The Hoang

Model Evaluation & Deployment in MLOps

Understand and Implement Production-Grade Machine Learning Operations

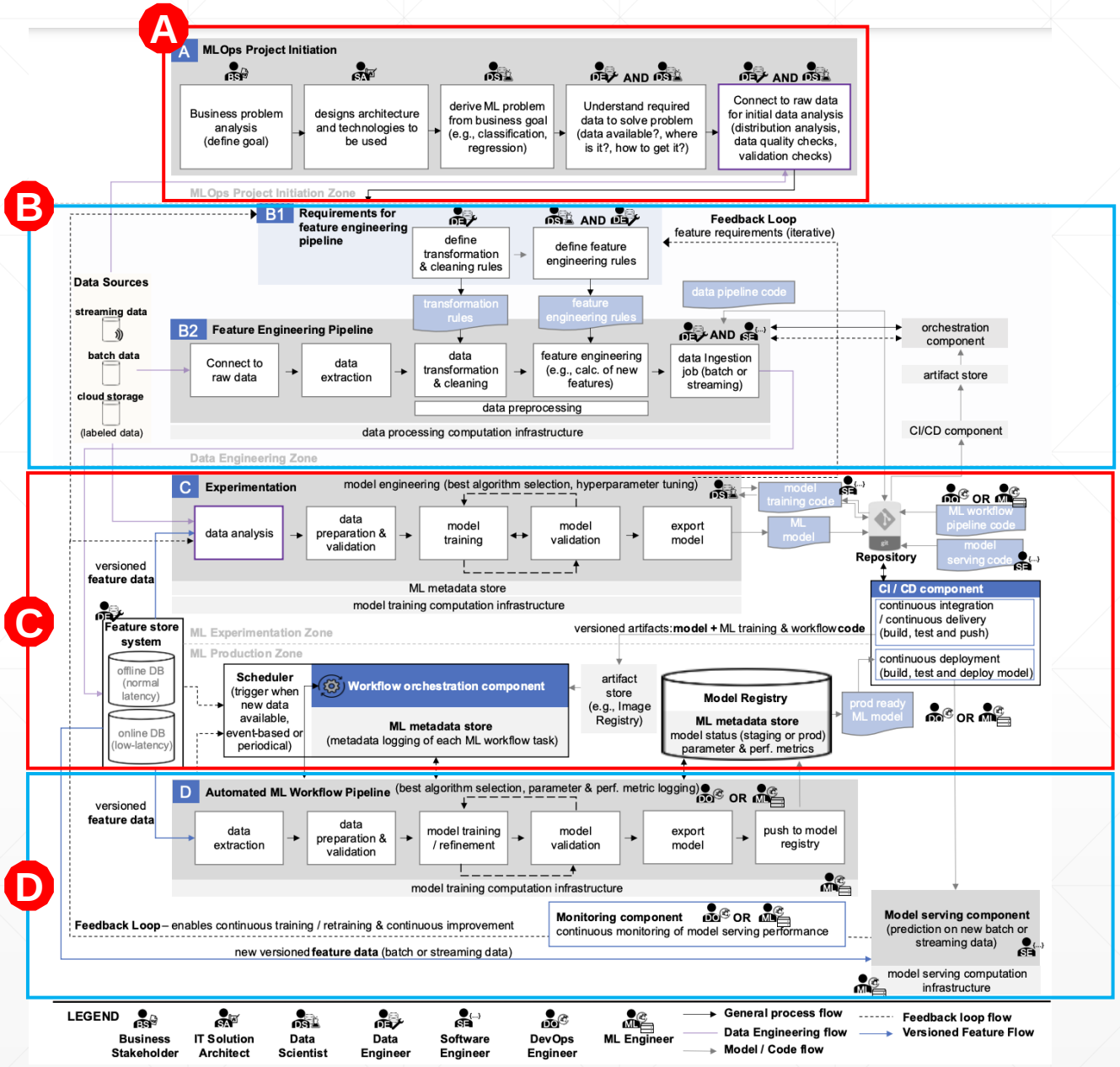
MLOps Architecture

A. **ML Ops Project Initiation**
   - Outcome: Features & Data Engineering Requirements

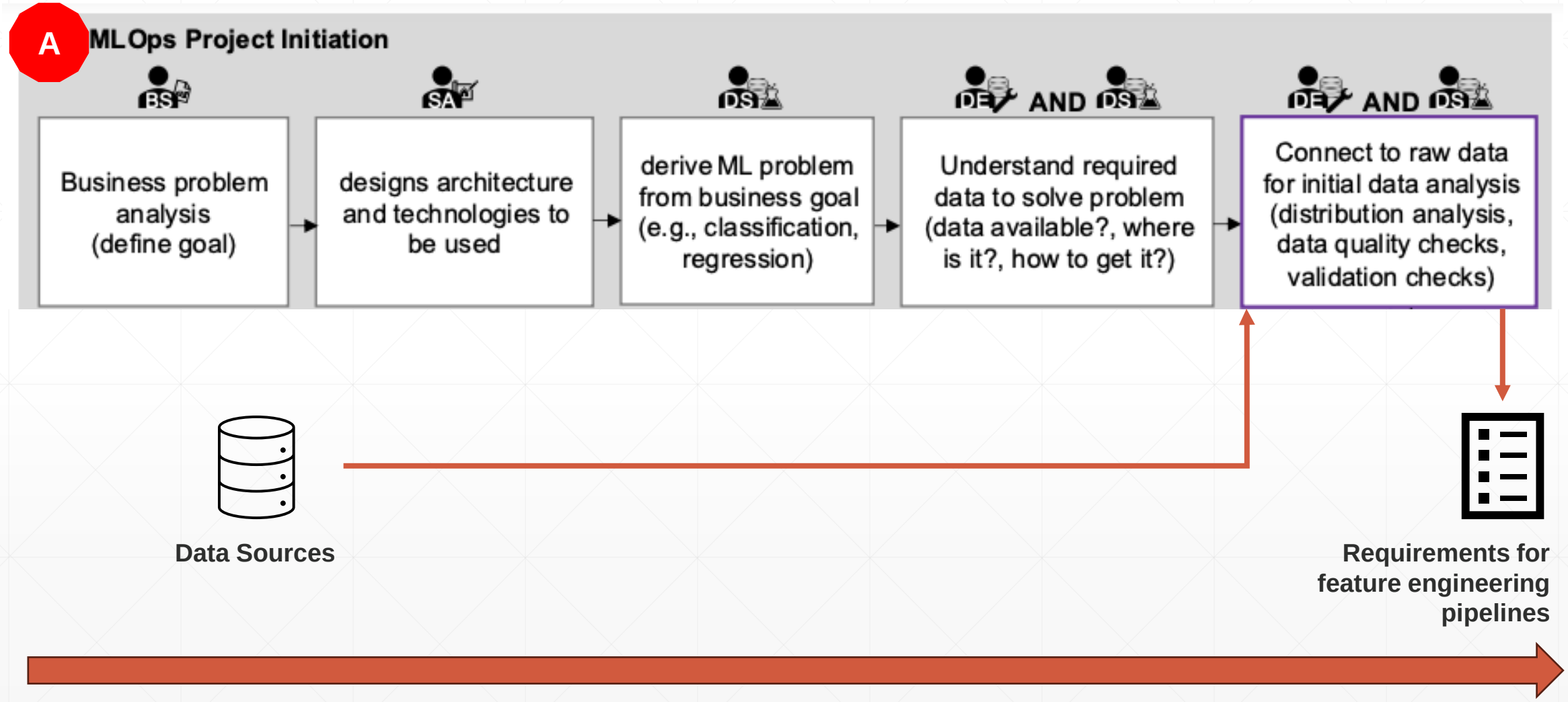
B. **Data Engineering**
   - Outcome: Data & Features engineering pipelines

C. **ML Experimentation (Dev)**
   - Outcome: Algorithms; model artifacts; model training & validation pipeline; hyperparameters; training data version

D. **Automated ML Workflow Pipeline (Prod)**
   - Outcome: automated ml workflow training & deployment; model registry; prediction service; model performance monitoring (CM); auto retraining (CT); feedback loops



MLOps Architecture - ML Ops Project Initiation



MLOps Architecture - ML Ops Project Initiation

A. ML Project Initiation

1. Business Problem Analysis: Define the business goals and objectives.

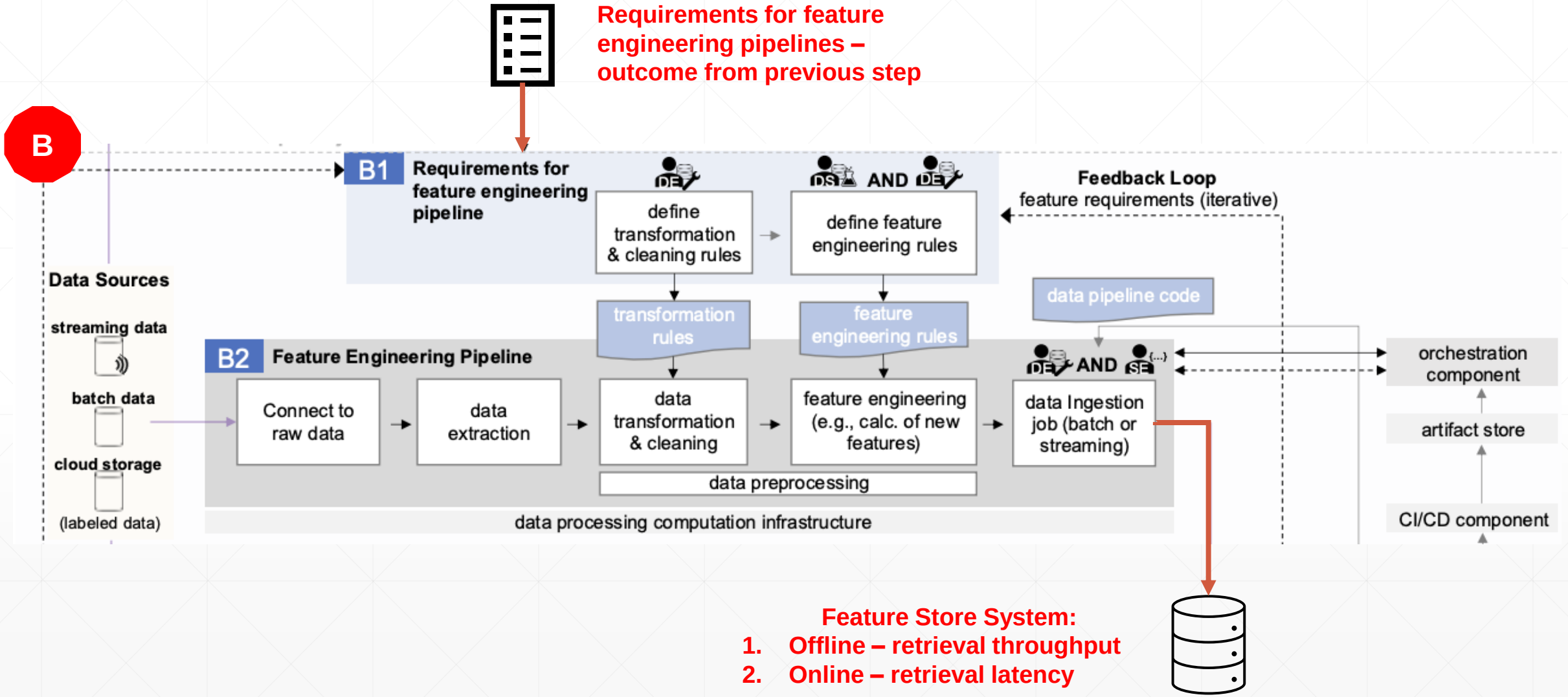
2. Design Architecture: Choose the technologies and tools for the ML solution.

3. Define ML Problem: Specify the type of ML problem (e.g., classification, regression).

4. Understand Data Requirements: Analyze data availability, distribution, quality checks, etc.

5. Connect to Raw Data Sources: Identify relevant batch or streaming data sources.

MLOps Architecture – Feature & Data Engineering



MLOps Architecture – Feature & Data Engineering

B1. Requirements for Feature Engineering Pipeline

1.Define Transformation Rules: Specify rules for cleaning and preprocessing raw data.

2.Define Feature Engineering Rules: Outline methods for creating new features from existing data.

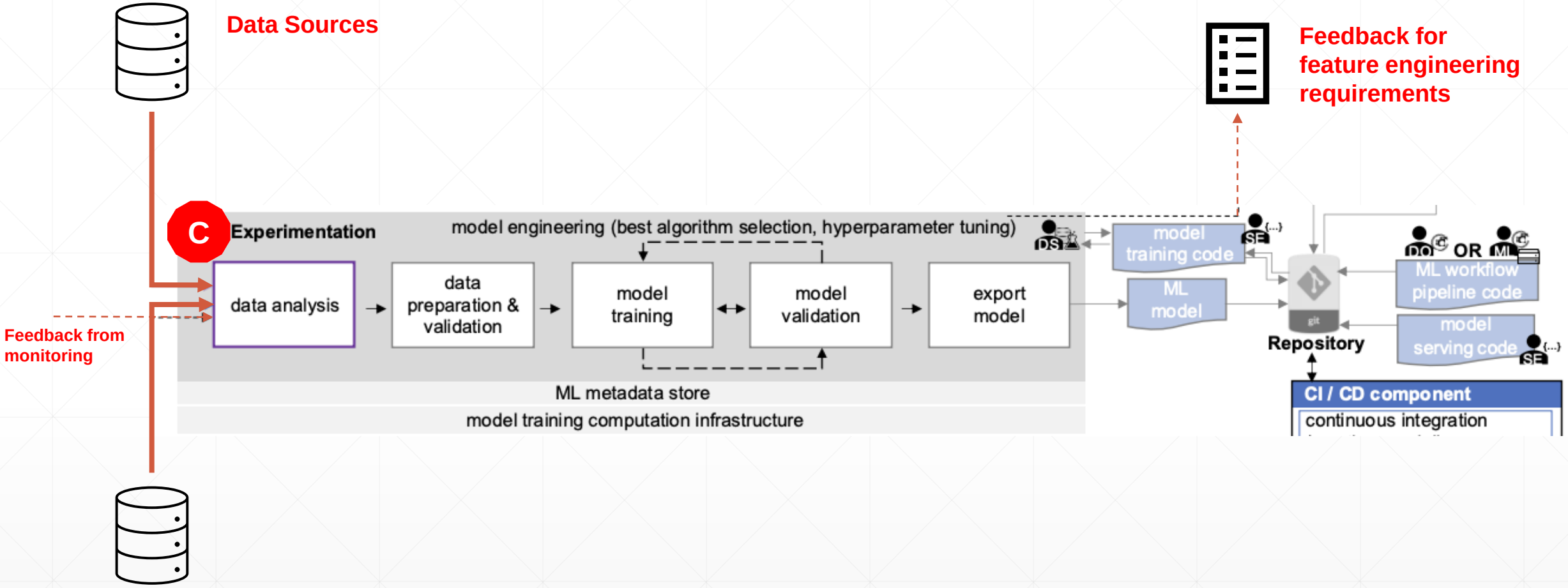
B2. Feature Engineering Pipeline

1.Data Extraction: Connect to raw data sources and extract relevant datasets.

2.Data Transformation & Cleaning: Apply transformation rules to preprocess data.

3.Feature Engineering: Generate new features based on predefined rules.

MLOps Architecture – ML Experimentation



Feature Store System:

1. **Offline – retrieval throughput**
2. **Online – retrieval latency**

MLOps Architecture – ML Experimentation

C. Experimentation

1.Data Analysis: Explore processed data for insights.

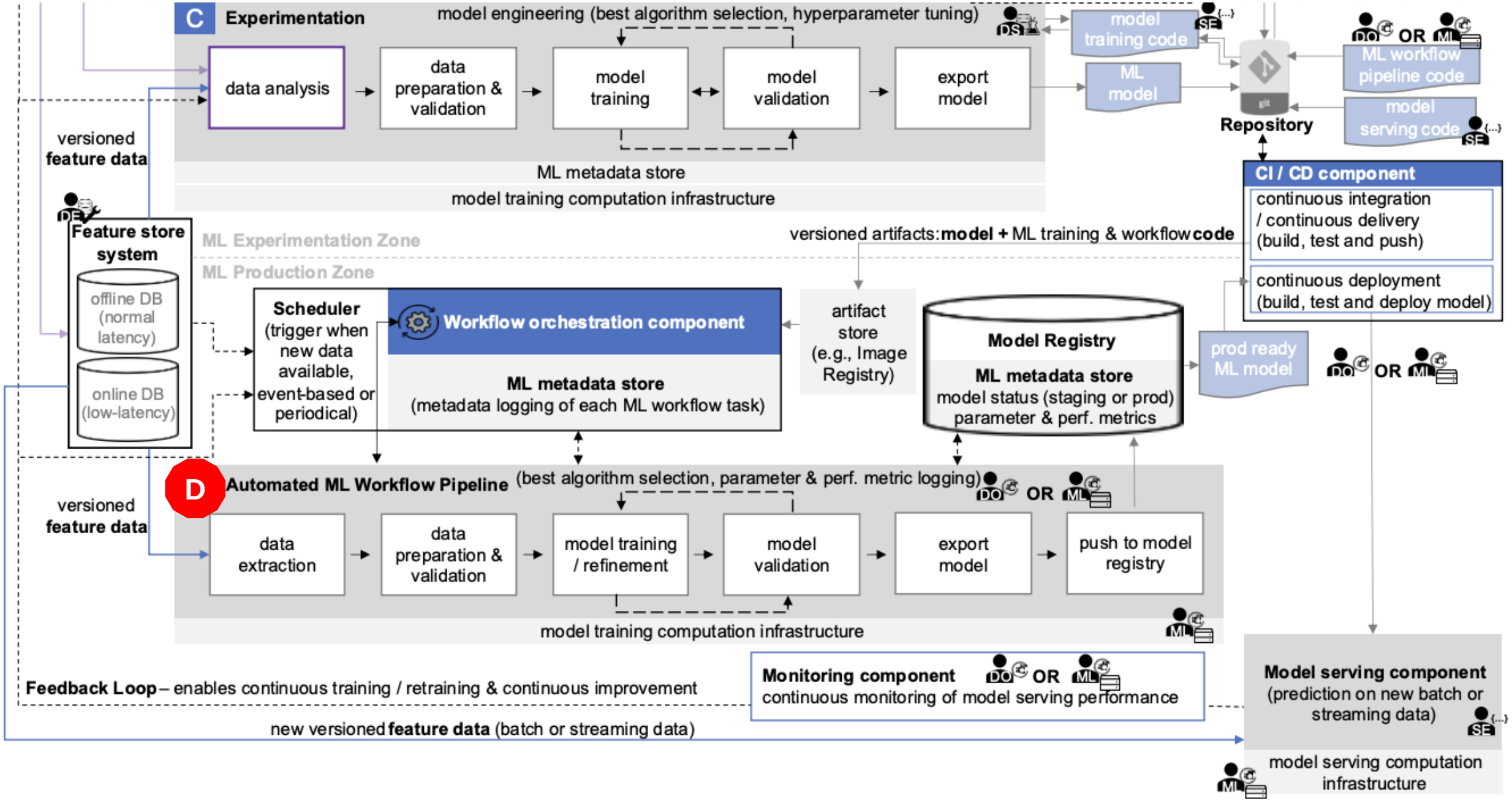
2.Data Preparation & Validation: Split datasets into training/validation sets and ensure quality.

3.Model Training: Train models using selected algorithms and hyperparameters.

4.Model Validation: Evaluate model performance using metrics.

5.Export Model: Save trained models for deployment.

MLOps Architecture – Automated ML Workflow



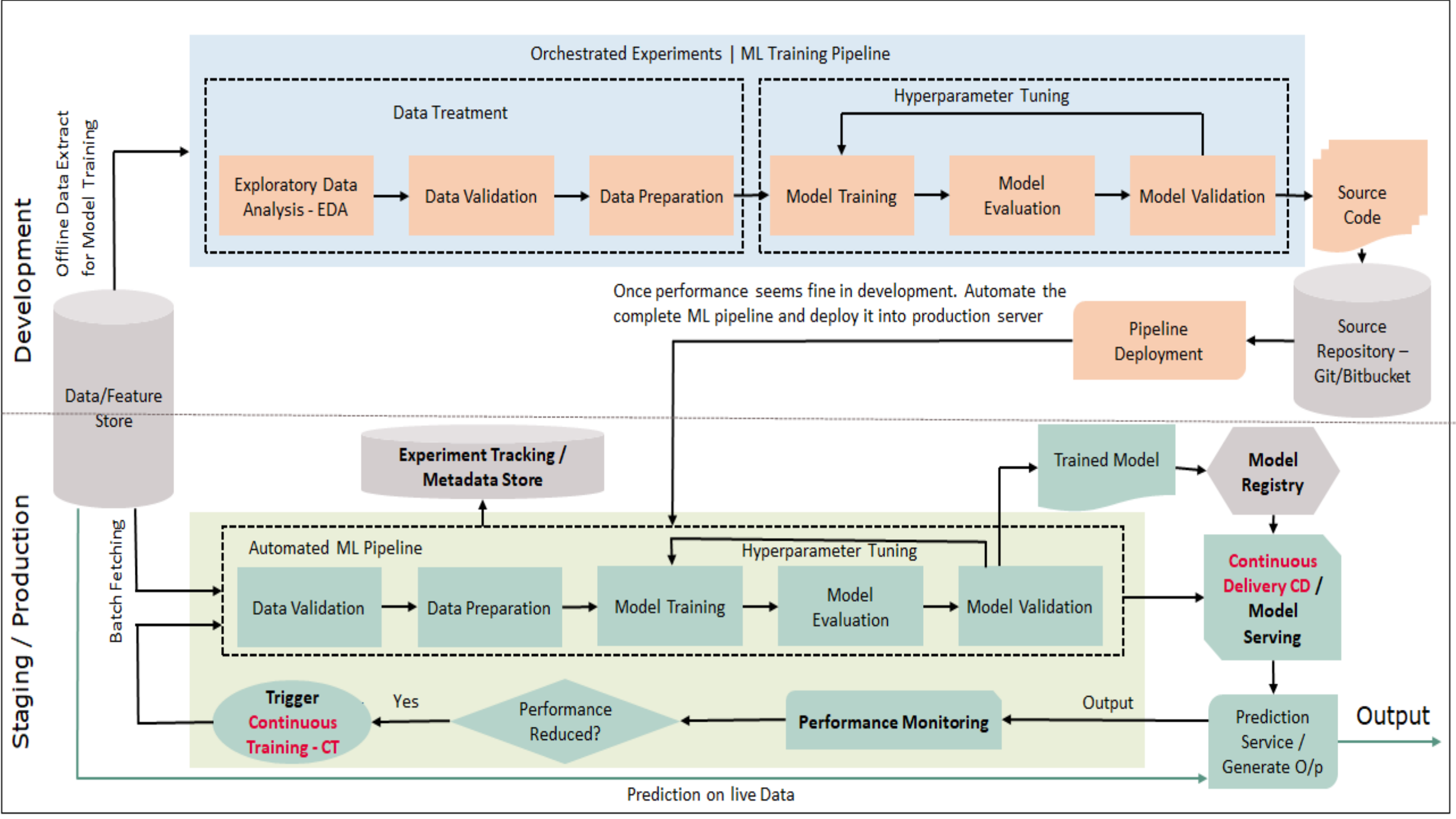
MLOps Architecture – Automated ML Workflow

D. Automated ML Workflow Pipeline

1. Scheduler Triggering: Automate workflows when new data becomes available (batch or streaming).
2. Data Preparation & Validation: Repeat preprocessing steps for new data batches.
3. Model Training/Refinement: Retrain or fine-tune models based on updated data.
4. Model Validation: Reassess model performance metrics.
5. Push to Model Registry: Store updated models in a registry (e.g., staging or production).

Additional Components

- CI/CD Pipeline for automating integration, testing, deployment of models.
- Monitoring Component to track model serving performance in production environments.


Development
Offline Data Extract for Model Training
Orchestrated Experiments | ML Training Pipeline
Data Treatment
Exploratory Data Analysis - EDA
Data Validation
Data Preparation
Hyperparameter Tuning
Model Training
Model Evaluation
Model Validation
Source Code
Source Repository – Git/Bitbucket
Pipeline Deployment
Once performance seems fine in development. Automate the complete ML pipeline and deploy it into production server
Data/Feature Store
Experiment Tracking / Metadata Store
Trained Model
Model Registry
Staging / Production
Batch Fetching
Automated ML Pipeline
Data Validation
Data Preparation
Model Training
Hyperparameter Tuning
Model Evaluation
Model Validation
Continuous Delivery CD / Model Serving
Trigger Continuous Training - CT
Yes
Performance Reduced?
Performance Monitoring
Output
Prediction Service / Generate O/p
Output
Prediction on live Data

MLOps Architecture – State-of-the-art Features

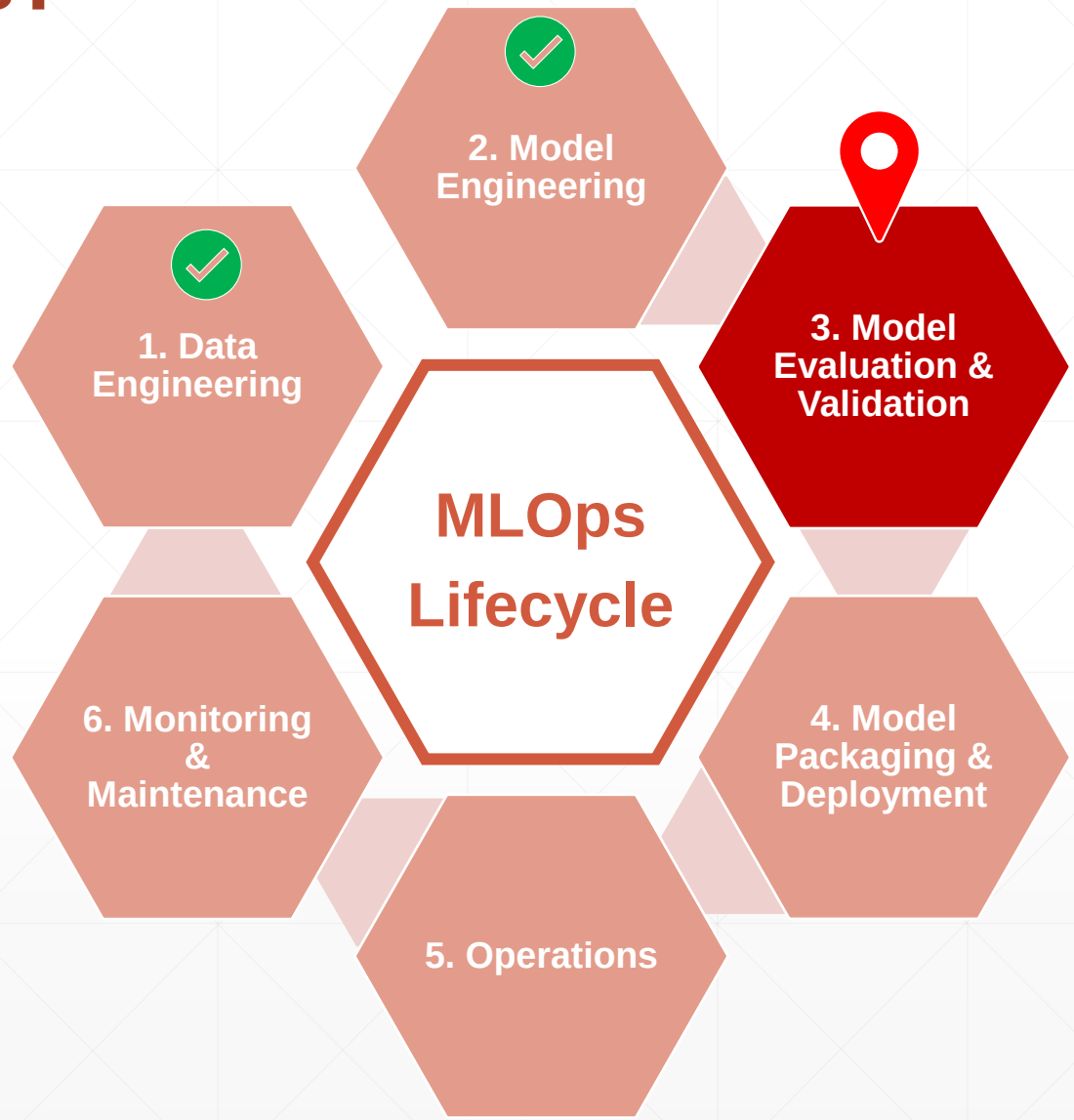
1. **Advanced Monitoring Techniques:**
    1. Real-time drift detection (data or model drift).
    2. Automated alert systems for performance degradation.
2. **Explainability & Interpretability Tools:**
    1. Integration of tools like SHAP or LIME to interpret model predictions.
3. **Robust Governance Frameworks:**
    1. Enforcing compliance with ethical AI practices and regulatory standards.
4. **Advanced Deployment Strategies:**
    1. Canary, A/B testing, Blue/Green, Active/shadow deployments or testing before full-scale rollout.
5. **Federated Learning Support:**
    1. Handling decentralized data sources with privacy-preserving techniques.
6. **Integration with Large Language Models (LLMs):**
    1. Supporting workflows for fine-tuning or serving generative AI models.
7. **Dynamic Resource Scaling:**
    1. Leveraging cloud-native solutions like Kubernetes for efficient scaling during training or inference.

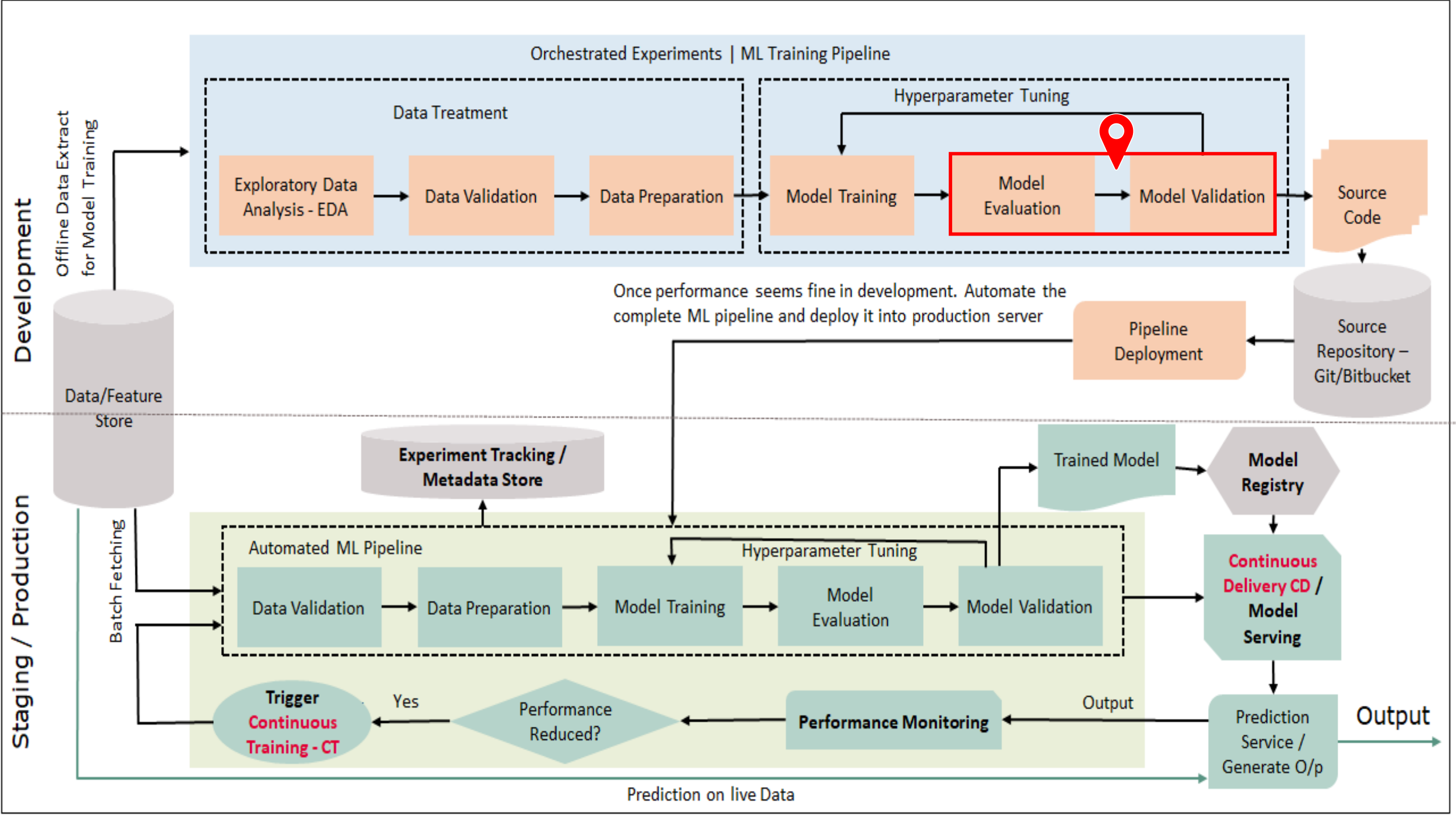
Learning Objectives

- Model Training & Evaluation
- Model Packaging & Versioning
- Model Deployment
- Model Registry
- ML Project
- Hands-On Practices with MLFlow

Position in MLOps Stages?

Define Problem & Plan




Development
Offline Data Extract for Model Training
Orchestrated Experiments | ML Training Pipeline
Data Treatment
Exploratory Data Analysis - EDA
Data Validation
Data Preparation
Hyperparameter Tuning
Model Training
Model Evaluation
Model Validation
Source Code
Source Repository – Git/Bitbucket
Pipeline Deployment
Once performance seems fine in development. Automate the complete ML pipeline and deploy it into production server
Data/Feature Store
Experiment Tracking / Metadata Store
Trained Model
Model Registry
Staging / Production
Batch Fetching
Automated ML Pipeline
Hyperparameter Tuning
Data Validation
Data Preparation
Model Training
Model Evaluation
Model Validation
Continuous Delivery CD / Model Serving
Trigger Continuous Training - CT
Yes
Performance Reduced?
Performance Monitoring
Output
Prediction Service / Generate O/p
Output
Prediction on live Data

Model Evaluation

Model Evaluation is the process of assessing the performance of a trained model to ensure it *meets the required business* or technical objectives.

Steps:

1. **Validation Techniques**: Use methods like cross-validation to assess model accuracy and robustness.
2. **Metrics Tracking**: Evaluate metrics such as precision, recall, F1-score, or RMSE based on the problem type.
3. **Comparison**: Compare the new model against the current production model using evaluation datasets.

Model Evaluation

Common Tools/Libraries:

- ***MLflow***: Tracks experiments, metrics, and artifacts for reproducibility.
- **TensorFlow Model Analysis**: Provides evaluation tools for TensorFlow models.
- **Neptune.ai**: Tracks hyperparameters, metrics, and performance visualizations.

Model Evaluation Example

Model Evaluation Example (*Technical Metrics*)
Goal: Measure accuracy and task-specific metrics for a model.
Tools: mlflow.evaluate(), built-in metrics, custom LLM-judged metrics.

```
24  # Train a model
25  model = RandomForestClassifier(random_state=42)
26  model.fit(X_train, y_train)
27
28  # Make predictions
29  y_pred = model.predict(X_test)
30
31  # Evaluate the model
32  accuracy = accuracy_score(y_test, y_pred)
33  precision = precision_score(y_test, y_pred, average="weighted")
34  recall = recall_score(y_test, y_pred, average="weighted")
35  f1 = f1_score(y_test, y_pred, average="weighted")
36  conf_matrix = confusion_matrix(y_test, y_pred)
37
38  # Print evaluation metrics
39  print("Accuracy:", accuracy)
40  print("Precision:", precision)
41  print("Recall:", recall)
42  print("F1 Score:", f1)
43  print("\nConfusion Matrix:\n", conf_matrix)
44  print("\nClassification Report:\n", classification_report(y_test, y_pred))
45
```

Model Evaluation in MLFlow

MLflow provides built-in support for evaluating models using the **mlflow.evaluate()** API.

```
# Training Data
X_train, X_test, y_train, y_test = \
    train_test_split(X, y,
    train_size=0.7,random_state=0)

# Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
# Dataset
eval_data = X_test
eval_data["test_label"] = y_test

# Evaluate model with Dataset
mlflow.evaluate(
    "runs:/run_id/model",
    eval_data,
    targets="test_label",
    model_type="regressor"
)
```

Model Validation

- Goals: Ensures the model meets business, ethical, and operational requirements.
- Scope: Broad assessment of model design, data integrity, and governance compliance
- Key question: *"Is this model safe, fair, and reliable for deployment?“*
- Timing: Occurs after evaluation but before deployment (pre-production)
- Involves fresh, unseen data and checks for bias, regulatory compliance, and robustness.
- Validation techniques / methods:
  - Fairness testing: individual, group
  - Compliance audits, ethical checks.

Model Validation - Example

Model Validation Example (Operational Checks)

Goal: Ensure the model meets fairness, latency, and compliance thresholds.

Tools: Validation thresholds, SHAP integration, Model Registry.

```python
from mlflow.models import MetricThreshold

# Set validation thresholds for deployment
thresholds = {
    "answer_correctness": MetricThreshold(threshold=0.8, higher_is_better=True),
    "latency": MetricThreshold(threshold=200, higher_is_better=False),
}

# Validate against a baseline model
validation_result = mlflow.evaluate(
    model="models:/prod_model/1",
    data=eval_data,
    model_type="question-answering",
    validation_thresholds=thresholds,
    baseline_model="models:/prod_model/0"
)

if validation_result.validation_passed:
    mlflow.register_model("runs:/<run_id>/model", "prod_model/2")
else:
    print("Validation failed: Model violates latency or correctness thresholds.")
```

Model Evaluation vs. Validation

Definitions & Objectives

Aspect	Model Evaluation	Model Validation
Primary Goal	Quantifies model performance using metrics (e.g., accuracy, F1-score) => Technical aspect.	Ensures the model meets business, ethical, and operational requirements. => Operational aspect.
Scope	Focuses on technical performance on test/holdout data	Broad assessment of model design, data integrity, and governance compliance
Key Question	*"How well does the model predict?"*	*"Is this model safe, fair, and reliable for deployment?"*

Model Evaluation vs. Validation

Methods & Techniques

Category	Model Evaluation	Model Validation
Core Techniques	- Train-test splits - Cross-validation - Metric calculation (e.g., ROC AUC, MSE)	- Bias testing (data slicing by demographic groups) - Stress testing - Compliance audits
Focus Areas	- Algorithm performance - Overfitting detection	- Ethical AI practices - Data lineage verification - Model interpretability

Model Evaluation vs. Validation

Tools & Libs

Purpose	Model Evaluation Tools	Model Validation Tools
Performance	- MLflow (experiment tracking) - Neptune.ai (metric visualization)	- Aequitas (bias detection) - IBM AI Fairness 360
Governance	N/A	- MLflow Model Registry (version control) - AWS SageMaker Model Monitor
Compliance	N/A	- TensorFlow Data Validation (TFDV) - Great Expectations

Model Evaluation vs. Validation

In Mlflow Context

Aspect	Model Evaluation (MLflow)	Model Validation (MLflow)
Focus	Accuracy, F1-score, Rouge	Latency, fairness, regulatory compliance
Tools Used	mlflow.evaluate(), custom metrics	Validation thresholds, Model Registry, SHAP
Output	Metric tables (eval_results_table)	Pass/fail status, governance metadata
Stage	During experimentation	Pre-deployment
Example Metric	answer_correctness (GPT-4 judged)	toxicity < 0.1 (via mlflow.metrics.toxicity())

Model Evaluation & Validation in MLFlow - Practice

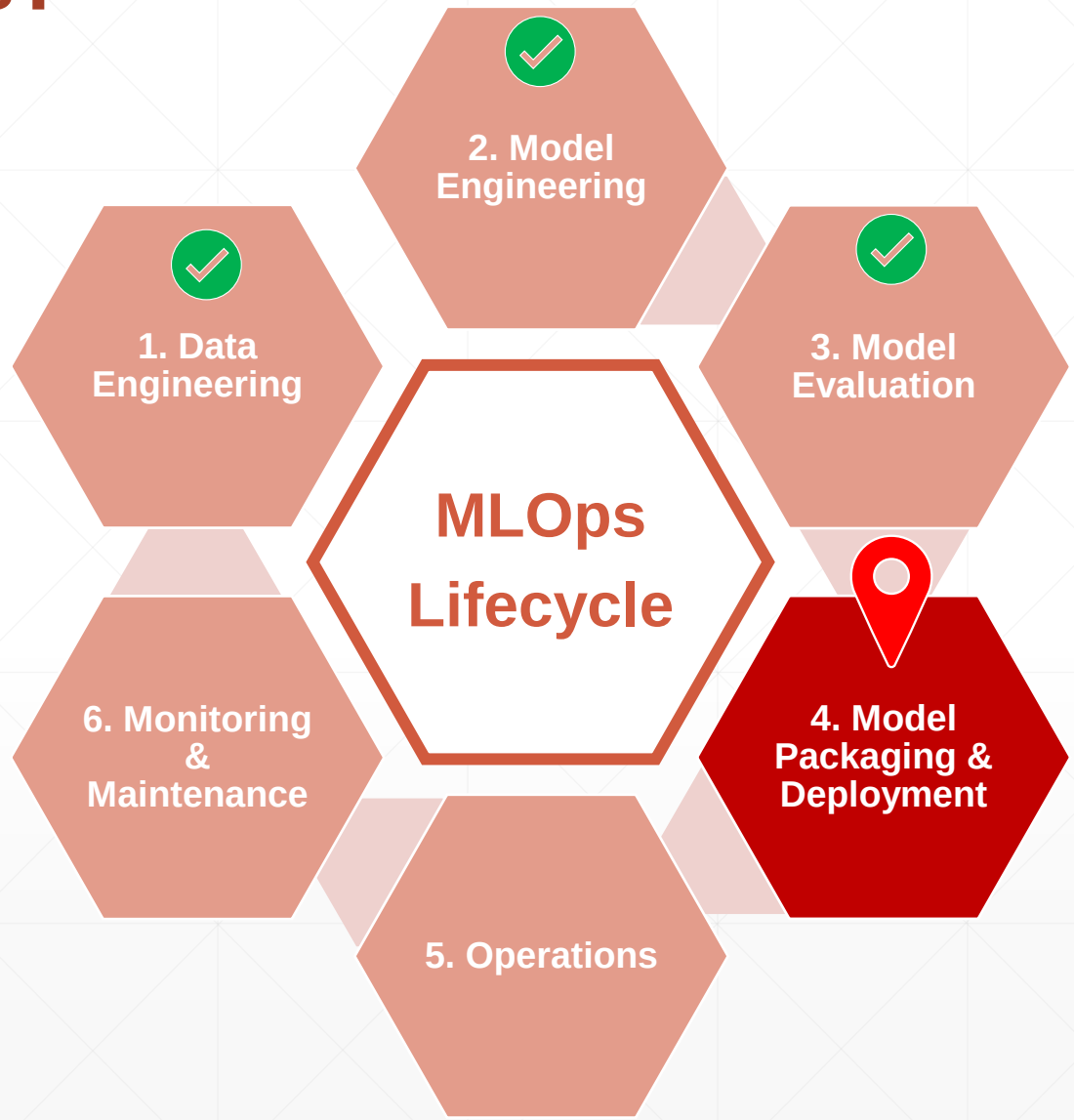
Visit: MLflow Models Evaluation

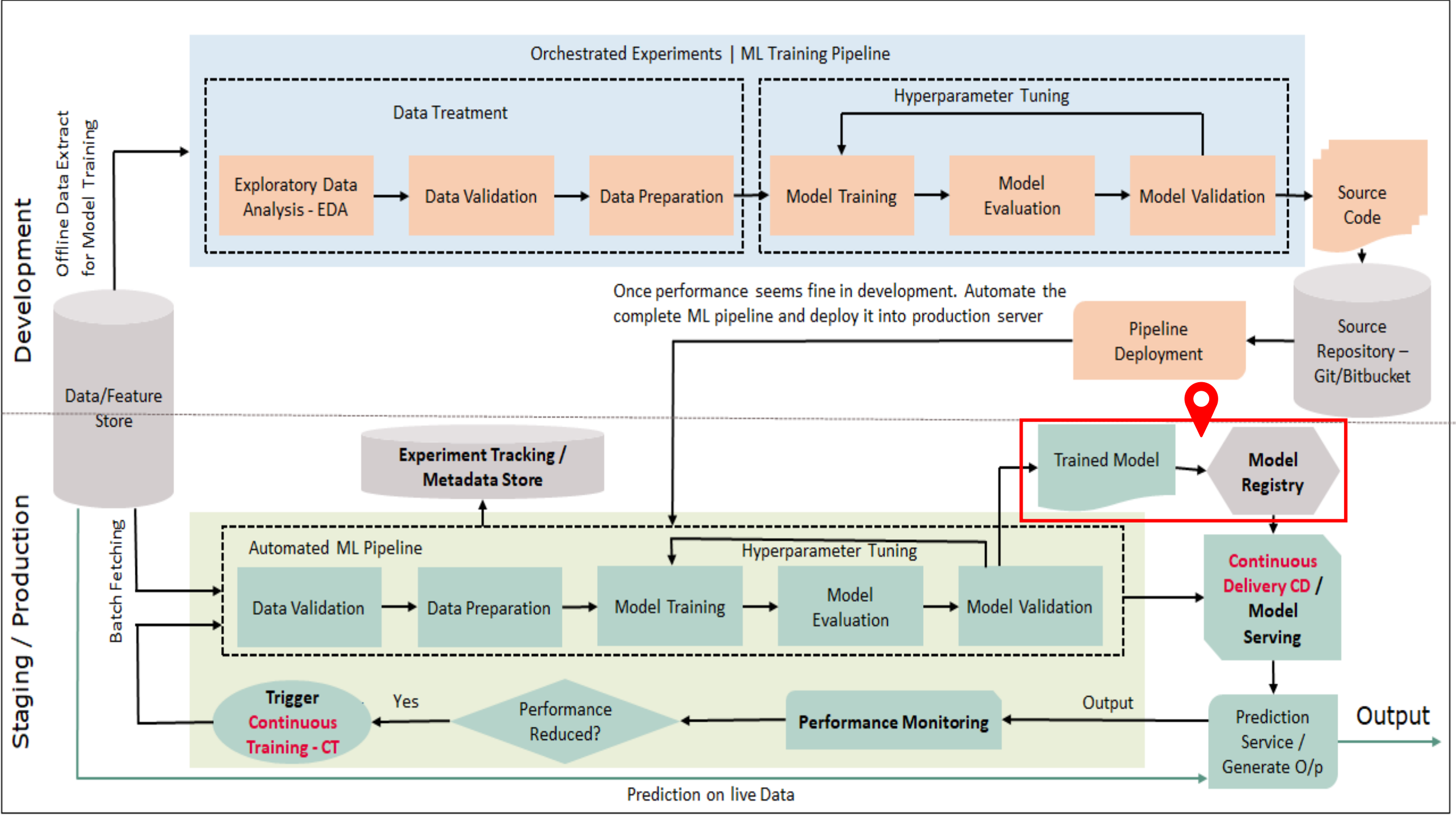
[In-Class] Hands on following practices:
- Model Evaluation
- Validate Models before Deployment

[Homework]
- Model Customization
- Batch inference on Apache Spark

Position in MLOps Stages?

Define Problem & Plan




Development
Offline Data Extract for Model Training
Orchestrated Experiments | ML Training Pipeline
Data Treatment
Exploratory Data Analysis - EDA
Data Validation
Data Preparation
Hyperparameter Tuning
Model Training
Model Evaluation
Model Validation
Source Code
Source Repository – Git/Bitbucket
Once performance seems fine in development. Automate the complete ML pipeline and deploy it into production server
Pipeline Deployment
Data/Feature Store
Experiment Tracking / Metadata Store
Trained Model
Model Registry
Staging / Production
Batch Fetching
Automated ML Pipeline
Data Validation
Data Preparation
Model Training
Model Evaluation
Model Validation
Hyperparameter Tuning
Continuous Delivery CD / Model Serving
Trigger Continuous Training - CT
Yes
Performance Reduced?
Performance Monitoring
Output
Prediction Service / Generate O/p
Output
Prediction on live Data

Model Packaging

Model packaging involves bundling the model with its dependencies for deployment.

Steps:

1. **Artifact Creation**: Package the trained model along with dependencies, configuration files, and metadata into a deployable format (e.g., Docker container or serialized file).
2. **Standardized APIs**: Ensure compatibility with serving frameworks like TensorFlow Serving or ONNX Runtime.
3. **Versioning**: Maintain multiple versions of the model for rollback and updates.

Model Packaging

Common Tools/Libraries:

- **Docker**: Containerizes models for consistent deployment environments.
- **TensorFlow Serving**: Serves TensorFlow models efficiently in production.
- **ONNX Runtime**: Optimizes cross-platform model deployment.

Model Packaging Example

```
# Load dataset
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=42)

# Train a model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Log the model with MLflow
with mlflow.start_run():
    mlflow.sklearn.log_model(model, "model")

# Save the model URI
model_uri = f"runs:/{mlflow.active_run().info.run_id}/model"

# Build a Docker image for the model
mlflow.models.build_docker(model_uri=model_uri, name="mlflow-docker-model")

# Generate a Dockerfile for the model
mlflow.models.generate_dockerfile(model_uri=model_uri, output_path="Dockerfile")
```

Model Versioning

Model versioning tracks changes in models over time to ensure reproducibility and rollback capability.

Steps:

1. Version Control: Use tools to manage changes in training data, code, and hyperparameters.
2. Logging Metadata: Store metadata such as training data versions, preprocessing steps, and evaluation metrics.
3. Registry Management: Register models in a centralized repository for easy access during deployment or rollback.

Model Versioning

Common Tools/Libraries:

- MLflow Model Registry: Manages model versions and lifecycle stages (e.g., staging, production).
- Git/GitHub: Tracks code changes related to model development.
- DVC (Data Version Control): Tracks changes in data and pipelines alongside code

Model Registry – Concepts

A model registry is a centralized system in MLOps that manages the lifecycle of machine learning models, enabling **version control**, **metadata tracking**, and seamless **collaboration** between teams.

It bridges the gap between model development and production deployment by **standardizing** how models are **stored**, **accessed**, and **monitored**.

Model Registry – Mlflow Concepts

Model – is created from an experiment or run is logged with one of the model flavor. Once logged, the model can then be registered with the Model Registry.

Registered model – has a unique name; contains versions; aliases; tags and other metadata.

Model version – one registered model can have multiple versions. New model is registered as version 1 and gets increased for each new model registered.

Model alias – mutable, named reference to a particular version of a registered model. E.g., “Champion”, “Challenger”, …

Tags – key-value pairs; to label and categorize registered models and versions. E.g., “validation_status:pending”

Model Registry – Practice: Model Registry UI

Visit this link: MLflow Model Registry | MLflow

Practice following steps:

- Register a Model (a model or run must be logged using `log_model` API first)
- Find registered models
- Deploy and organize models

Model Registry – Practice: Model Registry API

Visit this link: MLflow Model Registry | MLflow

Practice the steps which require local deployment in the given document:

- Adding an Mlflow model to the Model Registry
- Deploy and Organize Models with Aliases and Tags
- Fetching an Mlflow model from the Model Registry
- Serving an Mlflow Model from Model Registry

Deployment Architecture

Separate environments for each step in MLOps workflows.

Each environment (workspace) has its own dedicated Mlflow Tracking server to which metrics, parameters and model artifacts are logged.

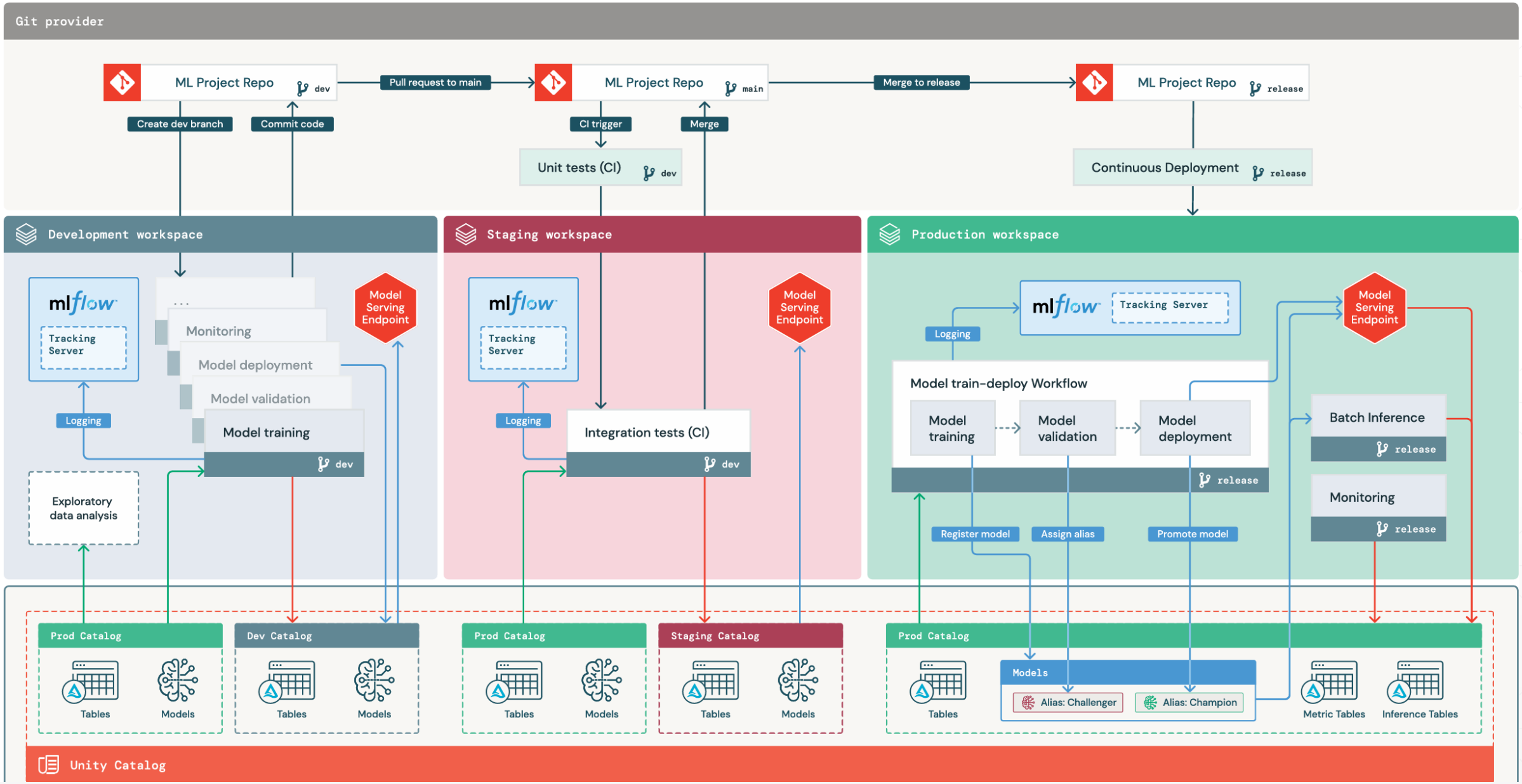
Environments: dev, staging, prod.

Model Stages: None; Staging; Production; Archived.

Purpose: Access control – the matter of “who can do what”.

In MLFlow: Use Mlflow Authentication. More details: *MLflow Authentication | MLflow*

Multi-Environment View (Databricks Ref)



Multi-Environment View (Databricks Ref)

1. **Development:** ML code is developed in the development environment, with code pushed to a dev (or feature) branch.
2. **Testing:** Upon making a pull request from the dev branch to the main branch, a CI trigger runs unit tests on the CI runner and integration tests in the staging environment.
3. **Merge code:** After successfully passing these tests, changes are merged from the dev branch to the main branch.
4. **Release code:** The release branch is cut from the main branch, and doing so deploys the project ML pipelines to the production environment.

Multi-Environment View (Databricks Ref)

5. **Model training and validation:** The model training pipeline ingests data from the prod catalog. Upon validating, the resulting model artifact is registered to the prod catalog. A “Challenger” alias is attached to the newly registered model version.

6. **Model deployment:** A model deployment pipeline evaluates the current “Champion” model versus “Challenger” model, with the best-performing model version taking the “Champion” alias after this evaluation.

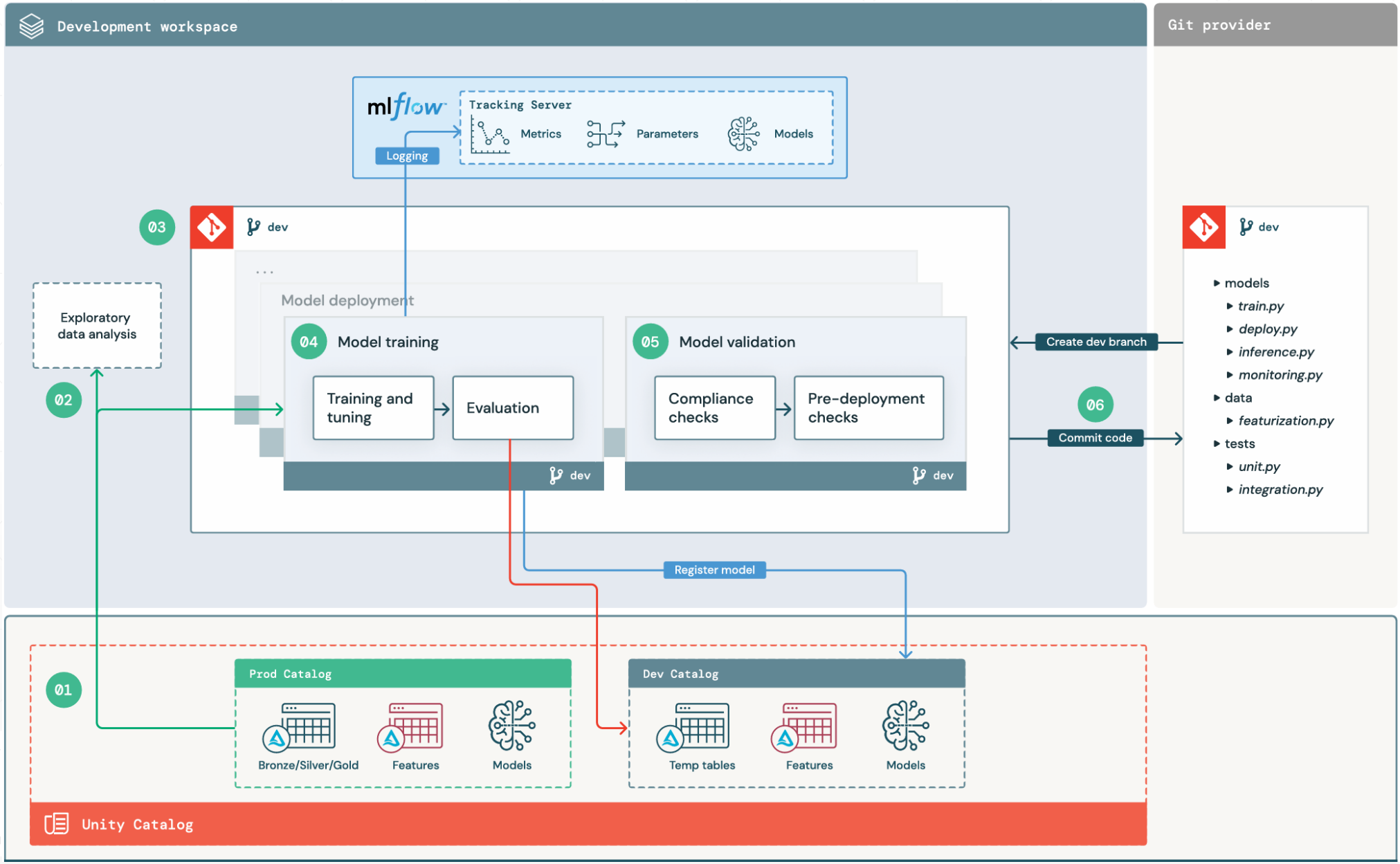
7. **Model inference:** Model Serving or other inference pipelines load the “Champion” model to compute predictions. Predictions are logged to inference tables, which can be used to monitor the “Champion” model’s performance.

8. **Monitoring:** Scheduled or continuous pipeline to refresh Lakehouse Monitoring metric tables. Inference tables are monitored to detect data or model drift. Databricks SQL dashboards are automatically created to display monitor metrics.

Development Environment

- Production data can be access (read only) from development environment, allowing data scientist to develop ML code using production data.
- Write access is restricted to only the development environment (data catalog, storage, ..)

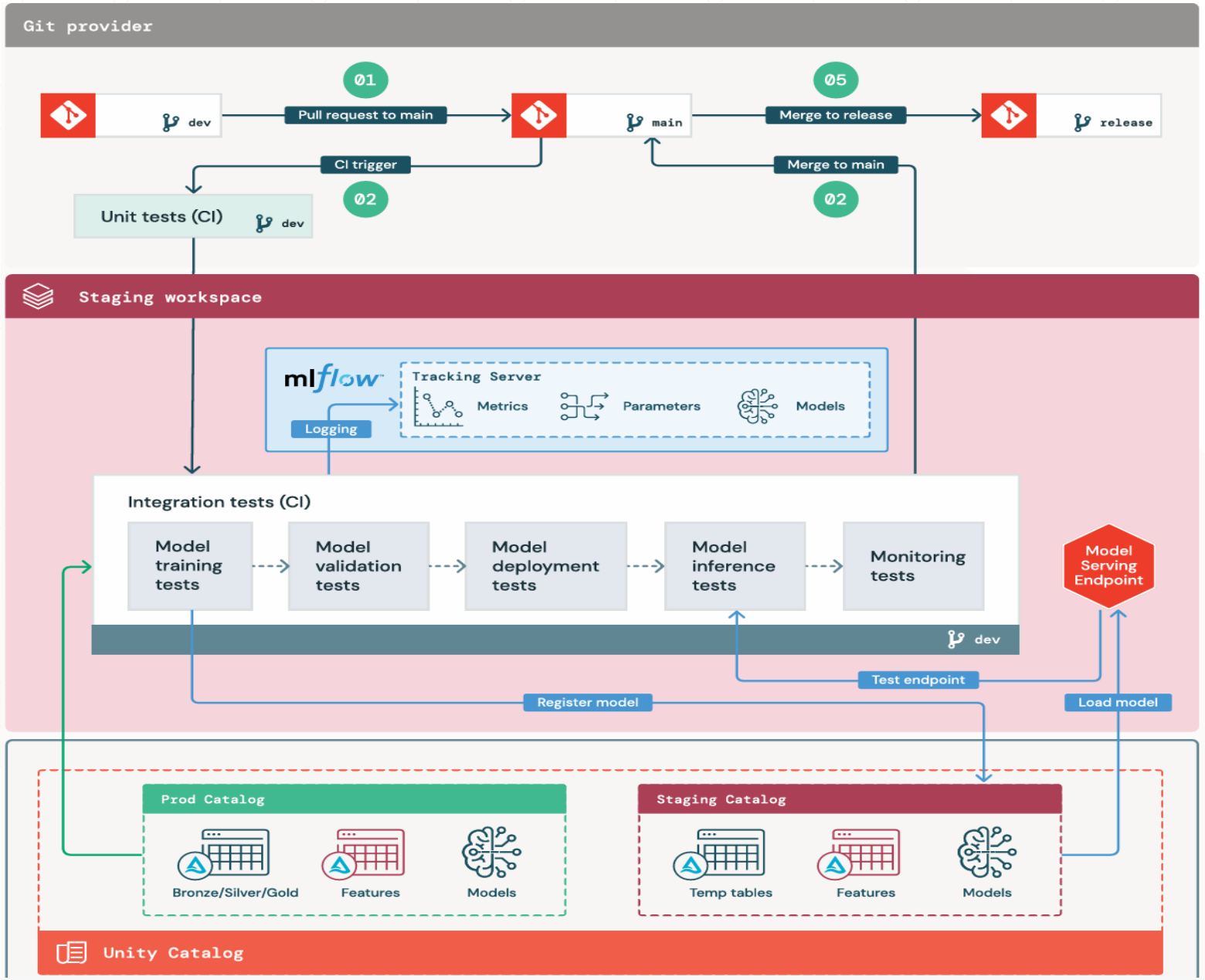
Development Environment



Staging Environment

- All codes are rigorously tested: Unit Tests; Integration Tests; Pre-deployment Tests;
- Continuous Integration (CI) pipelines and orchestration.
- Train and test with small data to ensure the integrations.

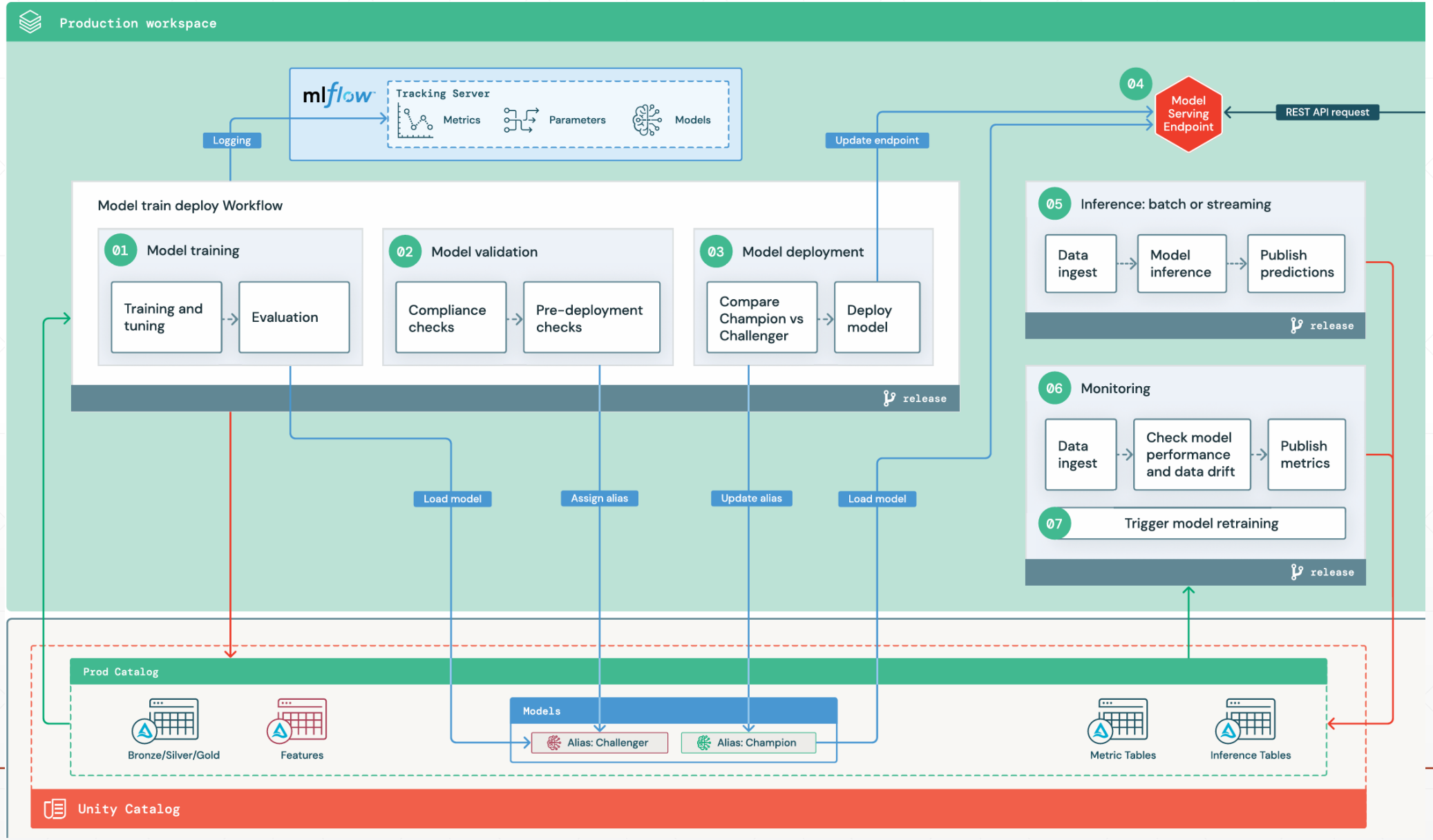
Staging Environment



Legend

- Workflow
- Job/Workflow task
- CI/CD pipeline
- Reads
- Writes
- MLflow API
- Git repo
- Git branch
- Registered model

Production Environment



Production Environment

- Data Scientist and ML Engineer have no access.
- Training:
  - The optimal set of hyperparameters for a model may be determined in the development environment / stage.
  - Only top performing options (algorithms & hyperparameters) are selected to the training codes.
  - Tuning is restricted. It will be performed through automated retraining.
- Evaluation: Testing held-out production data. Log results to Mlflow tracking server.
- Register: New trained model is then registered as a new version.
- Validation: everything from basic format and metadata; performance; compliance, etc.
- Passed validation: alias “Challenger” is assigned to the new model version.
- Deployment: Compare “Challenger” vs “”Champion” (existing running). Held-out data for offline model. For online: A/B, Canary, …

Model Registry Environments – Practice: Promoting

Visit this link: MLflow Model Registry | MLflow

Practice the following steps:

- Promoting an Mlflow model across environments
- *And other essential APIs …*

Model Registry – Practice: Authentication

Visit this link: MLflow Authentication | MLflow

Understand the core concepts of Mlflow authentication. Do practice following:

- Authenticate to MLflow
- Creating a new user

Pre-Deployment Model Testing

Deployment testing metrics:

- Latency: system response time
- Throughput: Queries per second - QPS; Transactions per second - TPS
- Standard load evaluation: analyse system behaviour under expected request volume, scaling from regular to anticipated peak levels.
- Stress Assessment: deliberately overwhelm system to observe its response to abnormally high demands, looking for graceful failure and effective recovery.

Real-time Model Deployment Strategies

Real-time models often require a paradigm of online model evaluation to ensure that the most accurate model is always serving predictions. This may involve updating a model on a more regular basis based on newly arriving data. As a result, this necessitates a different approach to deployment.

Model Deployment Strategies: A/B Testing

A/B Testing:

- Deploy multiple model versions concurrently and distribute the traffic among them to test and evaluate their performance. Based on predefined success criteria, such as accuracy or conversion rate, the best-performing model is then selected to handle all traffic.
- A/B testing usually involves splitting traffic evenly or in some predefined ration between model versions. Unlike gradual rollouts, where traffic is incrementally shifted based on performance metrics, A/B testing maintains its traffic splits until a decision is made.

Model Deployment Strategies: Gradual Rollout

Gradual Rollout (or Canary Deployment)

- Begins by exposing a new model version to a small selected segment of request traffic
- Performance is closely monitored
- If the new model version meets defined success criteria, traffic to this model version is gradually increased.
- Continuous expose to real traffic while still having the ability of scaling back if anomalies arise

Model Deployment Strategies: Shadow

Shadow Deployment (Active – Standby)

- New model version runs alongside the existing version but does not actively serve traffic
- New version receives a copy of the incoming traffic and generates predictions, which are logged to compare to the current version for evaluation purposes
- No affecting user experience
- If the shadow version is performing better the active model, this version will become active, and the other one will be swapped to shadow.

Model Deployment Methods

- Deploy model artifacts
- Deploy model build pipelines – training code

Model Deployment Strategies - Overview

Strategy	Description	Benefits	Challenges	Use Cases
Batch Deployment	Processes large volumes of data at intervals.	- Resource efficiency (scheduled during off-peak hours)	- Not suitable for real-time applications	Fraud detection, predictive maintenance, market analysis
Real-Time Deployment	Serves predictions instantly as data arrives.	- Immediate insights, critical for time-sensitive applications	- Requires robust infrastructure and continuous monitoring	Self-driving cars, healthcare diagnostics
Edge Deployment	Deploys models on edge devices for local processing.	- Low latency, reduced dependency on cloud resources	- Limited computational resources on edge devices	IoT applications, autonomous drones
Canary Deployment	Gradually exposes the new model to a subset of real users.	- Detects issues early with real-world feedback	- Requires careful monitoring and rollback planning	Testing new features with live users

Model Deployment Strategies - Overview

Strategy	Description	Benefits	Challenges	Use Cases
Blue-Green Deployment	Maintains two environments (blue: current, green: new) and switches traffic.	- Minimal downtime and easy rollback	- Requires duplicate infrastructure	Large-scale production updates
Rolling Deployment	Updates instances incrementally without downtime.	- Faster than blue-green deployments	- Potential inconsistency during the transition	Scaled services with multiple instances
A/B Testing	Compares two model versions by splitting traffic between them.	- Provides direct comparison of model performance	- Requires careful design to avoid bias	Marketing campaigns, recommendation systems
Shadow Deployment	Runs the new model alongside the current one without affecting live users.	- Safe testing without impacting users	- Does not provide real-world feedback	Testing model performance under production conditions

Model Deployment Practices

Visit this: Deploy MLflow Model as a Local Inference Server | MLflow

[In-Class] Practice following:

- Deploying Inference Server
- Inference Server Specification
- Running batch inference

[Homework]

- The rest.

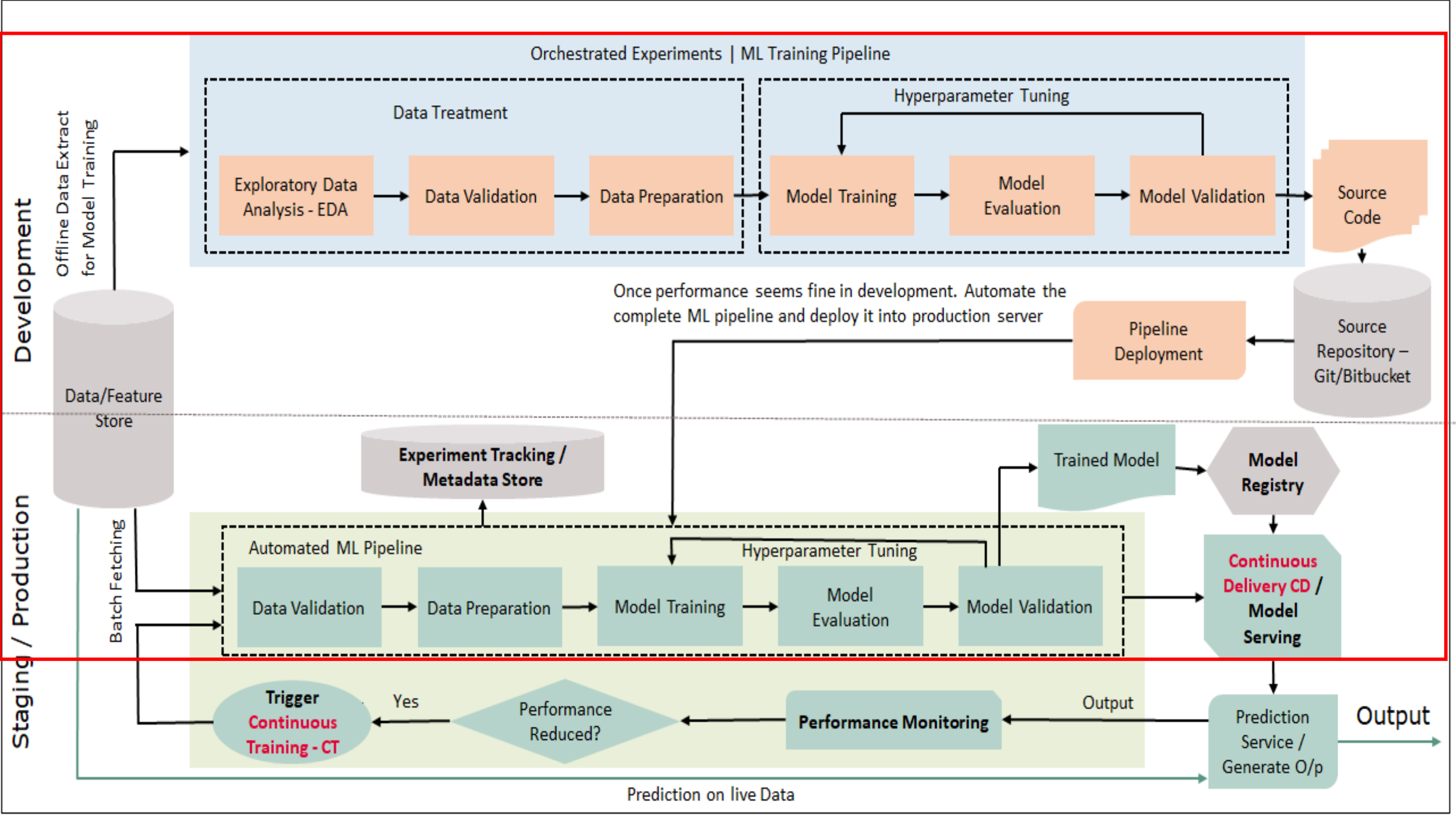
Team Collaboration (WIP)

- Self-Hosting
- Managed Services
- Access Control (Authentication)
- Mlflow Projects

Automation Tools & Libraries

Tool	Primary Stage(s) Involved	Purpose
AutoML	Model Training, Model Evaluation	Automates model selection, feature engineering, and hyperparameter tuning
Optuna	Model Training (Hyperparameter Optimization)	Efficient hyperparameter tuning using pruning and Bayesian optimization
Scikit-Optimize	Model Training (Hyperparameter Optimization)	Bayesian optimization for hyperparameters, built on top of scikit-learn

And many other tools …


Development
Staging / Production
Orchestrated Experiments | ML Training Pipeline
Data Treatment
Exploratory Data Analysis - EDA
Data Validation
Data Preparation
Hyperparameter Tuning
Model Training
Model Evaluation
Model Validation
Source Code
Offline Data Extract for Model Training
Data/Feature Store
Once performance seems fine in development. Automate the complete ML pipeline and deploy it into production server
Pipeline Deployment
Source Repository – Git/Bitbucket
Experiment Tracking / Metadata Store
Trained Model
Model Registry
Batch Fetching
Automated ML Pipeline
Hyperparameter Tuning
Data Validation
Data Preparation
Model Training
Model Evaluation
Model Validation
Continuous Delivery CD / Model Serving
Trigger Continuous Training - CT
Yes
Performance Reduced?
Performance Monitoring
Output
Prediction Service / Generate O/p
Output
Prediction on live Data

Let’s Practice – MLflow Models