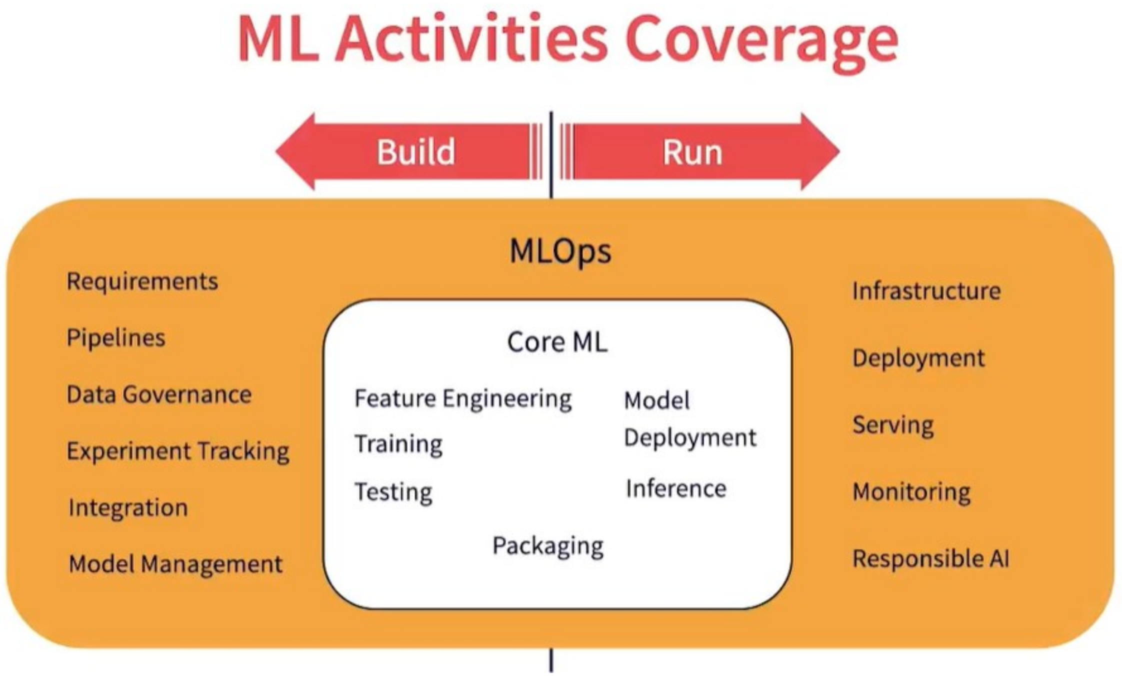
MLOps Introduction

MLOps is critical for anyone building and managing ML applications.

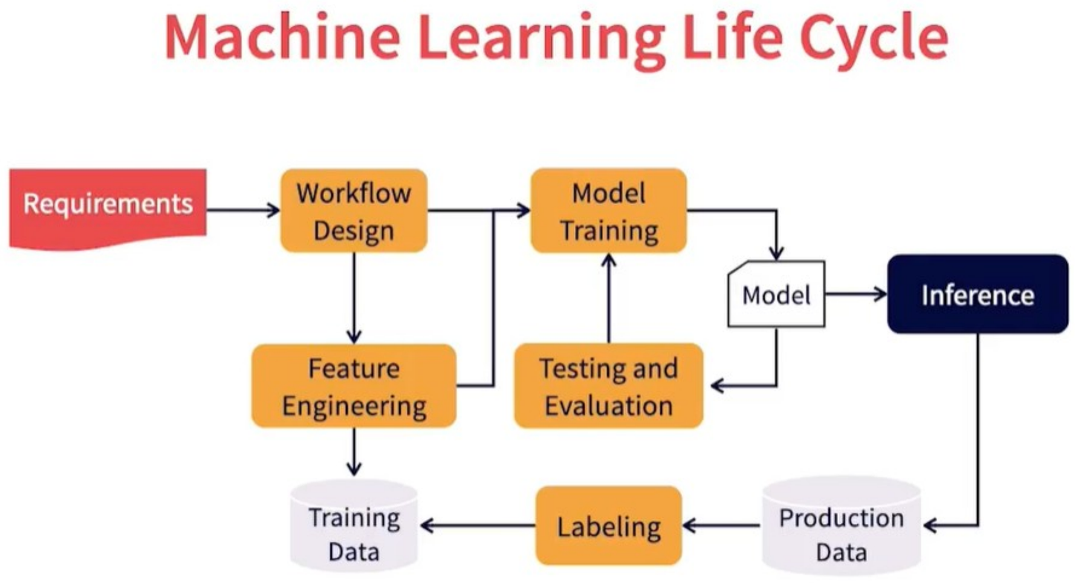


In Machine learning, we can divide the activities into Build and Run activities. Build activities focus on creating and testing the model. Run activities focus on deploying, executing and monitoring the model. Core ML activities include Feature engineering, Training, Testing and Packaging on the build side. Model deployment and Inference are the core activities on the run side. Surrounding these core activities is MLOps, which again can be split into Build and Run. For this part, we will only focus on the Build-side activities of MLOps. There are specific tools for MLOps on AWS, Azure and GCP

Machine Learning Life Cycle:
List of activities when building ML solutions
- Journey of an ML application
Concept => Model => Integration => Deployment => Improvements
It starts from the concept of the problem we are trying to solve, we build a model with training data, the model is deployed and used in production, then periodic improvements happen to the model over time. ML lifecycle is a cyclic process, the process continues for multiple iterations depending upon the use case. Continuous refinements happen during the life of a model like new training data, new user requirements and model decay can also cause the model to be retrained and improved.

Machine Learning Life Cycle
It starts with requirements of the model charted by a product owner. The requirements are then used to design the workflow for executing the project. Training data is acquired and made available for machine learning. Feature engineering is performed on the training data to cleanse, transform and extract useful features. This is then used to train the model. Model goes through testing and evaluation. Model is continuously refined until desired performance goals are achieved. Model is then deployed for Inference and it is used to predict the business workflows. During Inference, more data is collected from the production deployments this is then labeled to create additional training data. New training data again triggers Feature engineering and Model training operations. The process continues in a

cyclic fashion to keep the model optimized to accommodate business environment changes. We are not going to the process of building a model, but the ecosystem around this lifecycle.

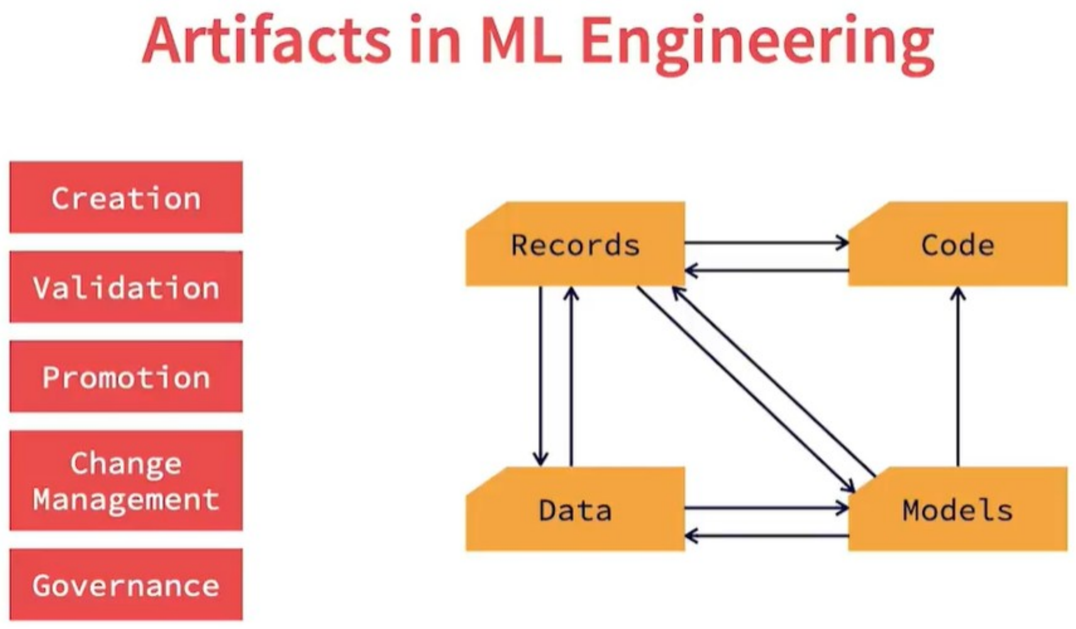


Unique challenges in building ML applications

Artifacts: An artifact is any entity that are created and/or consumed during a process.
Software engineering deals primarily with two types of artifacts: software code and records. Records are any documentation that are created or consumed during the process like requirements, design, test plans, test results, and acceptance results. Software lifecycle process deals with how the lifecycle of these artifacts are managed. During the lifecycle, artifacts are consumed and processed, and new artifacts are created.

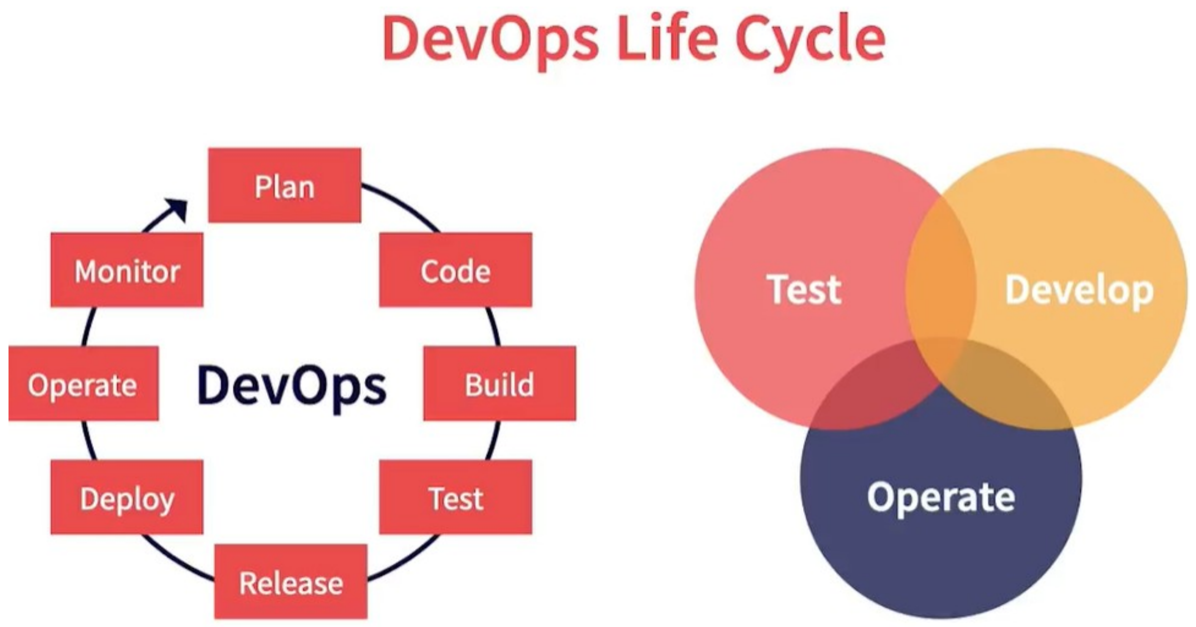
Artifacts: Records like Requirements, and design that are used to create code. In turn, Code is used to create other records like test results. Software engineering process deals with Creation, Validation, Promotion, Change management and Governance of these artifacts. Various processes and tools are used in managing the lifecycle. Artifacts feed off each other. One artifact is the input to create another.

Artifacts for ML Engineering:
All ML products still require code to run in executables. ML has the same two artifacts: code and records. In addition, it requires two more artifacts: Data and Models. Data is training data that is used to build the models. Records like Requirements are used to prepare training data, this in turn creates records about processing status and errors. Requirements and training data are used to build models. The training process in turn creates records around training performance and results. The model is then integrated into code for creating executable binaries. Execution of the models in turn creates new training data. This metrics' interdependence creates challenges in coordinating the lifecycle of these artifacts.



<u>What are the Unique challenges in ML development?</u>
Each artifact follows it own lifecycle but needs coordination
Managing collaboration between artifacts increases multifold
Needs for additional tools and workflows to manage and govern them
Each artifact needs a different sets of skills and teams to create and maintain.
There is a need for an integrated workflow to efficiently manage ML application development

<u>What's DevOps?</u>
DevOps is the most popular software creation and delivery methodology that drives continuous development in an Agile fashion. It combines development, test, and operations of software into one integrated workflow. Rather than having them as separate entities, integration helps with faster iterations of software. It unites people, processes and technology into a seamless workflow. It uses agile principles, best practices and tools for optimal software development. It helps to drive continuous changes to software due to new requirements or issues and helps move these changes quickly into production. DevOps focuses on automation to improve efficiency.
DevOps cycle keeps iterating through the Sprints. Requirements are converted to code, built and tests. Release happens in an automated fashion, deployed in production. The software is operated and monitored. The performance of the software is fed back into the cycle to plan for more requirements. The cycle then iterates over the life of the application. Test, Develop and Operate activities overlap in DevOps while in traditional software development environments, they are done by separate independent teams. In DevOps, the same team manages all three of them. DevOps is the foundation of MLOps.

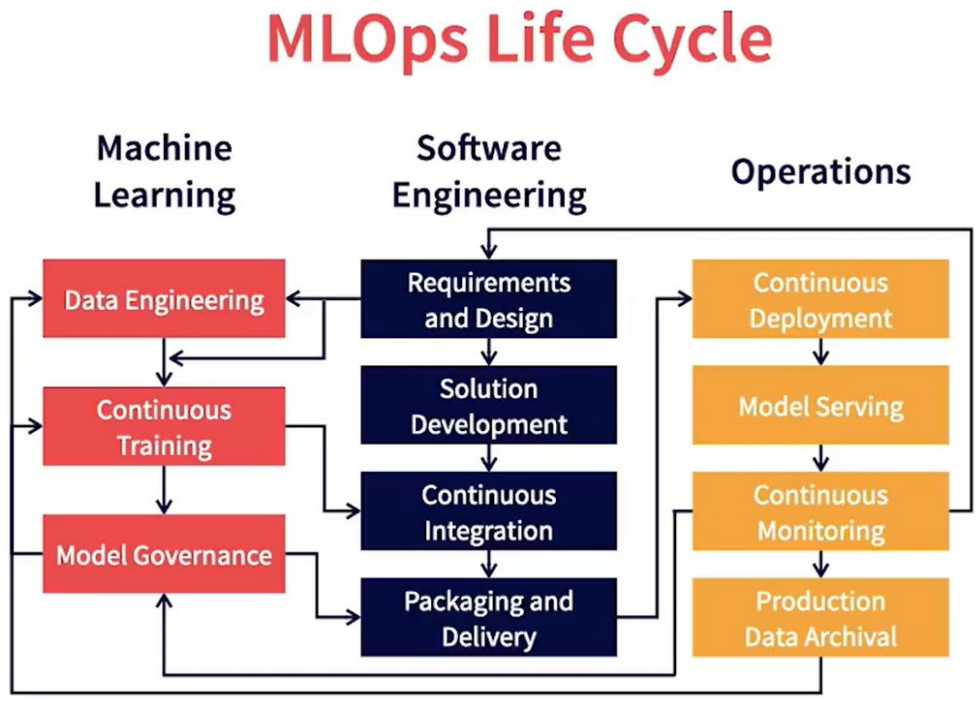


<u>What’s MLOps?</u>
MLOps is a set of best practices that helps manage the creation and use of ML artifacts through efficient workflows, collaboration, and tracking. MLOps is not a specific product or technique. It is a set of processes and best practices to build and run ML supported by automation and tools.

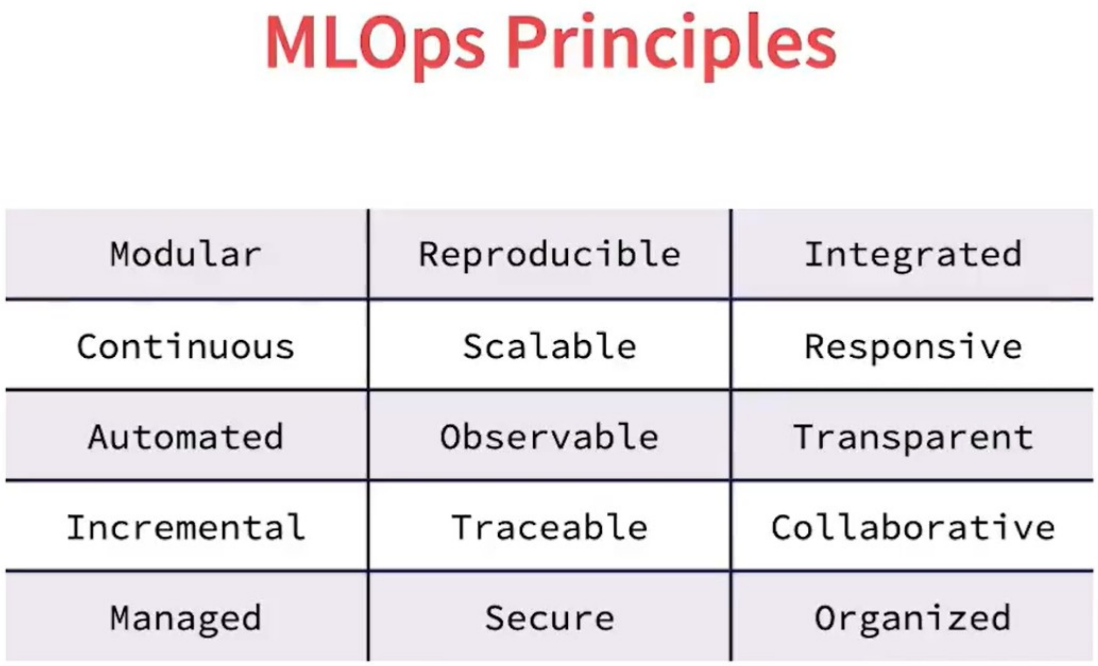
<u>What are the elements of MLOps?</u>
MLOps extends the DevOps methodology to building and serving machine learning solutions.
It integrates the activities of Data Engineering and model development into software engineering and deployment lifecycle
In addition to the Software engineering artifacts of code and records, it manages the machine learning artifacts like data and models.
It enables continuous model development and integration, thus following an Agile process to reduce time to market.
MLOps deals with model deployment and serving.
It also includes monitoring, performance analytics and generating feedback for further improvements.
It helps manage the ML processes effectively through automation and tools to improve efficiency.

<u>MLOPs Lifecycle</u>
Three groups: Machine learning, Software engineering and Operations
Software engineering and Operations are borrowed from DevOps. Additionally we have a Machine learning group too
Lets look into various activities and how they integrate with each other
Requirements and Design: Requirements for ML project and corresponding design. The design includes both non-ML parts like APIs, Services, Databases, User interfaces etc and ML pipelines like data engineering pipelines. This is then used to develop the non-ML parts of the overall solutions. Requirements also would feed into data engineering for converting raw data into usable features for ML. There is a continuous training cycle where a model is built and refined until it meets the stated requirements. Models that are created a managed under a model governance framework. As models are

built they are also integrated continuously with the non-ML code. Continuous here would be specific small interval, like each Sprint or each week. They are also integrated continuously with the non-ML code. After Continuous Integration, model and non-ML code are packaged together and delivered. After this the Operations process starts, Continuous Deployment takes care of deploying the approved packages to production. The model is then served to users. The performance of the model is monitored to ensure that it says within the expected thresholds. Model performance information as well as model drift and bias information is fed back into the model governance for tracking and evaluation. Input is also provided into requirements to see if changes or improvements are needed on the ML or non-ML functions. Finally feature and labeled data from production databases is captured and fed into data engineering pipeline to create new datasets. If the model governance determines that the model needs to be retrained then it kicks-off another training cycle with new dataset.



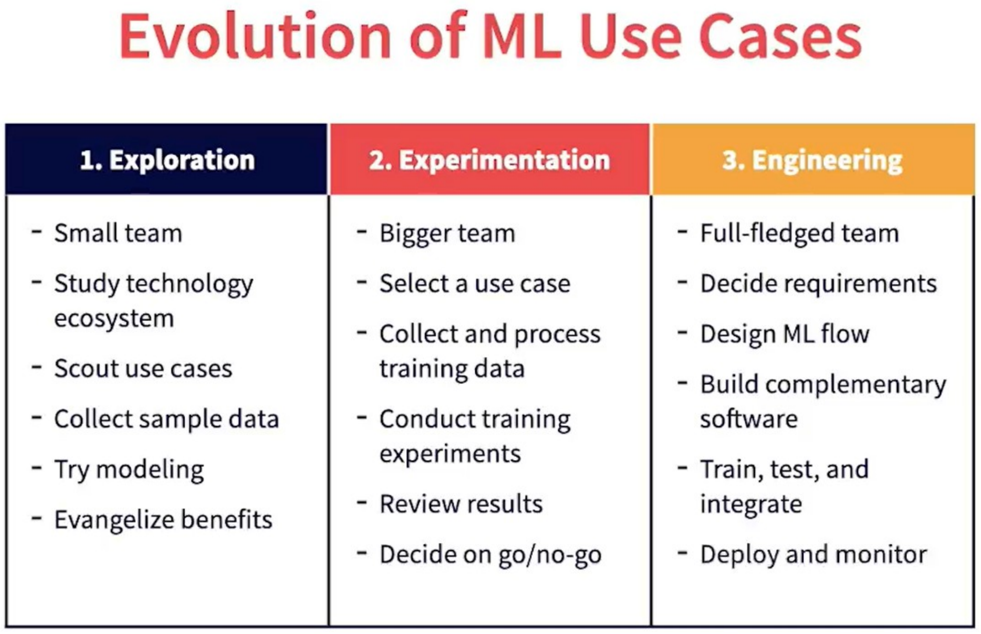
<u>Principles of MLOps:</u>
Overall objective of MLOps is to create an optimal end-to-end workflow that integrates different teams, modules, tools and artifacts to continuously improve and deliver machine learning solutions. The principles of MLOps focuses on helping achieve this overall objective. These principles should be kept in mind while designing MLOps in an organization.



MLOps Principles

Modular	Reproducible	Integrated
Continuous	Scalable	Responsive
Automated	Observable	Transparent
Incremental	Traceable	Collaborative
Managed	Secure	Organized

<u>When should an organization start investing in MLOps?</u>
How ML use cases evolve within an organization?
Three stages: Exploration, Experimentation and Engineering.



Evolution of ML Use Cases

1. Exploration	2. Experimentation	3. Engineering
- Small team - Study technology ecosystem - Scout use cases - Collect sample data - Try modeling - Evangelize benefits	- Bigger team - Select a use case - Collect and process training data - Conduct training experiments - Review results - Decide on go/no-go	- Full-fledged team - Decide requirements - Design ML flow - Build complementary software - Train, test, and integrate - Deploy and monitor

1. Requirements and Design:

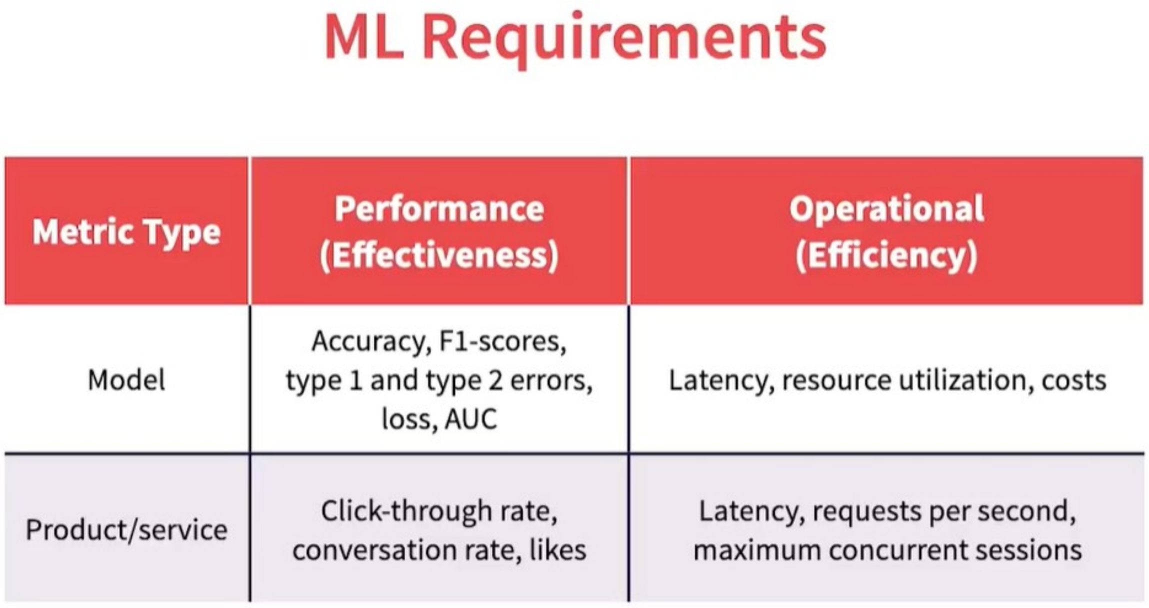
Selecting ML projects:
How we should select ML project for execution. The eventual goal of any ML project is to improve business outcomes. ML project should either help in increasing sales or reducing costs. We should build the ML project only if business benefits outweigh cost of building the model.
Why an organization should build an ML project as opposed to buying an equivalent one from the market?

ML Projects Selection Criteria:
Whether the model could bring core business value
Availability of training data including labels
Technology domain should exist in the specific domain for machine learning, this included algorithms, libraries, frameworks, and pre-trained models as needed. For example, if you use-case is in Computer vision then related-based technologies should be available and affordable.
Budget available to create a team
Time to market
Risk of failure (if model cannot meet desired performance requirements)

Creating requirements for ML projects:
Well-defined and verifiable requirements help machine learning projects progress with focus toward the end goal.

Solution requirements:
It includes both ML and non-ML components and should help solve the business problem
- Non-ML requirements: 1. User interface and API, 2. Solution functions like data-collected, transformations, reporting & analytics, 3. Deployment (how the solution is served to the customers), 4. Scale requirements set the maximum capacity of the system, 5. Security requirement state (how data and system are protected from unauthorized access and damage), 6. Serviceability requirement (how solution will build observability and handle issues)

Non-ML requirements are applicable to any Software projects
- ML-specific requirements: 1. Function (specific problem the model is supposed to solve like recommend next actions), 2. Performance goals (level of model performance desired), 3. Operational goals (specify requirements around accessing and using the model), 4. Cost requirements should layout cost limits.

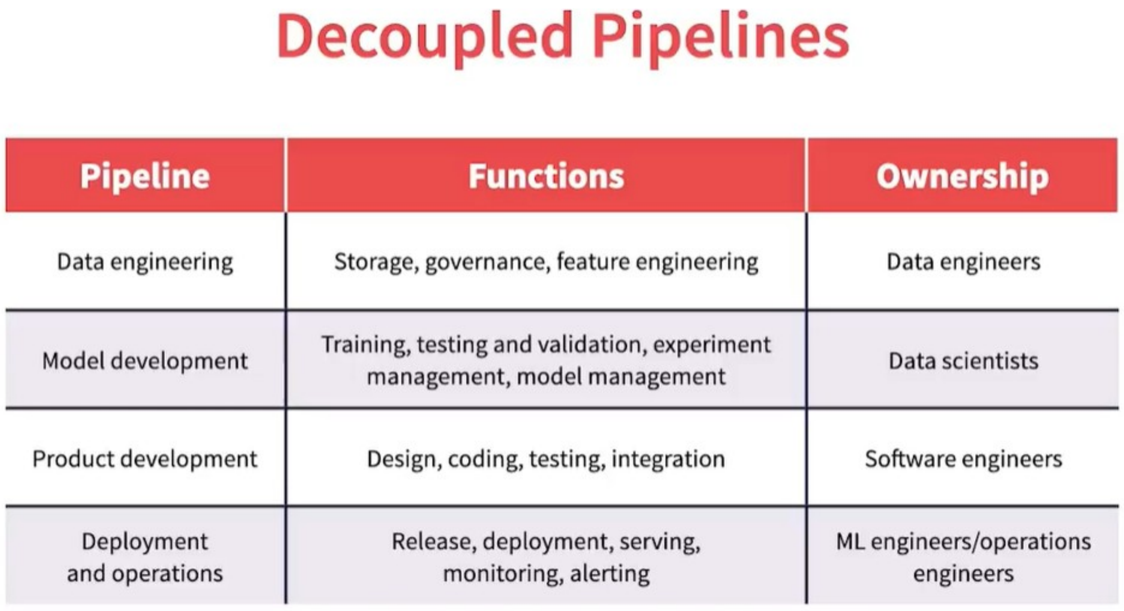


ML Requirements

Metric Type	Performance (Effectiveness)	Operational (Efficiency)
Model	Accuracy, F1-scores, type 1 and type 2 errors, loss, AUC	Latency, resource utilization, costs
Product/service	Click-through rate, conversation rate, likes	Latency, requests per second, maximum concurrent sessions

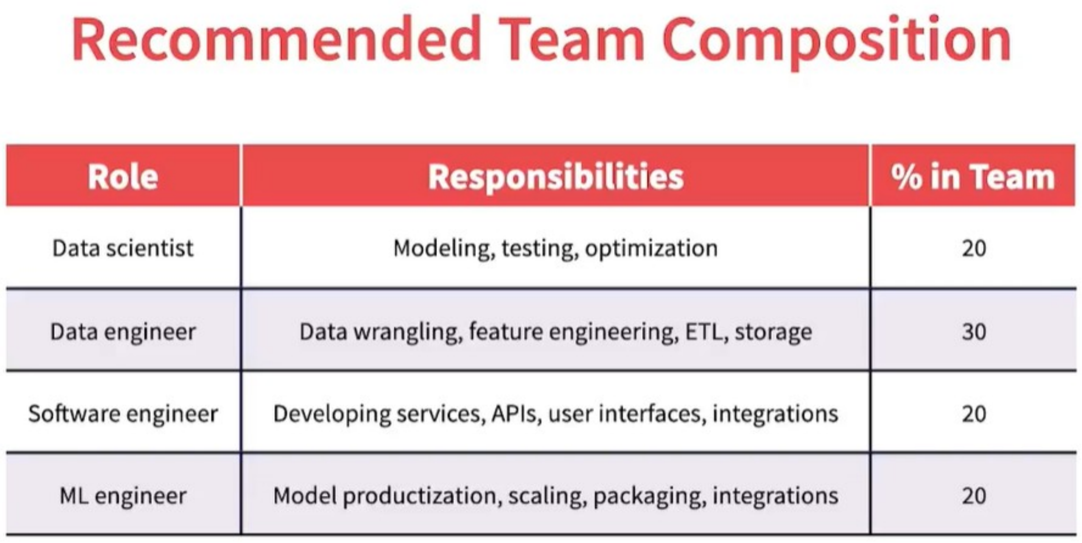
Designing the ML workflow:
ML workflow is a sequence of processes and iterations that helps to develop, manage, and improve machine learning products and services.

Create decoupled sub-pipelines. Having tight coupling leads to a lot of blocked work



Decoupled Pipelines

Pipeline	Functions	Ownership
Data engineering	Storage, governance, feature engineering	Data engineers
Model development	Training, testing and validation, experiment management, model management	Data scientists
Product development	Design, coding, testing, integration	Software engineers
Deployment and operations	Release, deployment, serving, monitoring, alerting	ML engineers/operations engineers



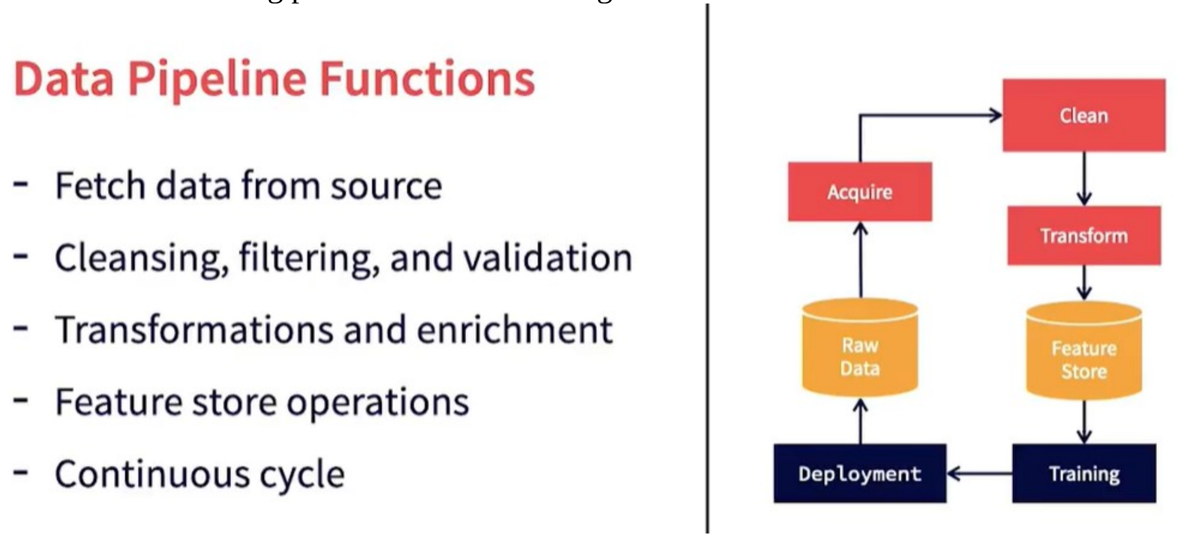
Recommended Team Composition

Role	Responsibilities	% in Team
Data scientist	Modeling, testing, optimization	20
Data engineer	Data wrangling, feature engineering, ETL, storage	30
Software engineer	Developing services, APIs, user interfaces, integrations	20
ML engineer	Model productization, scaling, packaging, integrations	20

MLOps Tools and Technologies Landscape:
Open source (MLFlow), Commercial (Weights & Biases), Some are built on cloud like AWS, Azure

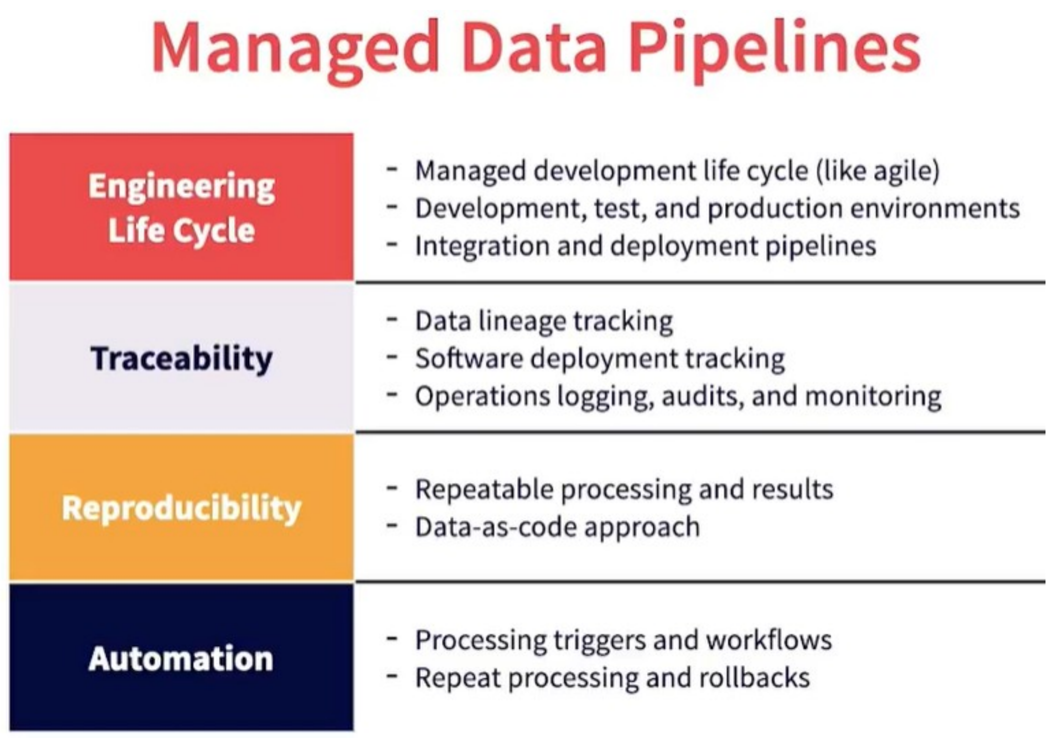
<u>2. Data Processing and Management:</u>

Managed Data Pipelines
A data pipeline is an integral part of an ML workflow. A robust managed data pipeline helps in creating repeatable machine learning processes while reducing human costs.



Data Pipeline Functions

- Fetch data from source
- Cleansing, filtering, and validation
- Transformations and enrichment
- Feature store operations
- Continuous cycle



Managed Data Pipelines

Engineering Life Cycle	- Managed development life cycle (like agile) - Development, test, and production environments - Integration and deployment pipelines
Traceability	- Data lineage tracking - Software deployment tracking - Operations logging, audits, and monitoring
Reproducibility	- Repeatable processing and results - Data-as-code approach
Automation	- Processing triggers and workflows - Repeat processing and rollbacks

Automated data validation:
It should be a key feature of any data pipeline.
Data scientists build models based on the initial set of data
New data is continuously acquired for automated model retraining
All need to be validated

What data should be validated in a data pipeline?
Basic feature validation like missing data, erroneous data, data formats (string, date etc), data formats and ranges
Data distribution and validation like metrics: Mean, SD, quartiles, class distribution for categorical data. This is to ensure that the new set of data belongs to the same distribution population as the original training dataset.
Out of distribution validation: this is to check if outliers occur in the data. We look for values beyond quartiles
Correlation validation: check for correlation between feature and target values, between features

One of the key features needed in the data pipeline for ML is the feature store.
A feature store is a centralized store for features. When we say features, that are of interest and directly consumed by ML models. Data is already pre-processed and converted to a form that is ready for machine learning. An organization may have a single feature store that may be shared by multiple teams and ML models. Feature store is regularly updated with new training data and new features. It usually has a Registry of data so that users can look up the registry to understand the data that is available in the store.

Data versioning:
It is a key aspect of tracking lineage for data. Managing training data with a data versioning system provides the ability to change data continuously while ensuring consistent training results and collaboration.

Version for data changes when its contents change.

Benefits of Data versioning:
Traceability in MLOps. We can trace the history of a dataset.
Reproducibility – recreate the same ML results again
Change log capture tool for feature stores
Allows data to be rolledback to the last known good state
Multiple users can reference different versions of the dataset

Data Governance:
Data governance is a key area in MLOps that deals the administration of data stored and used for machine learning. Data Governance is the practice of ensuring integrity, security, and usability of data through an organization of people, policies, and processes. First goal of governance is to ensure consistency of data. Data stored across the pipeline should be complete, accurate and conform to requirements.

Tools and Technologies for Data Engineering that help in ML logs
Big data processing technologies like Apache Hadoop, Spark, Kafka, etc (data processing + control, rollback, operations, logging, and change data capture)
Databases for data storage (RDBMS & NoSQL), MySQL, MongoDB, Apache Cassandra, HDFS (capabilities like resiliency, access control, recovery, schema management and versioning at various levels)
Data versioning tools like DVC, LakeFS, Neptune

<u>3. Continuous Training:</u>

Managed training pipelines
Managed training pipelines are similar to Managed data pipelines.
A robust managed training pipeline helps create repeatable ML training and testing workflows while reducing human costs

Training pipeline functions
Training inputs are fetched from the Feature store for model training
Hyper-parameters are also setup for training
An experiment is planned and executed. Executing the experiment results in the ML model.
The model is then validated during training to ensure desired levels of performance are achieved
An independent test dataset is used from the Feature store and the model is tested with this dataset.
Results of this testing is reviewed to see if desired performance goals are met
This process keeps repeating until a model with desired performance is achieved

Data labeling or Annotations:
It is the process of adding contexual tags/labels to training data that can then be used as targets for ML training. To build models from unstructured data, labeling is required.

How do we label data?
We can use Experts in the domain to label data
Crowdsourcing where large pool of volunteers are organized to look at the data and label them

Third-party professional annotators will do labeling for a fee
Programmatic labeling, a program does labeling instead of a human

How do we track ML Experiments?
Each ML training run is considered an experiment. Experiment tracking helps manage the evolution of an ML model toward stated performance goals.

Experiment tracking helps measure impact of changing model parameters, identify model behavior changes due to new training data, measure project progress toward stated requirements and goals, with the results to decide on whether model needs to be promoted to production. This analysis can be automated along with decision criteria for promotions thus leading to automated experimental analysis

What's AutoML?
AutoML is the automation of all machine learning activities to enable model building and decision-making without human intervention throughout the ML lifecycle. It deals with automating the job of Data scientists and helps scale machine learning.

What activities can be automated in AutoML?
Feature engineering, Model training (auto ML algorithm selection), Deployment (automatically determine if the model is ready for production deployment), Create custom model by use case.

Tools and Technologies for Model training:
Model development: Python, Jupyter notebooks, Git
Experiment management: Kubeflow, Mlflow, Weight & Biases
AutoML: Kubeflow, Databricks

Training Models in GenAI:
How training is bit different in GenAI?
In GenAI, there are two key tasks: Pre-training (create a GenAI foundation model from scratch, something like Wikepedia is used to train foundational models) and Fine-tuning (A foundational model is taken and fine-tuned for a specific domain/task)

4_ModelManagement:

Model versioning:
Similar to Data versioning, we need model versioning to keep track of the evolving models over time due to new training data and hyperparameters tuning.

Team can link data versions and code versions using model versions

Model registry:
A repository for storing and tracking machine learning models, like source code control for software

Benchmarking ML:
Compare models/versions against baselines/competing models to understand how they perform against each other on stated requirements and environments

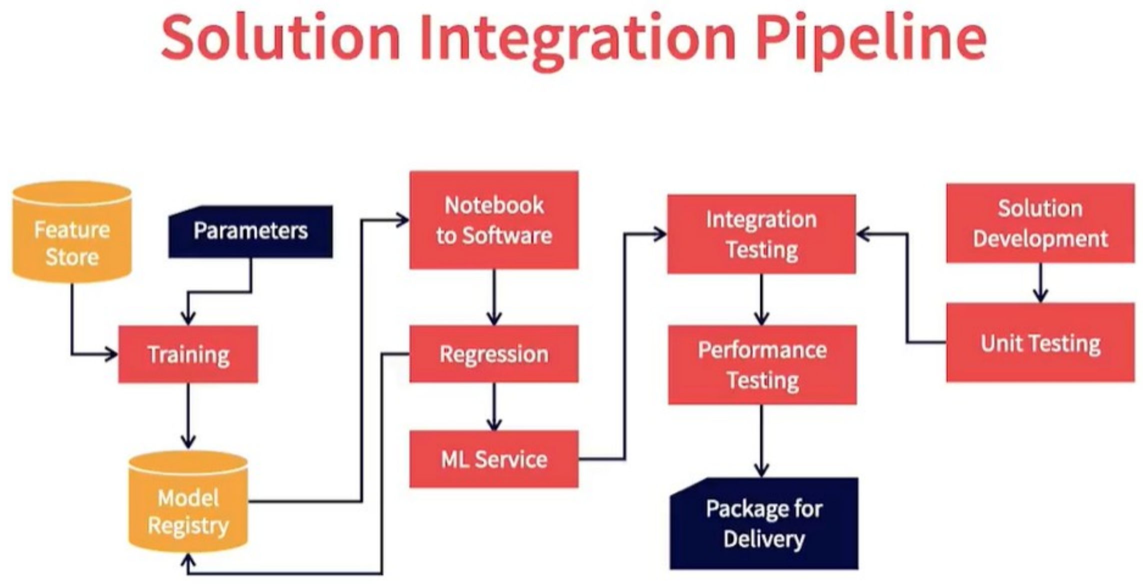
A well-defined model lifecycle with associated policies and processes helps organize, manage, and scale ML in an organization. When we are talking about lifecycle, we are talking about multiple model states and the state transitions that happen during life of a model.

Model has many states like Created state, Passed, Approved, Archived, Deployed, Retired state

5_Continuous_Integration:

Solution Integration pipelines:
Once the model is ready and benchmarked, it needs to be integrated with the non-ML part of the solution. ML models do not work stand-alone, they need to be embedded into other code to deliver end-to-end solutions. This requires integration with other code like APIs, UI, databases, and microservices.

We train the model using the input from Feature store and hyperparameters, we then store the model in the model registry. The model with its pre-processing and post-processing code is in the form of a notebook. It needs to be converted into a software form that's suitable for integration. We call this step Notebook to software. This method produces the model in an executable form. This is then regression tested to make sure the model still continues to perform as its baseline notebook form. This produces an ML service that is ready for integration. It could be a library or function as well instead of a service. The non-ML code is part of the Solution development and evolves independently.



Solution integration patterns:
When do we integrate ML and non-ML parts of the solution?

Say we have Model_v2.0 we train then produce Model_v3.0 => Notebook to software => ML Service v3.0
In parallel, we have Non-ML_v4.0 => Development & Testing => Non-ML v5.0

When do we integrate?
Hold ML constant during a non-ML iterative cycle
Hold Non-ML constant during a ML iterative cycle

How is model integration done when it comes to GenAI use-cases?
GenAI models are huge, so the model is usually used through a cloud service provider.