# The Modern MLOps Framework

MLOps (Machine Learning Operations) is a discipline that applies DevOps principles to machine learning systems. It ensures that ML models are not only developed but also deployed, monitored, and maintained in a scalable, automated, and reproducible manner.

**2. Definition and Scope**

* **MLOps**: A set of principles, practices, and tools that automate and streamline the lifecycle of ML models.
* **Scope**: Covers the entire ML lifecycle, from development to deployment and maintenance. This course focuses on the **Operations** phase post-training deployment, monitoring, and updates.

**3. MLOps vs DevOps**

* **DevOps**: Focuses on software development and deployment using source code.
* **MLOps**: Adds complexity by integrating data and models into the development and deployment pipeline.

**Example**: DevOps deploys a web application built from code. MLOps deploys a trained ML model that relies on data pipelines, model artifacts, and monitoring systems.

**4. Risks of ML Without MLOps**

* Manual workflows lead to inefficiency and errors.
* Lack of monitoring causes model drift and degraded performance.
* Accumulated technical debt slows innovation and increases operational cost.

**Technical Debt Definition**: The implied cost of additional rework caused by choosing an easy solution now instead of a better, longer-term approach.

**Reference**: Google’s paper “Machine Learning: The High-Interest Credit Card of Technical Debt” highlights how unmanaged ML systems accumulate risk and inefficiency.

**5. ML Workflow Stages**

Typical ML workflows include:

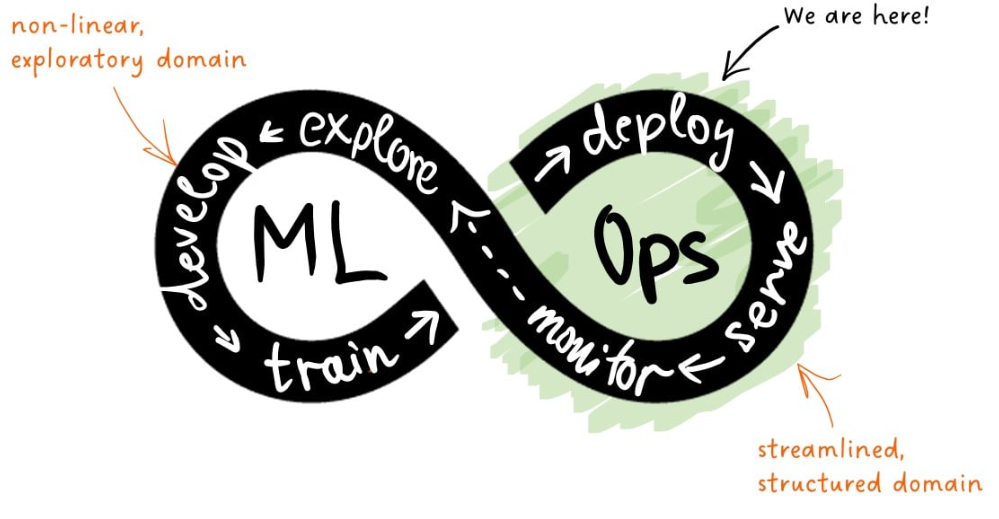
1. Data Collection and Preparation
2. Data Labeling
3. Model Selection
4. Model Training
5. Model Packaging
6. Model Deployment
7. Model Monitoring
8. Model Maintenance

The more these workflows are automated and integrated into the IT infrastructure, the higher the MLOps maturity of the organization.

**6. Benefits of MLOps Implementation**

* Automation of repetitive tasks
* Reproducibility of model results
* Faster deployment cycles
* Explainability and transparency
* Scalable and maintainable ML systems
* Improved customer trust and service quality

**7. Focus on the Operations Phase**

****

* Begins after model training
* Includes deployment as a service, real-time monitoring, and maintenance
* Transitions from exploratory, nonlinear development to structured, linear production workflows

A screenshot of a computer

AI-generated content may be incorrect.**Key Insight**: Once deployed, the model is exposed to users. Any error or drift is immediately visible, requiring rapid response and minimal margin for error.

**Model Life Cycle in MLOps**

**1. Introduction to Life Cycles in Machine Learning**

In machine learning, the term "life cycle" can refer to multiple layers of activity. To avoid confusion, it's important to distinguish between:

* **ML Project Life Cycle**
* **ML Application Life Cycle**
* **ML Model Life Cycle**

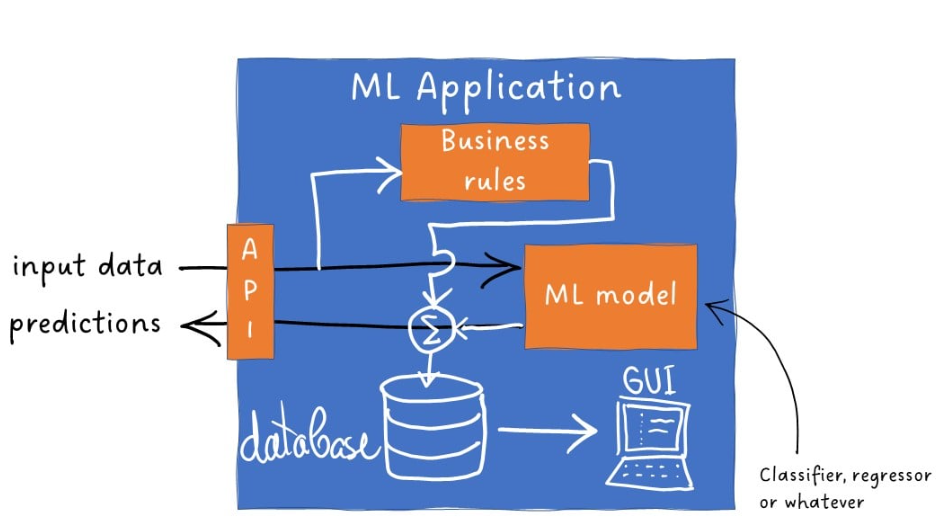
**ML Model Life Cycle**, which begins after model training and continues through deployment, monitoring, and eventual decommissioning.

**2. Definitions and Relationships**

**ML Project Life Cycle**

* Represents the overarching effort to solve a business problem using machine learning.
* Includes problem definition, data acquisition, experimentation, and solution delivery.
* If successful, it results in the creation of ML applications and models.

**ML Application vs ML Model**

* **ML Model**: A trained estimator that performs a specific task (e.g., predicting daily sales).
* **ML Application**: A complete software system that includes the ML model and additional components such as:
  + Business rules (e.g., fallback logic for cold-start users)
  + Databases (for storing features and logs)
  + Graphical User Interfaces (for configuration and troubleshooting)
  + APIs (for secure external communication)

**Example**: An ML model predicts movie ratings. An ML application uses that model, applies business rules, stores user data, and serves recommendations via an API.

**3. Monolithic vs Microservice Architecture**

**Monolithic ML Application**

* The model is tightly coupled with the application.
* Difficult to update or replace components independently.

**Microservice Architecture**

* Model and application are decoupled.
* Each component evolves independently.
* Enables separate life cycles for the application and the model.

**Analogy**: Think of the ML application as a car and the ML model as its tires. The car may last decades, but the tires (models) are replaced frequently.

A diagram of a diagram of a mobile application

AI-generated content may be incorrect.

**4. ML Model Life Cycle Stages**

**1. Deployment**

* A trained model, along with necessary resources, forms a **deployment package**.
* Deployment marks the beginning of the model's operational life.
* The model is exposed to real-world data and users.
* Model object + deployment resources = deployment package

**2. Monitoring**

* Continuous observation of model behavior post-deployment.
* Ensures the model is running and performing as expected.
* Detects issues like model drift, latency, or prediction errors.

**3. Decommissioning**

* Triggered when the model becomes outdated or underperforms.
* Reasons may include:
  + Better model available
  + Improved features
  + Changes in the underlying business process
* The model is retired and replaced.

**4. Archiving**

* Essential for regulated industries.
* Archived models must be reproducible able to be reloaded and executed exactly as before.
* Enables auditability and historical analysis.

**Example**: A financial institution may need to reproduce a credit scoring model from five years ago to justify a loan decision during an audit.

A screenshot of a computer

AI-generated content may be incorrect.

**Core Components of the MLOps Framework**

MLOps extends DevOps principles to machine learning systems, enabling automated, reproducible, and scalable workflows for building, deploying, and maintaining ML models and applications. This framework includes both general software engineering concepts and ML-specific infrastructure.

**2. Foundational Concepts**

**Workflows**

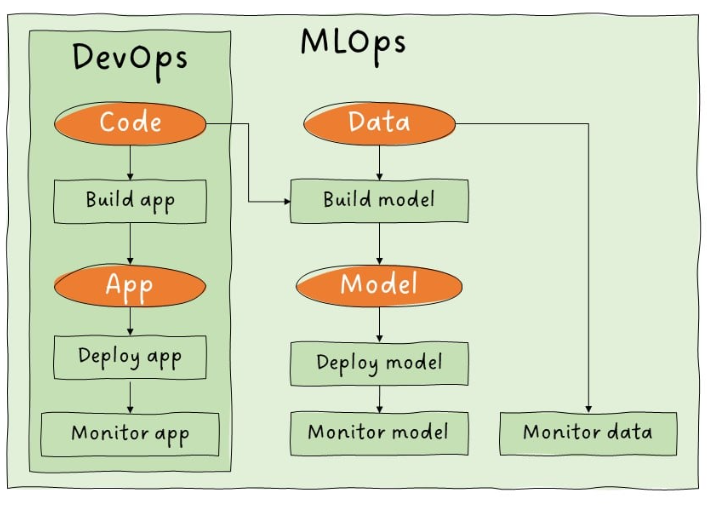
* A **workflow** is a sequence of tasks that transforms inputs into outputs.
* Execution modes:
  + Manual
  + Automatic
  + Semi-automatic

**Pipelines**

* A **pipeline** is a scripted, automated workflow.
* Common across DevOps, DataOps, and MLOps.
* Converts workflows into repeatable, programmable processes.

**Artifacts**

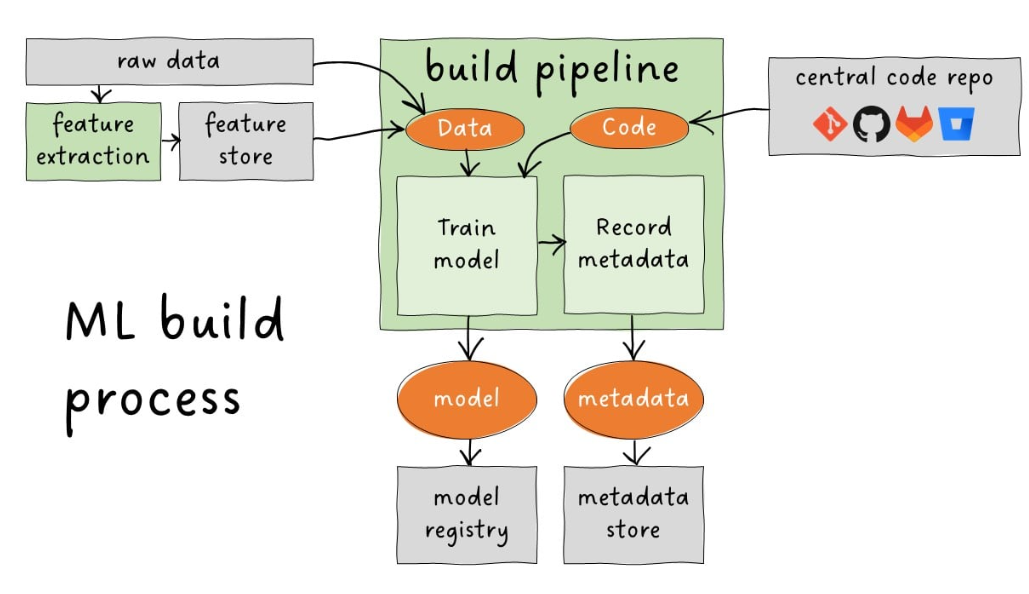
* **Artifacts** are outputs of pipelines.
* Includes compiled code, trained models, configuration files, and metadata.
* Used for deployment, versioning, and reproducibility.

**3. ML-Specific Components**

| **Component** | **Description** |
| --- | --- |
| Feature Store | A database for storing processed variables (features) used in model training. |
| Model Registry | A system for storing, versioning, and managing trained ML models. |
| Metadata Store | Stores auxiliary data about models, including training parameters, datasets, and performance metrics. |

**4. Build Pipelines**

**Model Build Pipeline (Training Pipeline)**

****

* Transforms raw or processed data and model code into a trained model.
* Requires:
  + Model definition
  + Training script
  + Training data
* A diagram of a software development process

  AI-generated content may be incorrect.Outputs:
  + Trained model artifact
  + Model metadata
* Stores results in:
  + Model Registry
  + Metadata Store

**Application Build Pipeline**

* Classical DevOps process for packaging ML applications.
* Steps:
  + Pull code from central repository
  + Apply transformations (e.g., containerization)
  + Store deployable app in a repository

**5. Deployment Pipeline**

* Combines model and application artifacts.
* Deploys them to a target serving platform.
* Ensures the system is ready for production use.

**Example**: The deployment pipeline may use Docker and Kubernetes to launch a REST API that serves predictions from a trained model.

**6. Monitoring**

* Begins post-deployment.
* Tracks:
  + Service uptime
  + Prediction accuracy
  + Latency
  + Model drift
* Ensures the model continues to perform as expected in real-world conditions.

**7. High-Level MLOps Architecture Summary**

| **Stage** | **Description** |
| --- | --- |
| Workflow Design | Define tasks and dependencies |
| Pipeline Scripting | Automate workflows via scripts |
| Build Pipelines | Train models and package applications |
| Artifact Management | Store outputs in registries and metadata stores |
| Deployment Pipeline | Launch model and app on serving infrastructure |
| Monitoring | Continuously evaluate performance and reliability |

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Deployment-Driven Development in MLOps**

Deployment-driven development is a mindset in MLOps that emphasizes planning for deployment from the earliest stages of model development. It ensures that models are not only accurate but also deployable, maintainable, and robust in real-world environments.

**Key Principle**: MLOps does not begin at deployment it begins during development. Ignoring deployment until the end risks producing models that are incompatible with production systems.

**2. Analogy: The Race Car Driver**

Just as a race car driver adjusts speed and steering before reaching a curve, ML engineers must anticipate deployment constraints early in the development process. This proactive approach prevents costly rework and deployment failures.

**3. Deployment Scenario: Inheriting a Colleague’s Model**

Imagine being tasked with deploying and maintaining someone else’s ML model and application. Several critical concerns immediately arise:

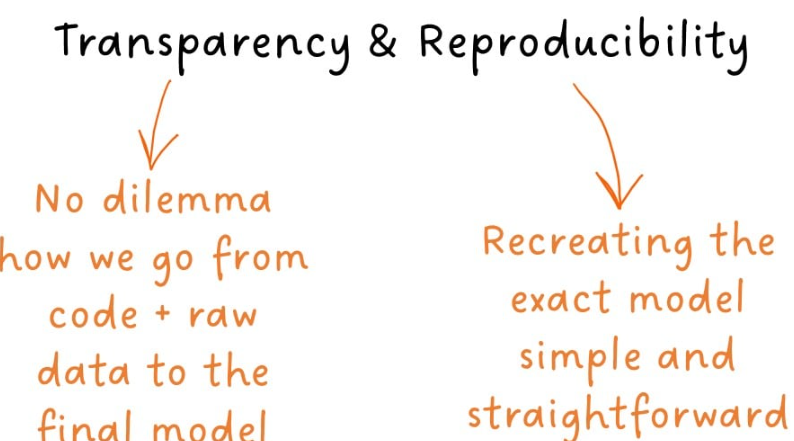
A cartoon of a person and a box

AI-generated content may be incorrect.**4. Key Deployment Concerns and Best Practices**

**1. Infrastructure Compatibility**

* **Concern**: Can the model run on the target platform?
* **Example**: A model trained on a 16-core server may be incompatible with a smartphone deployment.
* **Best Practice**: Understand deployment constraints (memory, CPU, OS) early in development.

**2. Transparency and Reproducibility**

* **Concern**: Is it clear who trained the model, when, and with what data and parameters?
* **Best Practice**:
  + Use versioned datasets and transparent pipelines.
  + Log all training experiments in a metadata store.
  + Ensure reproducibility by documenting every step from raw data to trained model.

**3. Input Data Validation**

* **Concern**: How do we handle invalid user inputs?
* **Example**: A user provides a negative value where only positive values are allowed.
* **Best Practice**:
  + Define data profiles or expectations.
  + Save validation rules in model metadata during the build pipeline.

**4. Performance Monitoring**

* **Concern**: How do we detect if model performance deteriorates over time?
* **Best Practice**:
  + Log inputs and predictions.
  + Monitor metrics like accuracy, latency, and drift.
  + Set up alerts for performance thresholds.

**5. Debugging**

* A diagram of a software program

  AI-generated content may be incorrect.**Concern**: Can we locate and fix bugs efficiently?
* **Best Practice**:
  + Implement detailed logging throughout the application.
  + Use specialized debugging tools.
  + Avoid ad hoc fixes build structured logging from the start.

**6. Code Maintainability and Testing**

* **Concern**: Can we safely modify the code without introducing new bugs?
* **Best Practice**:
  + Develop a comprehensive test suite:
    - Unit tests
    - Integration tests
    - Load tests
    - Stress tests
    - Deployment tests
  + Maintain test coverage as the codebase evolves.

**Data Profiling, Versioning, and Feature Stores in MLOps**

Modern MLOps frameworks rely on robust data management practices to ensure model reliability, reproducibility, and performance in production. Three foundational components are:

* Data profiling
* Data versioning
* Feature stores

These tools and practices help validate inputs, track data lineage, and prevent training-serving mismatches.

**2. Data Profiling**

* Automated analysis of input data to generate high-level summaries called **data profiles** or **expectations**.
* Used for validating and monitoring data in production environments.

**Purposes**

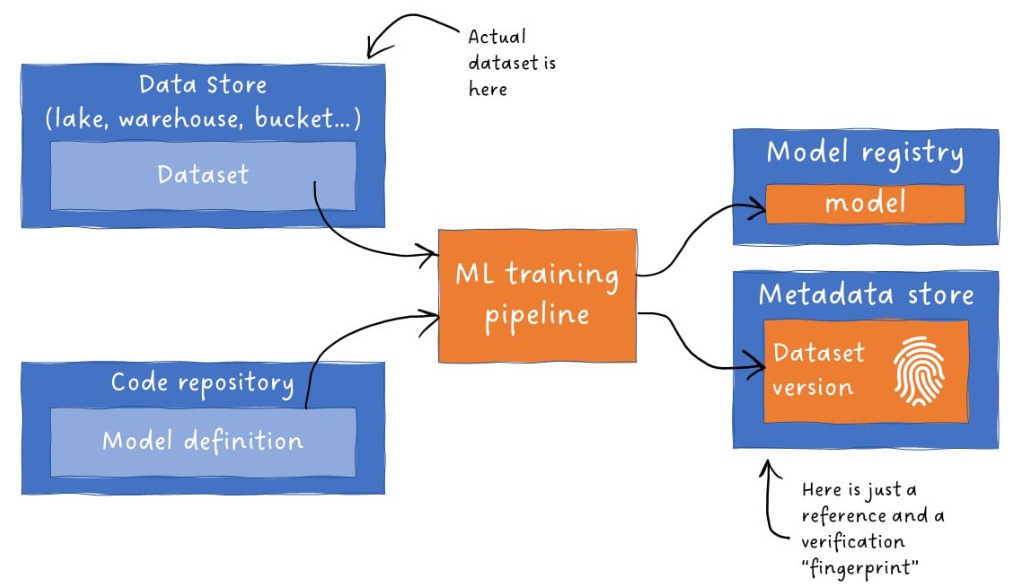
* Provide feedback to users submitting invalid inputs.
* Detect data drift and trigger model retraining.
* Prevent misattribution of model errors when inputs are flawed.

**Risks of Omission**

* Clients may blame models for poor predictions caused by bad inputs.
* No mechanism to detect when retraining is needed due to data drift.

**Best Practices**

* Include a profiling step in the model training pipeline.
* Store data profiles in the **metadata store** alongside model metadata.

**Tool Example**

* **Great Expectations**: A popular Python-based open-source tool for data profiling and validation.

**3. Data Versioning**

* Recording the exact version of the dataset used to train a model.
* Ensures reproducibility without duplicating entire datasets.

**Implementation**

* Store datasets in a centralized location.
* Record a **pointer** and a **dataset fingerprint** in the model metadata.
* Fingerprint ensures no records have changed since training.

**Bonus Practice**

* Record metadata to reconstruct the exact **train-test split** used during model evaluation.

**Tool Example**

* **DVC (Data Version Control)**: A widely used tool for tracking datasets, models, and experiments.

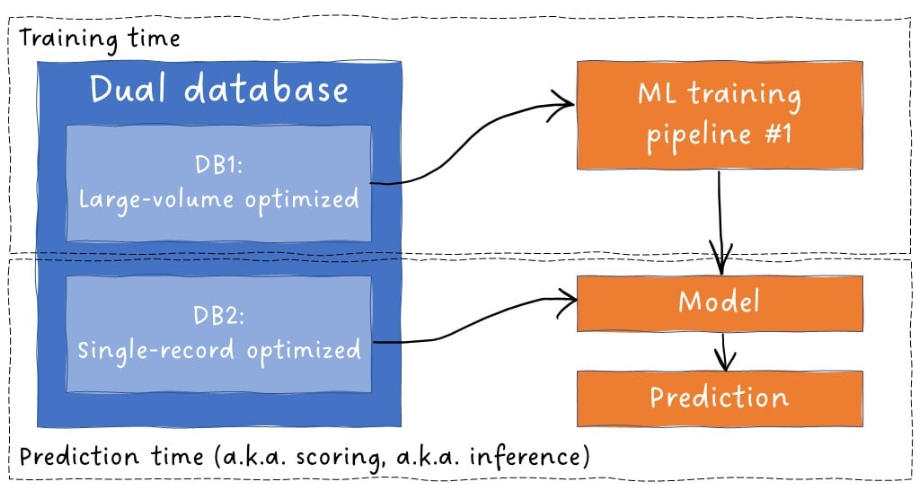
**4. Feature Stores**

Centralized databases that store processed variables (features) for ML training and inference.

**Benefits**

* Reuse features across multiple models and projects.
* Reduce time spent on feature engineering.
* Prevent **training-serving skew**.

**Architecture**

* Often implemented as **dual databases**:
  + One optimized for bulk data retrieval during training.
  + One optimized for fast, row-level access during inference.

**Training-Serving Skew**

* Occurs when models perform well during training but poorly in production due to inconsistent data preprocessing like cleaing data in test but forgot to o so in production.
* Example: Training a spam filter on clean text emails, but deploying it on HTML emails.

**5. insum**

| **Component** | **Purpose** | **Tools/Practices** |
| --- | --- | --- |
| Data Profiling | Validate inputs, detect drift, guide retraining | Great Expectations, metadata store |
| Data Versioning | Ensure reproducibility, track dataset lineage | DVC, dataset fingerprinting |
| Feature Stores | Centralize features, prevent training-serving skew | Dual DB architecture, reusable features |

Robust data profiling, versioning, and feature management are essential for building reliable, maintainable, and auditable ML systems. These practices reduce risk, improve transparency, and ensure consistent performance across environments.

A screenshot of a phone

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Model Build Pipelines in CI/CD for MLOps**

In MLOps, model build pipelines are automated workflows that train machine learning models and prepare them for deployment. These pipelines are distinct from model pipelines (which define the data processing steps within a model) and must meet rigorous standards for deployment, reproducibility, monitoring, and CI/CD integration.

**2. Distinction: Model Pipeline vs Model Build Pipeline**

| **Term** | **Definition** |
| --- | --- |
| Model Pipeline | Sequence of data processing steps (e.g., cleaning, feature extraction, prediction) executed by the model during inference. |
| Model Build Pipeline | Automated workflow that loads training data and model code, trains the model, and outputs deployment-ready artifacts. |

**3. Dual Build Pipelines in ML Systems**

1. **Application Build Pipeline**
   * Standard DevOps pipeline for packaging and deploying the ML application.
   * Includes code compilation, dependency resolution, and containerization.
2. **Model Build Pipeline**
   * Central to MLOps.
   * Trains the model and generates a complete deployment package.

**4. Requirements for an MLOps-Grade Model Build Pipeline**

**1. Deployment Readiness**

* Outputs a **complete model package**, not just the trained model.
* Includes:
  + Model object
  + Dependency specifications (e.g., environment files, Dockerfiles)
  + Configuration files
* Supports **test deployments** to verify compatibility with target infrastructure.

**2. Reproducibility**

* Ability to recreate the model from scratch at any time.
* Key practices:
  + **Code versioning** (e.g., Git commits)
  + **Data versioning** (e.g., DVC pointers and fingerprints)
  + Recording both in the **model metadata**

**3. Monitoring Enablement**

* Ensures the model behaves as expected in production.
* Requires:
  + Logging of inputs and predictions
  + Creation of **data profiles** during pipeline execution
  + Integration with monitoring tools and dashboards

**4. CI/CD Integration**

* Pipeline runs within a **Continuous Integration/Continuous Deployment** framework.
* Benefits:
  + Prevents use of unversioned code or data
  + Enforces consistency and traceability
  + Automates testing, packaging, and deployment
* Requires:
  + Connection to input sources (e.g., code repo, feature store)
  + Storage for generated artifacts (e.g., model registry, metadata store)

**5. insum**

| **Component** | **Purpose** |
| --- | --- |
| Model Build Pipeline | Trains and packages models for deployment |
| Deployment Artifacts | Ensure compatibility and reproducibility |
| Monitoring Integration | Enables performance tracking and drift detection |
| CI/CD Enablement | Automates and secures the build process |

A robust model build pipeline is the backbone of scalable, trustworthy MLOps. It ensures that models are not only trained effectively but also deployed reliably, monitored continuously, and reproducible under audit conditions.