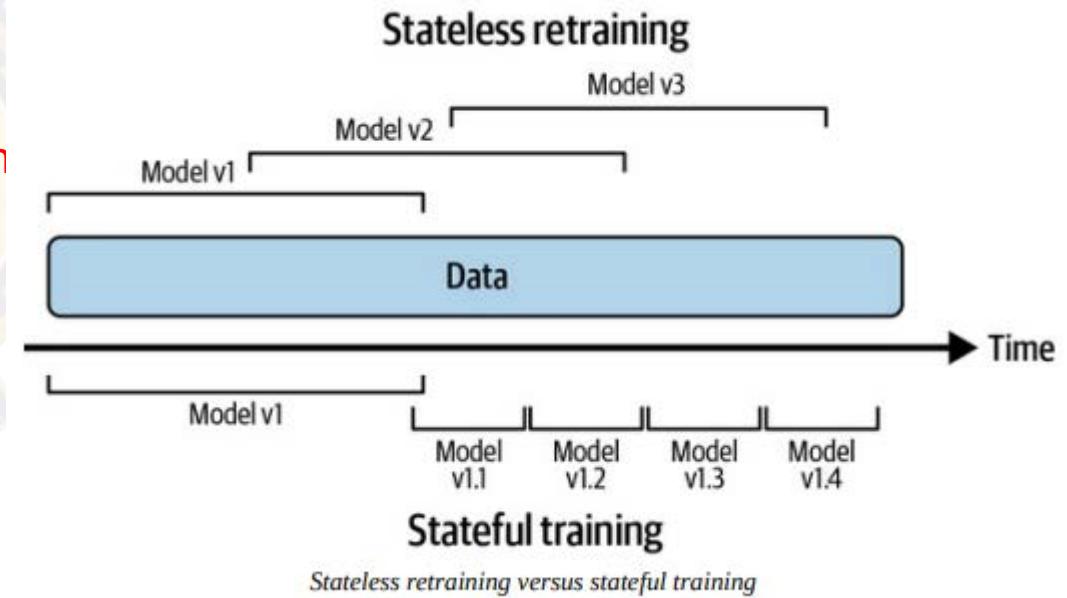
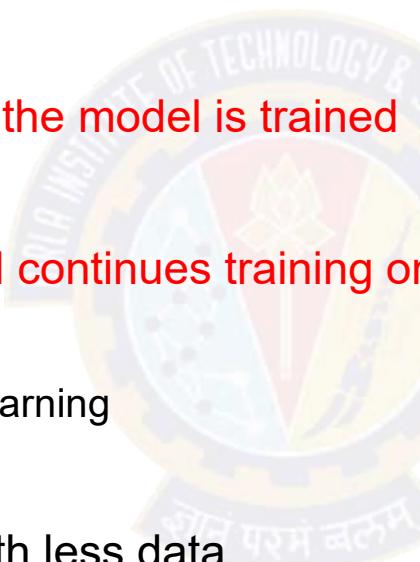
Champion-Challenger

- The updated model shouldn't be deployed until it's been evaluated
 - means that shouldn't make changes to the existing model directly
- Instead,
 - should create a replica of the existing model
 - update this replica on new data
 - only replace the existing model with the updated replica if
 - the updated replica proves to be better
- The existing model - champion model
- The updated replica - the challenger
- An oversimplification of the process for the sake of understanding!
 - In reality, a company might have multiple challengers at the same time



Stateless Retraining Versus Stateful Training

- Continual learning isn't about the retraining frequency,
 - but the manner in which the model is retrained
- Most companies do **stateless retraining** — the model is trained from scratch each time
- Some follows **stateful training** — the model continues training on new data
 - also known as fine-tuning or incremental learning
- Stateful training allows to update model with less data
- Training a model from scratch tends to require a lot more data
- One beautiful property that is often overlooked is that with stateful training,
 - it might be possible to avoid storing data altogether



Stateless Retraining Versus Stateful Training(2)

- The companies that have most successfully used stateful training
 - also occasionally train their model from scratch on a large amount of data to calibrate it
- Once infrastructure is set up to allow both stateless retraining and stateful training,
 - the training frequency is just a knob to twist
 - can update models once an hour, once a day, or whenever a distribution shift is detected
- Continual learning is about **setting up infrastructure** in a way that
 - allows a data scientist or ML engineer,
 - to **update models whenever it is needed**,
 - whether **from scratch or fine-tuning**,
 - to **deploy this update quickly**

Model iteration vs Data iteration

- Stateful training sounds cool, but **how does this work if needs to add a new feature or another layer to model?**
- Must differentiate **two types of model updates**:
 - Model iteration - A new feature is added to an existing model or the model architecture is changed
 - Data iteration - Model architecture and features remain the same, but **refresh this model with new data.**
- As of today, stateful training is mostly applied for data iteration!
- Changing model architecture or adding a new feature still requires training the resulting model from scratch
 - research shows that it might be possible to bypass training from scratch for model iteration
 - by using techniques such as knowledge transfer (Google, 2015) and model surgery (OpenAI, 2019)

Why Continual Learning?

Why would you need the ability to update your models as fast as you want?

- The first use case of continual learning is to combat data distribution shifts,
 - especially when the shifts happen suddenly
- Another use case of continual learning is to adapt to rare events
- Can help overcome is the continuous cold start problem

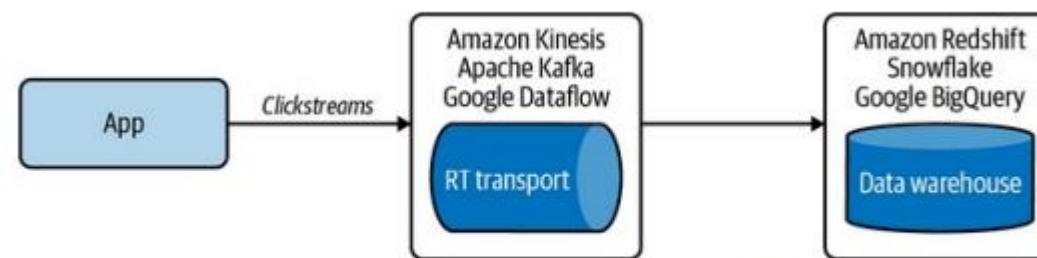
- “Why continual learning?” should be rephrased as “why not continual learning?”
 - Continual learning is a **superset of batch learning** - allows to do everything traditional batch learning can do
 - If continual learning **takes the same effort to set up and costs the same to do as batch learning**,
 - there’s no reason not to do continual learning!

- **MLOps tooling for continual learning is maturing**, which means, one day not too far in the future,
 - it might be as easy to set up continual learning as batch learning

Continual Learning Challenges

Fresh data access challenge - Data Sources

- If want to update model every hour, need new data every hour!
- Data Sources
 - Currently, many companies pull new training data from their data warehouses
 - The speed at which data can be pulled from data warehouses depends on the speed at which this data is deposited into data warehouses
 - The speed can be slow, especially if data comes from multiple sources
 - An alternative is to allow pull data before it's deposited into data warehouses,
 - e.g., directly from real-time transports such as Kafka and Kinesis that transport data from applications to data warehouses



Pulling data directly from real-time transports, before it's deposited into data warehouses, can allow you to access fresher data

Continual Learning Challenges(2)

Fresh data access challenge - Labelling

- If model needs labeled data to update, data will need to be labeled as well!
 - In many applications, the speed at which a model can be updated is bottlenecked by the speed at which data is labeled
- The best candidates for continual learning are tasks where can get natural labels with short feedback loops
 - dynamic pricing, estimating time of arrival, stock price prediction,
 - ads click-through prediction, and recommender systems
- If model's speed iteration is bottlenecked by labeling speed,
 - possible to speed up the labeling process by leveraging programmatic labeling tools like Snorkel to generate fast labels
 - possible to leverage crowdsourced labels to quickly annotate fresh data
- Given that tooling around streaming is still nascent,
 - architecting an efficient streaming-first infrastructure for accessing fresh data and extracting fast labels from real-time transports can be engineering-intensive and costly

Continual Learning Challenges(3)

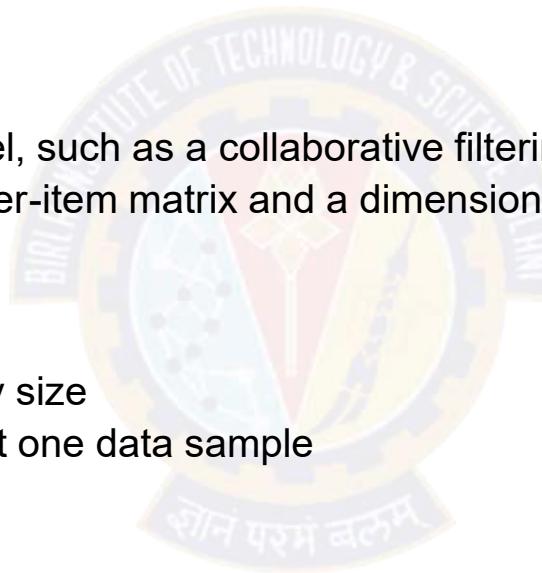
Evaluation challenge

- ML systems make catastrophic failures in production,
 - from millions of minorities being unjustly denied loans,
 - to drivers who trust autopilot too much being involved in fatal crashes
- The biggest challenge is in making sure that this update is good enough to be deployed!
- The risks for catastrophic failures amplify with continual learning
 - More frequently models are updated, the more opportunities there are for updates to fail
 - Makes models more susceptible to coordinated manipulation and adversarial attack
- To avoid similar or worse incidents, it's crucial to thoroughly test each of model updates to ensure its performance and safety before deploying the updates to a wider audience.
- When designing the evaluation pipeline for continual learning, keep in mind that evaluation takes time,
 - which can be another bottleneck for model update frequency.

Continual Learning Challenges(4)

Algorithm challenge - “softer” challenge

- Challenge as it only affects certain algorithms and certain training frequencies.
 - only affects matrix-based and tree-based models that want to be updated very fast (e.g., hourly)
- Consider two different models:
 - a neural network and a matrix-based model, such as a collaborative filtering model
 - The collaborative filtering model uses a user-item matrix and a dimension reduction technique
- A neural network model
 - Can update model with a data batch of any size
 - Can even perform the update step with just one data sample
- Collaborative filtering model
 - For updating model, first need to use the entire dataset to build the user-item matrix before performing dimensionality reduction on it
 - if matrix is large, the dimensionality reduction step would be too slow and expensive to perform frequently
 - less suitable for learning with a partial dataset than the preceding neural network model



Four Stages of Continual Learning

- How to overcome these challenges and make continual learning happen?
- The move toward continual learning happens in four stages
 - Stage 1: Manual, stateless retraining
 - Stage 2: Automated retraining
 - Stage 3: Automated, stateful training
 - Stage 4: Continual learning



Four Stages of Continual Learning(2)

Stage 1: Manual, stateless retraining

- In the beginning, the ML team often focuses on developing ML models to solve as many business problems as possible
 - Because team is focusing on developing new models, updating existing models takes a backseat
 - No fixed frequency to update the models
- Update an existing model only when the following two conditions are met:
 - the model's performance has degraded to the point that it's doing more harm than good
 - team has time to update it
- The process of updating a model is manual and ad hoc
 - Someone, usually a data engineer, has to query the data warehouse for new data.
 - Someone else cleans this new data, extracts features from it, retrains that model from scratch on both the old and new data, and then exports the updated model into a binary format.
 - Someone else takes that binary format and deploys the updated model.
- A vast majority of companies outside the tech industry—
 - e.g., any company that adopted ML less than three years ago and doesn't have an ML platform team—are in this stage

Four Stages of Continual Learning(3)

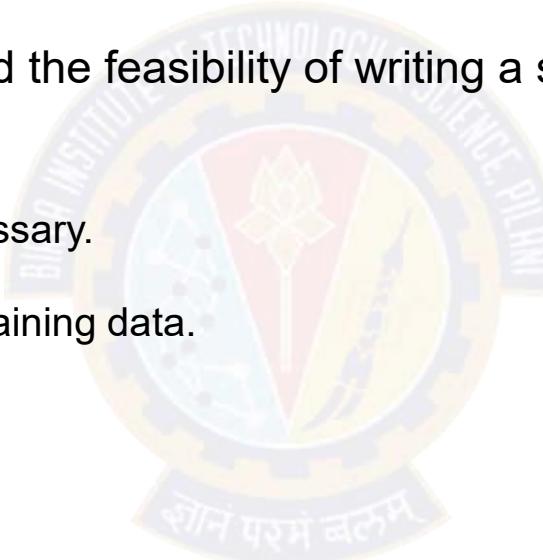
Stage 2: Automated retraining

- After a few years, team has managed to deploy models to solve most of the obvious problems
 - Priority is no longer to develop new models, but to maintain and improve existing models
 - The ad hoc, manual process of updating models mentioned from the previous stage has grown into a pain point too big to be ignored
- Team decides to write a script to automatically execute all the retraining steps
 - run periodically using a batch process such as Spark
- Status
 - Most companies with somewhat mature ML infrastructure are in this stage
 - Some sophisticated companies run experiments to determine the optimal retraining frequency
 - For most companies in this stage, the retraining frequency is set based on gut feeling
 - e.g., “once a day seems about right” or “let’s kick off the retraining process each night when we have idle compute”

Four Stages of Continual Learning(4)

Stage 2: Automated retraining - Requirements for setup

- If company has ML models in production,
 - likely that company already has most of the infrastructure pieces needed for automated retraining
- The feasibility of this stage revolves around the feasibility of writing a script to automate workflow and configure infrastructure to automatically:
 1. Pull data.
 2. Downsample or upsample this data if necessary.
 3. Extract features.
 4. Process and/or annotate labels to create training data.
 5. Kick off the training process.
 6. Evaluate the newly trained model.
 7. Deploy it.
- In general, the three major factors that will affect the feasibility of this script are:
 - scheduler,
 - data,
 - and model store



Four Stages of Continual Learning(4)

Stage 3: Automated, stateful training

- Need to reconfigure automatic updating script so that, when the model update is kicked off
 - it first locates the previous checkpoint and loads it into memory before continuing training on this checkpoint
- Requirements
 - The main thing needed at this stage is a way to track data and model lineage
- Imagine you first upload model version 1.0
 - is updated with new data to create model version 1.1, and so on to create model 1.2
 - Another model is uploaded and called model version 2.0
 - is updated with new data to create model version 2.1
 - After a while, you might have model version 3.32, model version 2.11, model version 1.64
- Might want to know
 - how these models evolve over time,
 - which model was used as its base model,
 - which data was used to update it so that you can reproduce and debug it
- Need model store that has this model lineage capacity

Four Stages of Continual Learning(5)

Stage 4: Continual learning

- At stage 3, models are still updated based on a fixed schedule set out by developers
 - Finding the optimal schedule isn't straightforward and can be situation-dependent
 - might want models to be automatically updated whenever data distributions shift and the model's performance plummets
 - The move from stage 3 to stage 4 is steep!
- Requirements
 - First need a mechanism to trigger model updates
 - Time-based - every five minutes
 - Performance-based - whenever model performance plummets
 - Volume-based - whenever the total amount of labeled data increases by 5%
 - Drift-based - whenever a major data distribution shift is detected
 - For this trigger mechanism to work, need a solid monitoring solution
 - Hard part is not to detect the changes, but to determine which of these changes matter
 - If monitoring solution gives a lot of false alerts, model will end up being updated much more frequently than it needs to be
 - Need a solid pipeline to continually evaluate model updates.
 - Writing a function to update models isn't much different from stage 3
 - Hard part is to ensure that the updated model is working properly

How Often to Update Your Models?

- Depends on how much gain model will get from being updated with fresh data
 - The more gain model can get from fresher data, the more frequently it should be retrained
- The question on **how often to update your model is a difficult one to answer!**
 - In the **beginning**, when **infrastructure** is nascent and the **process** of updating a model is **manual** and **slow**, the answer is: **as often as you can**.
 - As **infrastructure matures** and the **process** of updating a model is partially **automated** and can be done in a matter of **hours, if not minutes**,
 - the answer to this question is contingent on the answer to the following question:
 - “How much performance gain would I get from fresher data?”
- Points to be considered
 - **Value of data freshness**
 - **Model iteration versus data iteration**

Value of data freshness

- The question of how often to update a model becomes a lot easier
 - if knows how much the model performance will improve with updating
- For example,
 - If switch from retraining model every month to every week, how much performance gain can we get?
 - What if we switch to daily retraining?
- People keep saying that data distributions shift, so fresher data is better,
 - but how much better is fresher data?
 - One way to figure out the gain is by training model on the data from different time windows in the past and evaluating it on the data from today to see how the performance changes.

Model iteration versus data iteration

- Not all model updates are the same - can be model iteration or data iteration!
- Might wonder not only how often to update model, but also what kind of model updates to perform
 - In theory, can do both types of updates
 - In practice, should do both from time to time
- The more resources spent in one approach, the fewer resources can be spent in another
 - if iterating on data doesn't give much performance gain, then should spend resources on finding a better model
 - if finding a better model architecture requires 100X compute for training and gives 1% performance
 - whereas updating the same model on data from the last three hours requires only 1X compute and also gives 1% performance gain, will be better off iterating on data
- Currently no book can give the answer on which approach will work better for specific model on specific task
 - have to do experiments to find out.



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Test In Production

Pravin Y Pawar

Adapted from “Designing Machine Learning Systems”
By Chip Huyen

Offline Evaluation

- An offline model evaluation happens when the model is being trained by the analyst
- The analyst tries out different
 - features,
 - models,
 - algorithms,
 - and hyperparameters
- Model training is guided in the right direction by
 - Tools like confusion matrix
 - Various performance metrics, such as precision, recall, and AUC
- Process
 - First, validation data is used to assess the chosen performance metric and compare models
 - Once the best model is identified, the test set is used, also in offline mode, to again assess the best model's performance
 - This final offline assessment guarantees post-deployment model performance



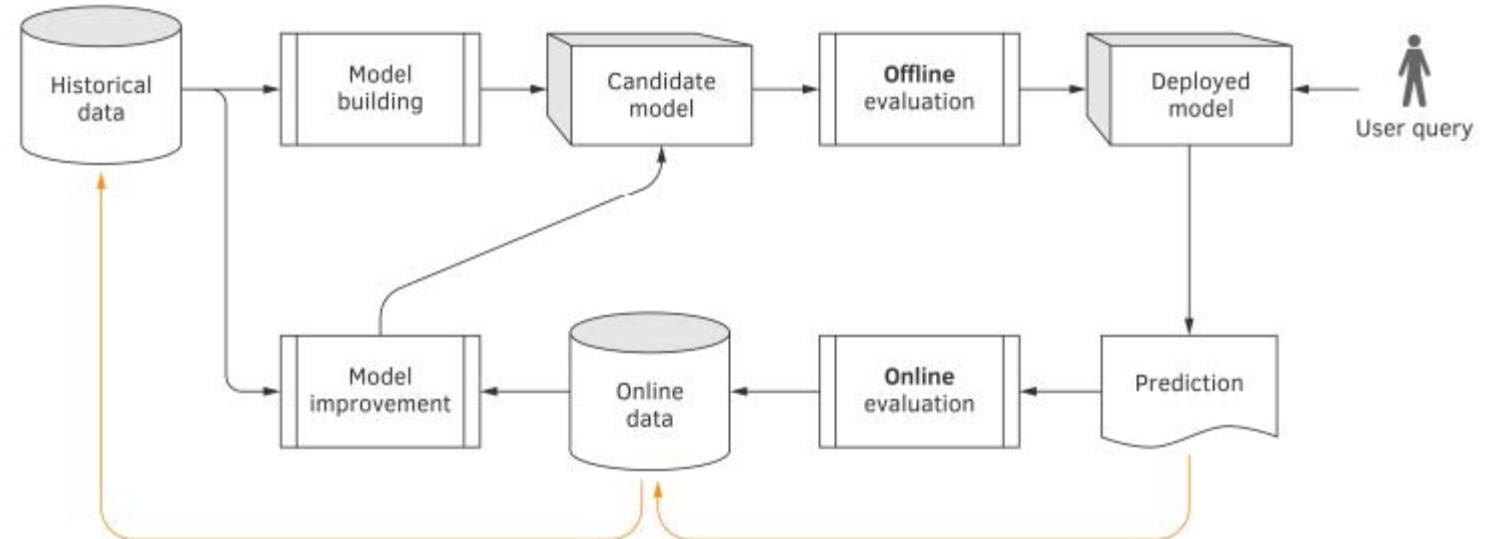
Offline Evaluation(2)

Two major test types for offline evaluation

- Test splits
 - are usually static and have to be static so that you have a trusted benchmark to compare multiple models
 - will be hard to compare the test results of two models if they are tested on different test sets
- Backtests
 - method of testing a predictive model on data from a specific period of time in the past
 - If model is updated to adapt to a new data distribution, it's not sufficient to evaluate this new model on test splits from the old distribution
 - one idea is to test model on the most recent data that you have access to - after updated model on the data from the last day, might want to test this model on the data from the last hour
- **The question is whether backtests are sufficient to replace static test splits. Not quite!**
 - If something went wrong with data pipeline and some data from the last hour is corrupted, evaluating model solely on this recent data isn't sufficient.
 - With backtests, should still evaluate model on a static test set that have been extensively studied and (mostly) trust as a form of sanity check.

Offline and Online Evaluation

- Historical data is first used to train a deployment candidate
 - Then it is evaluated offline
 - If the result is satisfactory, deployment candidate becomes the deployed model, and starts accepting user queries
- User queries and the model predictions are used for an online evaluation of the model
 - The online data is then used to improve the model
 - To close the loop, the online data is permanently copied to the offline data repository



The placement of offline and online model evaluations in a machine learning system.

Source: Machine Learning Engineering by Burkov

Why do we evaluate both offline and online?

- The offline model evaluation reflects how well the analyst succeeded
 - in finding the right features, learning algorithm, model, and values of hyperparameters
 - reflects how good the model is from an engineering standpoint
- Online evaluation, focuses on measuring business outcomes,
 - such as customer satisfaction, average online time, open rate, and click-through rate
 - may not be reflected in historical data, but it's what the business really cares about
- Offline evaluation doesn't allow us to test the model in some conditions that can be observed online,
 - such as connection and data loss, and call delays

Test in production

Aka Online Evaluation

- The only way to know whether a model will do well in production is to deploy it
 - led to one seemingly terrifying but necessary concept
- Techniques to help to evaluate models in production (mostly) safely
 - Shadow deployment
 - A/B testing
 - Canary analysis
 - Interleaving experiments
 - Bandits
- To sufficiently evaluate models, first need a mixture of offline evaluation and online evaluation!



Shadow Deployment

Might be the safest way to deploy model or any software update

- Works as follows
 1. Deploy the candidate model in parallel with the existing model
 2. For each incoming request, route it to both models to make predictions, but only serve the existing model's prediction to the user
 3. Log the predictions from the new model for analysis purposes.
- Only when found that the new model's predictions are satisfactory do replace the existing model with the new model!
- The risk of this new model doing something funky is low
 - as don't serve the new model's predictions to users until made sure that the model's predictions are satisfactory
- Always not favorable because it's expensive
 - doubles the number of predictions system has to generate, which generally means doubling inference compute cost

A/B Testing

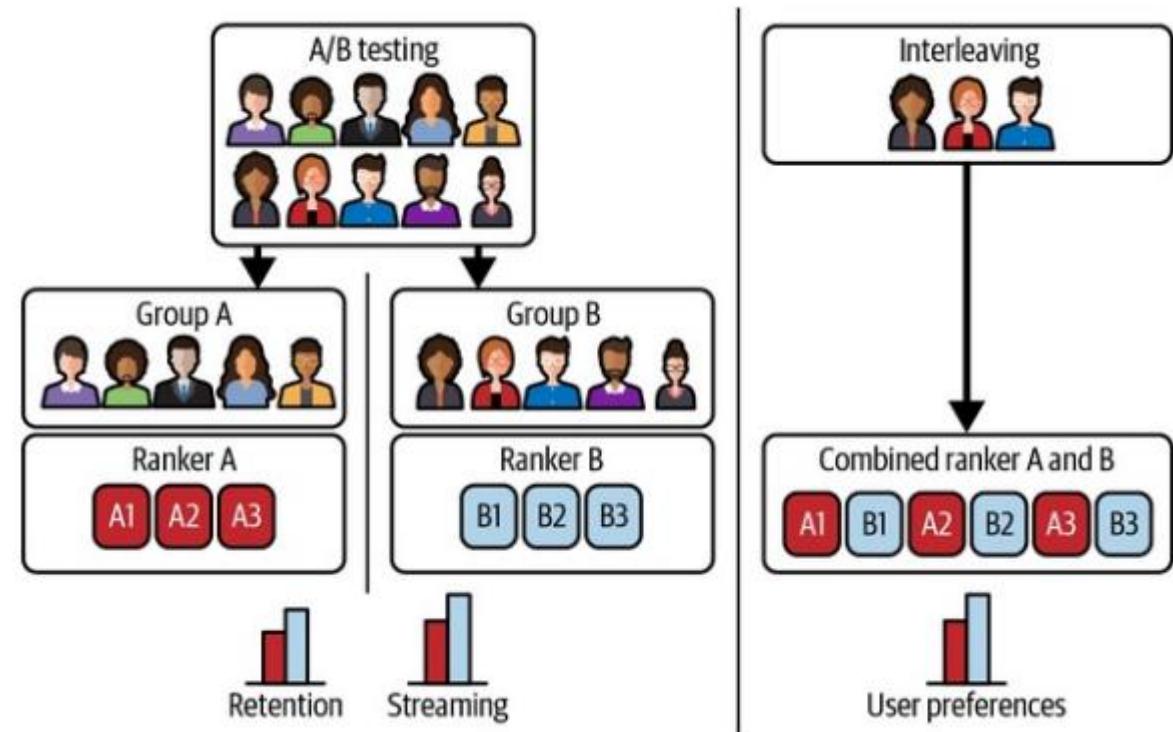
- A way to compare two variants of an object, typically by testing responses to these two variants, determining which of the two variants is more effective
 - the existing model as one variant, and the candidate model (the recently updated model) as another variant
- Works as follows:
 1. Deploy the candidate model alongside the existing model
 2. A percentage of traffic is routed to the new model for predictions; the rest is routed to the existing model for predictions.
 3. Monitor and analyze the predictions and user feedback, if any, from both models to determine whether the difference in the two models' performance is statistically significant.
- Two important things:
 - First, A/B testing consists of a randomized experiment: the traffic routed to each model has to be truly random.
 - If not, the test result will be invalid
 - Second, A/B test should be run on a sufficient number of samples to gain enough confidence about the outcome.
 - How to calculate the number of samples needed for an A/B test is a simple question with a very complicated answer.
- Often, in production, you don't have just one candidate but multiple candidate models
 - possible to do A/B testing with more than two variants, which means can have A/B/C testing or even A/B/C/D testing

Canary Release

- A technique to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users
 - before rolling it out to the entire infrastructure and making it available to everybody
- Works as follows:
 1. Deploy the candidate model alongside the existing model. The candidate model is called the canary
 2. A portion of the traffic is routed to the candidate model.
 3. If its performance is satisfactory, increase the traffic to the candidate model.
If not, abort the canary and route all the traffic back to the existing model.
 1. Stop when either the canary serves all the traffic (the candidate model has replaced the existing model) or when the canary is aborted
- The candidate model's performance is measured against the existing model's performance according to the metrics you care about
 - If the candidate model's key metrics degrade significantly, the canary is aborted and all the traffic will be routed to the existing model
- Canary releases can be used to implement A/B testing due to the similarities in their setups
 - don't have to randomize the traffic to route to each model
 - first roll out the candidate model to a less critical market before rolling out to everybody

Interleaving Experiments

- Imagine two recommender systems, A and B, and want to evaluate which one is better
 - Each time, a model recommends 10 items users might like
- With A/B testing, divide users into two groups: one group is exposed to A and the other group is exposed to B.
 - Each user will be exposed to the recommendations made by one model
- What if instead of exposing a user to recommendations from a model, **we expose that user to recommendations from both models** and see which model's recommendations they will click on?



An illustration of interleaving versus A/B testing. Source: Adapted from an image by Parks.

Interleaving Experiments(2)

- In A/B testing, core metrics like retention and streaming are measured and compared between two groups
- In interleaving, two algorithms can be compared by measuring user preferences,
 - no guarantee that user preference will lead to better core metrics
- Netflix found that interleaving “reliably identifies the best algorithms with considerably smaller sample size compared to traditional A/B testing.”
- When recommendations from multiple models are shown to users,
 - the position of a recommendation influences how likely a user will click on it
 - users are much more likely to click on the top recommendation than the bottom recommendation
- For interleaving to yield valid results, must ensure that at any given position,
 - a recommendation is equally likely to be generated by A or B.

Bandits

- Bandit algorithms originated in gambling!
- A slot machine is also known as a one-armed bandit
 - A casino has multiple slot machines with different payouts
 - don't know which slot machine gives the highest payout
 - Can experiment over time to find out which slot machine is the best while maximizing payout
- Multi-armed bandits are algorithms
 - allow you to balance between
 - exploitation (choosing the slot machine that has paid the most in the past)
 - and exploration (choosing other slot machines that may pay off even more)
- When multiple models needs to be evaluated, each model can be considered a slot machine whose payout (i.e., prediction accuracy) is unknown
- Bandits allow to determine how to route traffic to each model for prediction to determine the best model while maximizing prediction accuracy for users

Bandits(2)

- Bandit is stateful: before routing a request to a model, need to calculate all models' current performance
- Requires three things:
 - Model must be able to make online predictions
 - Preferably short feedback loops
 - A mechanism to collect feedback, calculate and keep track of each model's performance, and route prediction requests to different models based on their current performance
- Bandits are well-studied in academia and have been shown to be a lot more data efficient than A/B testing
 - require less data to determine which model is the best
 - at the same time, reduce opportunity cost as they route traffic to the better model more quickly
 - But a lot more difficult to implement than A/B testing because it requires computing and keeping track of models' payoffs



Thank You!

In our next session:



Presentation

Model Monitoring, Drift Detection, Re-training
and Continuous Learning

Abhishek Singh
Rajendra Mishra

BITS Pilani
Pilani Campus



Agenda

- Introduction
- Dashboard Tools
- Infrastructure monitoring
- ML Model Monitoring
- Model Training Monitoring
- Model Tuning Monitoring
- Model Serving resources monitoring
- Model Serving Monitoring
- Model re-training and deployment
- Model Re-training Policy
- Stateless Vs Stateful model re-training
- Four Stages of Continuous Learning
- Summary
- Python Packages available to detect drifts
- References
- Demo

Introduction

In machine learning system lifecycle, an important step begins, once we deploy our models to production.

Traditionally, with rule based, deterministic, software, the majority of the work occurs at the initial stage and once deployed, our system works as we've defined it.

But with machine learning, we haven't explicitly defined how something works but used data to architect a ***probabilistic solution***.

This approach is subject to **natural performance degradation over time**, as well as unintended behaviour, **since gradually the data exposed to the model will be different from what it has been trained on**. This isn't something we should be trying to avoid but rather **understand and mitigate as much as possible**.

In addition, there are **many aspects of the machine learning system that we need to monitor** to ensure that our machine learning use cases works as expected.

- Infrastructure monitoring
- ML Model Monitoring
 1. Model Training Monitoring
 2. Model Serving Monitoring
- Model Serving resources monitoring

Dashboard Tools

Dashboard tools play a crucial role in the MLOps lifecycle by providing insights into model performance, detecting potential issues, and ensuring that machine learning systems remain robust and efficient.

Prometheus

Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability. It is widely used for monitoring both machine learning models and infrastructure components, making it an essential tool for MLOps. Key features of Prometheus include:

- Flexible data model and powerful query language (PromQL), enabling users to define custom metrics and perform complex queries to gain insights into their systems
- Pull-based data collection, allowing Prometheus to scrape metrics from multiple sources, such as applications, databases, and other infrastructure components
- Built-in alerting system, enabling users to define custom alert rules and receive notifications when specific conditions are met
- Integration with popular visualization tools, such as Grafana, providing users with customizable dashboards and visualizations of their monitoring data

By incorporating Prometheus into their MLOps workflows, data scientists and engineers can effectively monitor their machine learning models and infrastructure components, ensuring the reliability and performance of their systems.

Dashboard Tools

Grafana

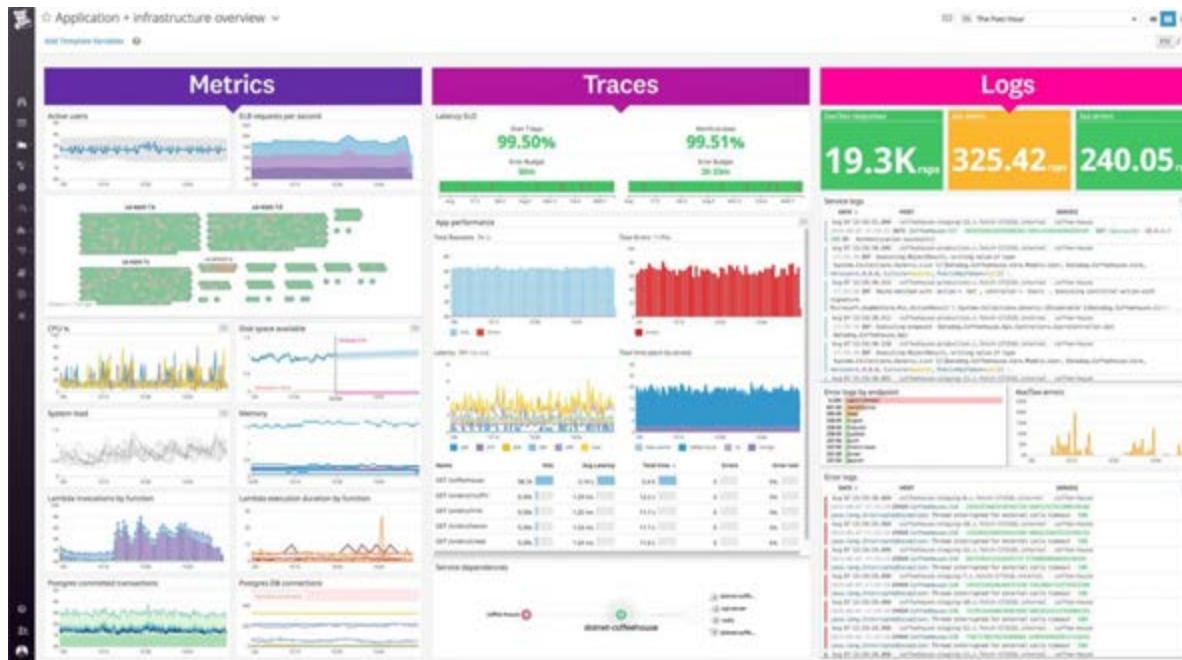
Grafana is an open-source analytics and monitoring platform that allows users to create, explore, and share dashboards and visualizations of their data. Grafana supports a wide range of data sources, including Prometheus, Elasticsearch, and InfluxDB, making it a versatile option for monitoring and performance management in MLOps. Key features of Grafana include:

- Customizable dashboards, allowing users to create and share visualizations of their monitoring data, such as model performance metrics and infrastructure metrics
- Flexible alerting system, enabling users to define custom alert rules and receive notifications through various channels, such as email, Slack, or PagerDuty
- Extensive plugin ecosystem, providing users with additional data sources, panels, and themes to customize their monitoring experience
- Integration with popular authentication and authorization systems, such as OAuth, LDAP, and GitHub, ensuring secure access to monitoring data

By using Grafana in their MLOps workflows, data scientists and engineers can gain valuable insights into their models' performance and infrastructure, identify potential issues, and optimize their systems for better results

Infrastructure monitoring

The first step to ensure that our model is performing well is to ensure that the actual system is up and running as it should. This can include metrics specific to service requests such as latency, throughput, error rates, etc. as well as infrastructure utilization such as CPU/GPU utilization, memory, etc. Fortunately, most MLOps providers and even CaaS layers will provide this insight into our system's health through a dashboard. In the event we don't, we can easily use [Grafana](#), [Datadog](#), etc. to ingest system performance metrics from logs to create a customized dashboard and set alerts. (we need to rely on the underlying platform for this)



ML Model Monitoring

Just monitoring the system's health won't be enough to capture the underlying issues with our model.

So we need various levels of model monitoring.

We need to **monitor training and tuning metrics**, **monitoring the model serving resources** e.g. throughput & latency, and finally, **model monitoring for issues like data drift, model decay** to take a decision to trigger re-training of the model in case the prediction drifts from ground truth beyond a threshold.

Model Training Monitoring

The first level of metrics to monitor and log involves the model training and tuning performance. These would be quantitative evaluation metrics that we used during model training and evaluation e.g. accuracy, precision, f1, etc. There are different metrics to be evaluated in case of regression or classification cases.

Regression Metrics

The most common metrics for evaluating predictions on regression machine learning problems:

- Mean Absolute Error.
- Mean Squared Error.
- R².
- Training parameters

Classification Metrics

Classification problems are perhaps the most common type of machine learning problem and as such there are a lots of metrics that can be used to evaluate predictions for these problems:

- Classification Accuracy.
- Logarithmic Loss.
- Area Under ROC Curve.
- Confusion Matrix.
- Classification Report.
- Training parameters

Model Tuning Monitoring

Once we've done model training and evaluation, it's possible that we want to see if you can further improve our training in any way. We can do this by **tuning hyper-parameters**. There were a few parameters we implicitly assumed when we did our training, and now is a good time to go back and test those assumptions and try other values. the following parameters can be monitored and logged:

- Fine-tuning parameters
- Metric related to model accuracy

In machine learning, the terms **model parameters** and **hyperparameters** are often used interchangeably. However, there is a subtle difference between the two.

• **Model parameters** are the values that are learned by the model during training. They are typically represented by a vector of numbers. The number of model parameters depends on the complexity of the model. For example, a simple linear regression model may have only a few model parameters, while a complex deep learning model may have millions or even billions of model parameters.

• **Hyperparameters** are the values that are set before training the model. They control the learning process and the behavior of the model. For example, a hyperparameter might be the number of iterations to train the model, the learning rate, or the batch size. Hyperparameters can have a significant impact on the performance of the model.

In general, model parameters are learned from data, while hyperparameters are set by the user. However, there are some cases where model parameters can be set by the user, such as when using a pre-trained model.

Model Serving resources monitoring

An important part of monitoring system resources is monitoring the performance of model serving infrastructure. Once the model is trained and served, we are going to use the model serving for quiet sometime. So its important to ensure that the performance of model serving is as per expectation..

[← Model server details](#)



Model Serving Monitoring

There are 2 questions which we need to answer:

1. Does new incoming data reflect the same patterns as the data on which model was originally trained on ?
2. Is the model performing as well in development as it did in design phase ? If not, why ?

Machine Learning Model Drift

How well a model performs is the reflection of the data used to train it. If there is a significant change in distribution or composition of values of input variables or the target variables, it could lead to data drift which could cause the model to degrade. Model degradation can lead to inaccurate predictions and insights, therefore we want to carefully monitor model degradation. *To track model degradation there are 2 approaches to consider:*

1. ***Based on Ground Truth*** - Labelled data is compared to prediction
2. ***Based on Data Drift*** - Instead of waiting for the labelled data, the training data and recent data are compared statistically.

Model Serving Monitoring

Model Monitoring based on Ground Truth vs Data Drift

The ground truth is the correct answer that the model was asked to solve -- when we know the ground truth for all the predictions a model has made, we can judge with certainty how well the model is performing.

Ground truth monitoring requires waiting for the label event and then computing the performance of the model based on these ground truth observations. When the ground truth is available quickly it can be the best solution to monitor model degradation, however obtaining ground truth can be slow and costly.

If our use cases require rapid feedback, or if the ground truth is not available or hard to compute, input drift evaluation may be the way to go. The basis for input drift, is that the model is only going to predict accurately , if the data it was trained on is an accurate reflection of the real world, so if a comparison of the recent requests to a deployed model against the training data , shows distinct differences, then there is a strong likelihood that the model performance may be compromised.

Unlike for ground truth evaluation, the data required for input drift evaluation already exists, so there is no need to wait for any other information.

Note: If we wait to catch the model decay based on the ground-truth performance, it may have already caused significant damage to downstream business pipelines that are dependent on it. We need to employ more fine-grained monitoring to identify the sources of model drift prior to actual performance degradation.

Model Serving Monitoring

Types of Drift

We need to first understand the different types of issues that can cause our model's performance to decay (model drift). The best way to do this is to look at all the moving pieces of what we're trying to model and how each one can experience drift.

Drift Type	Description
Feature/Input Drift → (X)	Input feature(s) distributions deviate. Also known as Covariate Shift .
Label/Target Drift → (y)	Label/Target distribution deviates. Also known as Prior Probability Shift .
Prediction Drift → P(y)	Model prediction distribution deviates. Also known as Model Drift
Concept Drift → P(y X)	External factors cause labels to evolve. Also known as Task Drift

Model Serving Monitoring

Drift types and actions to take:

Feature/Data Drift	<ul style="list-style-type: none">• Investigate feature generation process• Retrain using new data
Label/Target Drift	<ul style="list-style-type: none">• Investigate label generation process• Retrain using new data
Prediction Drift	<ul style="list-style-type: none">• Investigate Model training process• Assess business impact of changes in prediction
Concept Drift	<ul style="list-style-type: none">• Investigate additional feature engineering• Consider alternative approach/solution• Retrain/tune using new data

Model re-training and deployment

- It's usually a good idea to establish policies around when we're going to retrain our model.
- There's no right or wrong answer here, so it will depend on what works in our particular situation.
- we could simply choose to retrain our model whenever it seems to be necessary.
- That includes situations where we detect a drift, but it also includes situations where we always retrain the models according to a schedule, or on demand.
- ***In practice, this is what many companies do, because it's simple to understand and in many domains, it works fairly well.***
- It can, however, incur higher training and data gathering costs unnecessarily.
- If we can automate the process of detecting the conditions which require model retraining, that will be ideal.
- That includes being able to detect model performance degradation or when we detect significant data drift, and triggering retraining.
- In both cases, in order to automate retraining, we should have data gathered and labelled automatically and only retrain when sufficient data is available.
- We need to have continuous training, integration and deployment setup to make the process fully automated.

Model Re-training Policy

Policy	Intuition
On Demand	Re-train the model on need basis (e.g. when ACTS detects that the deployed model is old, it triggers re-training and deployment)
On Schedule	Re-train at a fixed internal on a daily, weekly or monthly basis.
On Data Drift	We find significant changes in the input data and trigger model retraining.
On Model Performance Degradation	Prediction Drift based on ground truth, trigger model retraining.
On New data availability	When new data is available on ad-hoc basis and there is no trend. Re-train model when new labelled data is collected and available in source database.

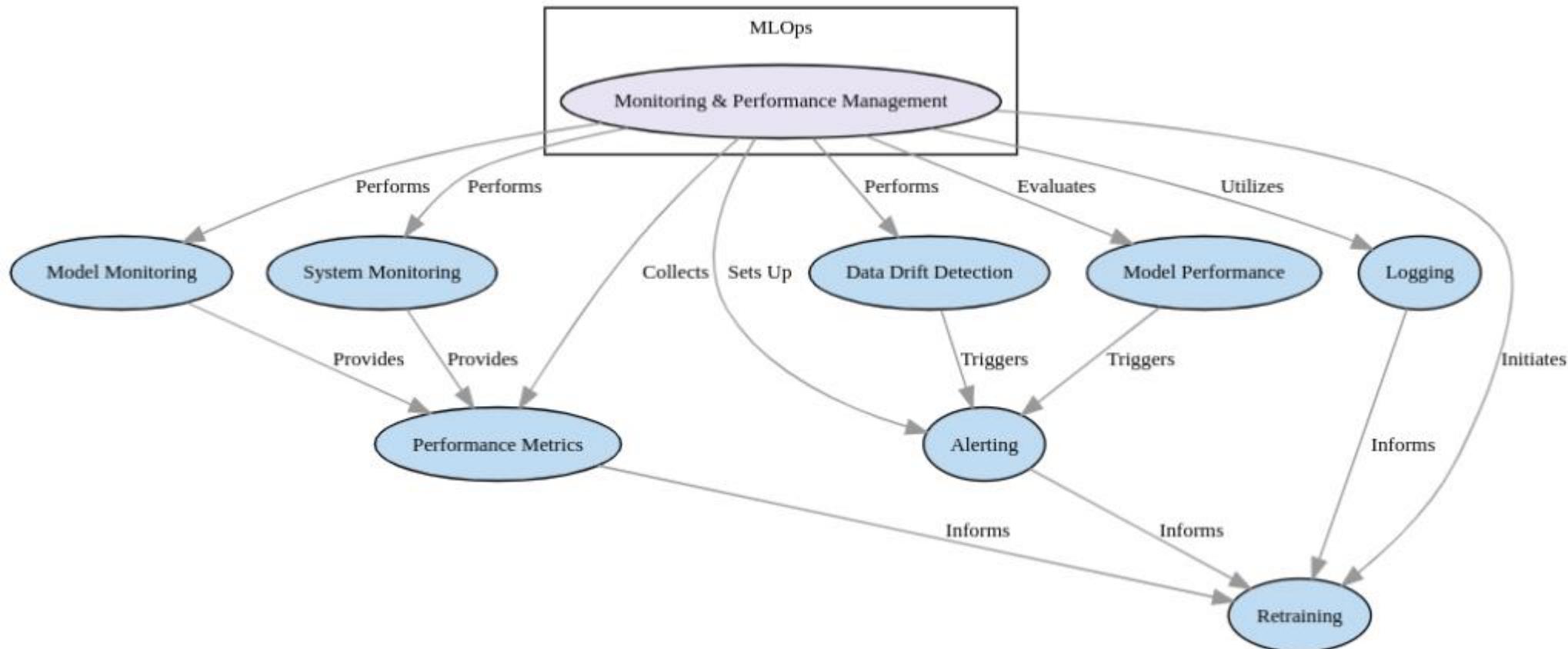
Stateless Vs Stateful model re-training

Policy	Intuition
Stateless	<ul style="list-style-type: none">New model is retrained from scratch each time.
Stateful	<ul style="list-style-type: none">The existing model continues to be trained on new dataStateful re-training is also known as fine-tuning or incremental learning

Four Stages of Continuous Learning

Stage	Description
Stage 1: On-Demand, Stateless retraining	This is the first stage of continuous learning. In this stage we update the models on demand. In this stage the models are updated on need basis or as deemed suitable.
Stage 2: Automated, Stateless retraining	In this stage we write automated and recurring pipelines which rely mostly on schedule to trigger re-training. Most companies with somewhat mature ML infrastructure are in this stage.
Stage 3: Automated, Stateful retraining	In this stage we update existing model which continues to be trained on new data. Stateful re-training is also known as fine-tuning or incremental learning.
Stage 4: Continual learning	Till Stage 3 our models are still updated based on a fixed schedule. This might not be optimal always as quite a few models might not show any performance degradation and hence the stage 3 might lead to higher compute costs. On the flip side some of our models could decay/degrade much faster and require a much faster and automated retraining. In stage 4 our re-training trigger can be combination of one or more factors e.g. data drift, model performance degradation (beyond threshold) or arrival of new data.

Summary



Python Packages available to detect drifts

Some of the open-source packages available to detect drift are:

Evidently AI: Open-Source Tool To Analyze Data Drift. Evidently is an open-source Python library for data scientists and ML engineers. It helps evaluate, test, and monitor the performance of ML models from validation to production.

drift_detection: This package contains some developmental tools to detect and compare statistical differences between 2 structurally similar pandas dataframes. The intended purpose is to detect data drift — where the statistical properties of an input variable change over time. It provides a class DataDriftDetector which takes in 2 pandas dataframes and provides a few useful methods to compare and analyze the differences between the 2 datasets.

skmultiflow: One of the key features of skmultiflow is its ability to handle concept drift, which is the change in the underlying distribution of the data over time. It includes algorithms for detecting concept drift and adapting the machine learning model in real-time to account for the changes in the data. This makes it an ideal tool for building machine learning models that can adapt and continue to perform well as the data changes over time.

Deepchecks: Deepchecks Open Source is a python library for data scientists and ML engineers. The package includes extensive test suites for machine learning models and data, built in a way that's flexible, extendable, and editable.

NannyML: NannyML is an open-source Python library that helps you monitor and improve the performance of your machine learning models

References

- [An overview of unsupervised drift detection methods](#)
- [Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift](#)
- [Monitoring and explainability of models in production](#)
- [Detecting and Correcting for Label Shift with Black Box Predictors](#)
- [Outlier and anomaly pattern detection on data streams](#)



Thank You !

Demo