Lecture 09: MLOps and The Importance of Good Data

CSCI E-103: Reproducible Data Science and Machine Learning

Harvard University

■ **Course:**	CSCI E-103: Reproducible Data Science
■ **Week:**	Lecture 09
■ **Instructors:**	Anindita Mahapatra & Eric Gieseke
■ **Objective:**	Master MLOps workflows, understand the roles involved, and learn why data quality matters more than model complexity

Contents

1 What is MLOps and Why Does It Matter?

Lecture Overview

This lecture focuses on the **operational** side of machine learning: how to take a model from a Jupyter notebook to a production system that reliably delivers value.

Key Topics:

- The definition and importance of MLOps
- The five key personas in ML projects
- The three core pipelines: Training, Inferencing, Monitoring
- Model-Centric vs. Data-Centric AI
- Deployment strategies: Deploy Models vs. Deploy Code
- Hyperparameter tuning and model selection
- End-to-end MLOps lab walkthrough (Customer Churn Prediction)

1.1 From DevOps to MLOps

In traditional software, **DevOps** (Development + Operations) revolutionized how we build and deploy applications by automating builds, tests, and deployments.

But AI systems are fundamentally different:

- **Traditional Software = Code**
- **AI Systems = Code (Model/Algorithm) + Data**

Because AI systems depend on **both** code and data, they are sensitive to changes in either. A model trained on last year's data might perform poorly on this year's data, even if the code hasn't changed.

Definition: MLOps

MLOps (Machine Learning Operations) is the set of standards, tools, processes, and methodologies that aim to:

- **Optimize time and efficiency** in ML development
- **Ensure quality** of models in production
- **Guarantee governance** (security, compliance, auditability)

It's the intersection of **Machine Learning**, **Data Engineering**, and **DevOps**.

1.2 Business Context: Customer Churn Prediction

Every ML project starts with a business problem. Let's use **customer churn prediction** as our running example.

Example: The Business Case for Churn Prediction

The Problem:

- Customers sign up for a service, use it for a while, then leave ("churn")
- Acquiring a new customer costs 5-10x more than retaining an existing one
- The business wants to **predict** which customers are about to leave **before** they do

The ML Solution:

- Build a model that scores each customer's "churn risk"
- When a customer is flagged as high-risk, offer them a discount or incentive to stay
- Result: Reduced churn, saved revenue, happier customers

Why MLOps?

- The model needs to be trained on historical data
- It needs to be deployed so the marketing team can act on predictions
- It needs to be monitored—customer behavior changes over time!
- If performance degrades, it needs to be automatically retrained

MLOps is the system that makes all of this happen reliably and repeatedly.

2 The Five Key Personas in MLOps

MLOps is not a solo endeavor. It requires collaboration between specialists with different skills.

2.1 1. Business Stakeholder / Business Analyst

- **Primary Mission:** Translate vague business goals into clear, measurable ML problems
- **Key Responsibilities:**
  - **Define the problem:** "What is churn?" → "A customer who hasn't purchased in 90 days OR cancelled their subscription within 30 days"
  - **Set KPIs:** How will success be measured? (e.g., reduction in churn rate, ROI of interventions)
  - **Stakeholder alignment:** Work with marketing, sales, product, and finance to ensure the model serves real business needs
  - **Validate results:** After deployment, verify that the model is actually reducing churn
- **Deliverables:** Problem statement, data requirements, evaluation criteria, business impact analysis

2.2 2. Data Engineer (DE)

- **Primary Mission:** Build and maintain the data infrastructure
- **Key Responsibilities:**
  - **Data collection:** Pull data from CRM systems, web logs, billing, support tickets
  - **ETL/ELT pipelines:** Extract, Transform, Load data into usable formats
  - **Data quality:** Handle missing values, duplicates, inconsistencies, schema validation
  - **Infrastructure:** Manage scalable storage on AWS, Azure, or GCP
  - **Lineage & versioning:** Track where data came from and which version was used
- **Deliverables:** Production-grade, trusted datasets ready for analysis and modeling

2.3 3. Data Scientist (DS)

- **Primary Mission:** Transform data into predictive insights (build the model)
- **Key Responsibilities:**
  - **Exploratory Data Analysis (EDA):** Understand patterns in the data
  - **Feature Engineering:** Create meaningful variables (e.g., "number of support tickets in last 30 days")
  - **Model Selection & Training:** Experiment with algorithms (Logistic Regression, Random Forest, XGBoost)
  - **Evaluation:** Measure performance using accuracy, precision, recall, F1
  - **Interpretation:** Understand which factors drive churn (feature importance)
  - **Collaboration:** Work with business stakeholders to validate that the model makes sense
- **Deliverables:** Trained models, feature sets, validation reports, insights

2.4 4. Machine Learning Engineer (MLE)

- **Primary Mission:** Take the model from prototype to production and keep it running
- **Key Responsibilities:**
  - **Deployment:** Expose the model via APIs or batch scoring pipelines
  - **CI/CD:** Automate retraining, testing, version control, and deployment
  - **Monitoring:** Track model performance, data drift, latency, and bias
  - **Scaling:** Ensure the model can handle production traffic
  - **Feedback integration:** Incorporate new data and trigger retraining when needed
- **Deliverables:** Production churn prediction service, monitoring dashboards, automated retraining workflows

2.5 5. Data Governance Officer

- **Primary Mission:** Ensure compliance and security across the entire pipeline
- **Key Responsibilities:**
  - Data security and access controls
  - PII (Personally Identifiable Information) anonymization
  - Regulatory compliance (GDPR, HIPAA, etc.)
  - Audit trails and documentation

Role Overlap in the ML Pipeline

Different stages of the pipeline require collaboration:

Stage	Primary Owners
Data Prep	Data Engineer
EDA & Feature Engineering	Data Scientist
Model Training & Validation	Data Scientist + ML Engineer
Deployment	ML Engineer
Monitoring	ML Engineer
Business Validation	Business Analyst + Data Scientist
Governance	Data Governance (all stages)

3 The Three Core Pipelines of MLOps

MLOps consists of three interconnected pipelines that form a continuous cycle.

3.1 Pipeline 1: Model Training

Training Pipeline Goal

Transform raw data into a trained model artifact that generalizes well (doesn't overfit).

Key Steps:

1. **Data Preparation:** Split data into Train / Validation / Test sets
2. **Feature Engineering:** Create meaningful features from raw variables
3. **Model Selection:** Choose appropriate algorithms
4. **Hyperparameter Tuning:** Optimize model settings to improve accuracy and reduce overfitting
5. **Evaluation:** Measure performance (Accuracy, Precision, Recall, F1, AUC-ROC)
6. **Artifact Storage:** Save the trained model and log metadata to MLflow

3.2 Pipeline 2: Model Inferencing (Serving)

Inferencing Pipeline Goal

Use the trained model to generate predictions on new data and support business decisions.

Two Main Approaches:

Table 1: *Batch vs. Real-time Inferencing*

Aspect	Batch Inferencing	Real-time Inferencing
How it works	Process large datasets periodically	Return predictions instantly per request
Latency	Minutes to hours	Milliseconds to seconds
Example	"Every night, score all customers for churn risk"	"When customer hovers over cancel button, show discount popup"
Infrastructure	Scheduled jobs (Airflow, Databricks)	REST APIs, streaming (Kafka)

Key Considerations: Latency, throughput, scalability, security

3.3 Pipeline 3: Model Monitoring

Monitoring Pipeline Goal

Continuously track model performance in production and detect issues before they cause business harm.

What to Monitor:

Table 2: *Types of Drift to Monitor*

Drift Type	What Changed?	How to Detect
Data Drift	The statistical distribution of **input features** changed *Example: New marketing campaign brings in customers from a different demographic*	Compare input data statistics (mean, variance, distribution) against baseline
Concept Drift	The **relationship** between inputs and target changed *Example: Customers now churn due to poor service, not price*	Most difficult. Monitor business KPIs even when data drift is absent
Model Decay	Overall model performance degradation	Track accuracy, F1, precision, recall against ground truth when available

Actions on Detection:

- Send alerts to data scientists
- Trigger automatic retraining pipeline
- Roll back to a previous model version

Example: How is Drift Detected Automatically?

Data Drift:

1. During training, save statistical profiles (mean, std, distribution) of each feature as a **baseline**
2. In production, periodically compute the same statistics on incoming data
3. Use statistical tests (KS test, Jensen-Shannon divergence) to compare
4. If difference exceeds threshold → alert!

Model Drift:

1. When ground truth becomes available (e.g., did the customer actually churn?), compare predictions to reality
2. Track metrics over time (e.g., F1 score by week)
3. If metrics drop below threshold → trigger retraining

Concept Drift:

- Hardest to detect automatically
- Often manifests as: "Data looks the same, model metrics look okay, but business KPIs are declining"
- Requires human investigation and domain expertise

4 Deployment Strategies: Models vs. Code

When promoting a model from Dev → Staging → Production, there are two main strategies.

4.1 Strategy 1: Deploy Models

- **How it works:** Train the model in Dev. Take the resulting **model artifact (file)** and copy it to Staging, then to Production.
- **Flow:** [Train in Dev] → [Model file] → [Copy to Staging] → [Copy to Prod]
- **Pros:** Faster (no retraining), less compute cost
- **Cons:** Model was trained on Dev data, which may not represent Production data

4.2 Strategy 2: Deploy Code

- **How it works:** Develop the **training code** in Dev. Copy the code to Staging and Production. **Retrain the model in each environment using that environment's data.**
- **Flow:** [Develop code in Dev] → [Copy code to Staging] → [Retrain with Staging data] → [Copy code to Prod] → [Retrain with Prod data]

4.3 Why Deploy Code is Often Preferred

Table 3: *Benefits of Deploy Code Strategy*

Benefit	Explanation
Data Access Control	**The biggest advantage.** Sensitive production data never needs to leave the production environment. Each environment trains only on its own data.
Reproducibility	Engineering controls the training environment in each stage, making reproduction easier
Real Data Issues	Production data has skews and volumes that Dev data doesn't. Training in Prod catches issues that only appear at scale.
Modular Code	Forces data scientists to write clean, modular, testable code instead of handing off notebooks

Downsides of Deploy Code

- **More compute cost:** Model is retrained in every environment
- **Infrastructure requirements:** Requires CI/CD setup with unit/integration testing
- **Skill requirements:** Data scientists must write production-quality code, not just notebooks

5 Data-Centric AI: The Key Insight

This is one of the most important concepts in modern MLOps.

5.1 Two Ways to Improve AI

1. **Model-Centric AI:** "How can I improve the model/algorithm/code to get better performance?"
2. **Data-Centric AI:** "How can I systematically improve the **data** to get better performance?"

Definition: Data-Centric AI

Data-Centric AI is the approach of improving AI system performance by focusing on the quality, consistency, and coverage of the **data** rather than tweaking the model architecture or hyperparameters.

5.2 Big Data vs. Good Data

Having more data (Big Data) doesn't guarantee better models. **Bad data in, bad predictions out.**

What is "Good Data"?

- **Unambiguous labels:** Ground truth is consistent and correct
- **Good coverage:** Includes important edge cases and scenarios (e.g., if predicting for all states, don't train only on Massachusetts)
- **Timely feedback:** Fresh data from production is quickly incorporated
- **Appropriately sized:** Not too small (underfitting), not unnecessarily large

5.3 Why Data-Centric AI Wins

Example: The Cooking Analogy

Data is the ingredients. The model is the recipe.
Even the best chef (model) can't make a great dish with rotten ingredients (bad data).
Model-Centric: "Let's adjust the cooking time from 15 minutes to 16 minutes." → Minor improvement
Data-Centric: "Let's source fresher, higher-quality ingredients." → Major improvement

Real-World Evidence:

In industrial case studies (steel defect detection, solar panel inspection), improving data quality led to **10-16% improvement** in model performance, while tweaking the model algorithm led to only **0-4% improvement**.

The takeaway: MLOps should prioritize processes that ensure high-quality, consistent data flowing into your models.

Table 4: *Model-Centric vs. Data-Centric Results (Example)*

Use Case	Model-Centric Improvement	Data-Centric Improvement
Steel Defect Detection	+0%	+16.9%
Solar Panel Inspection	+0.4%	+3.1%
Surface Inspection	+0.3%	+4.0%

6 Hyperparameter Tuning and Model Selection

A critical part of the Training Pipeline is finding the best model configuration.

6.1 What are Hyperparameters?

Definition: Hyperparameters

Hyperparameters are model settings that the developer chooses **before** training, as opposed to **parameters** which the model learns during training.

Examples:

- Learning rate
- Number of trees in a forest
- Maximum tree depth
- Regularization strength

6.2 Tuning Methods

Table 5: *Hyperparameter Tuning Methods*

Method	How It Works	Pros / Cons
Grid Search	Try **all combinations** of specified parameter values	Simple but expensive (exponential growth: $O(n^k)$)
Random Search	Randomly sample from parameter space	Often more efficient than grid search for the same compute budget
Bayesian Search	Use past results to intelligently choose next combination	Converges faster to optimal; used by Optuna, Hyperopt
Genetic Algorithms	Combine "good" and "bad" parameter sets like breeding	Good for very large search spaces (deep learning)

Tools: `Optuna`, `Hyperopt` (covered in Lecture 08)

6.3 K-Fold Cross Validation

Definition: K-Fold Cross Validation

A technique to evaluate model performance without "cheating" by using the test set.

Process (K=3):

1. Split training data into 3 "folds": [Fold 1], [Fold 2], [Fold 3]
2. **Round 1:** Train on [1, 2], validate on [3]
3. **Round 2:** Train on [1, 3], validate on [2]
4. **Round 3:** Train on [2, 3], validate on [1]
5. Final score = **average** of 3 validation scores

Benefit: The test set (holdout) remains untouched until final evaluation, preventing overfitting to the validation set.

7 MLOps Maturity Model

Organizations don't achieve full MLOps maturity overnight. There's a progression:

Table 6: *MLOps Maturity Levels*

Level	Characteristics	Example
Level 1: Beginner	Use familiar tools; manual processes; plan for more complex workloads	Data scientist trains model manually, emails pickle file to engineering
Level 2: Intermediate	Scale data and workloads; focus on **automation and reproducibility**; unify data teams	Automated retraining on schedule; MLflow tracking in place
Level 3: Advanced	**Faster production cycles** (deploy multiple times daily); end-to-end automation; **resilience testing** (Chaos Monkey); robust rollback	Netflix-style: intentionally break production to test recovery time

Example: Netflix's Chaos Monkey

Netflix intentionally introduces failures into their production systems to test how quickly the system can recover. This philosophy extends to MLOps: if a model deployment fails, how quickly can you roll back? How fast can you retrain?

The goal is to not fear failure, but to build systems that recover gracefully.

8 Champion-Challenger Pattern

When deploying a new model, you don't immediately replace the production model. Instead, you use a **Champion-Challenger** approach.

Definition: Champion-Challenger

- **Champion:** The current production model (proven performance)
- **Challenger:** The new candidate model (potentially better, but unproven)

Process:

1. Train a new model (Challenger)
2. Compare Challenger's metrics (F1, accuracy) against Champion's
3. If Challenger is better → promote Challenger to Champion
4. Archive the old Champion

In MLflow, this is implemented using **Model Aliases**:

```
# Set new model as challenger
client.set_registered_model_alias("churn_model", "challenger", version)

# After validation, promote to champion
client.set_registered_model_alias("churn_model", "champion", version)

# Load model by alias for inference
model = mlflow.pyfunc.load_model("models:/churn_model@champion")
```

9 End-to-End MLOps Lab Walkthrough

The lab demonstrates a complete MLOps pipeline for customer churn prediction using Databricks, MLflow, and Unity Catalog.

9.1 Lab Architecture

- **Data Layer:** Delta Lake tables in Unity Catalog
- **ML Lifecycle:** MLflow for tracking, models, and registry
- **Governance:** Unity Catalog for data and model governance
- **Algorithm:** LightGBM classifier

9.2 Step 1: Data Preparation & Feature Engineering

```
# Read raw customer data
telco_df = spark.table("mlops_churn_bronze_customer")

# Create new feature: number of optional services
def add_optional_services(df):
    service_cols = ['online_security', 'online_backup',
                    'device_protection', 'tech_support']
    df['num_optional_services'] = (
        df[service_cols].apply(lambda x: (x == 'Yes').sum(), axis=1)
    )
    return df

# Save to Delta table for training
cleaned_df.write.format("delta").saveAsTable("mlops_churn_training")
```

Listing 1: Feature Engineering

9.3 Step 2: Model Training with Data Lineage

The key insight: **Log which data was used to train the model.**

```
import mlflow

# Load and log the SOURCE DATA
source_dataset = mlflow.data.load_delta(
    table_name="catalog.schema.mlops_churn_training",
    version=latest_version  # Track exact version!
)
mlflow.log_input(source_dataset, context="training")

# Now train model
with mlflow.start_run():
    mlflow.autolog()  # Automatically log params, metrics, model

```

```
    model = LGBMClassifier(**params)
    model.fit(X_train, y_train)

    # Model + data lineage now tracked together!
```

Listing 2: Logging Data Lineage (Critical!)

Important: Why Log Data Lineage?

When something goes wrong in production, you need to answer: "Which data and which code produced this model?"
Without data lineage, you can't reproduce the model. MLflow's `log_input()` stores **metadata** (not a copy) linking the model to the exact data version used.

9.4 Step 3: Register to Model Registry

```
# Search for best run by F1 score
best_run = mlflow.search_runs(
    experiment_ids=[exp_id],
    filter_string="status = 'FINISHED'",
    order_by=["metrics.f1_score DESC"],
    max_results=1
).iloc[0]

# Register to Unity Catalog
model_uri = f"runs:/{best_run.run_id}/model"
mlflow.register_model(model_uri, "catalog.schema.mlops_churn")

# Set alias
client.set_registered_model_alias("mlops_churn", "challenger", 1)
```

Listing 3: Finding Best Model and Registering

9.5 Step 4: Champion-Challenger Validation

```
# Load challenger model
challenger = mlflow.pyfunc.load_model("models:/mlops_churn@challenger")
challenger_f1 = get_metric(challenger, "f1_score")

# Try to load champion (may not exist yet)
try:
    champion = mlflow.pyfunc.load_model("models:/mlops_churn@champion")
    champion_f1 = get_metric(champion, "f1_score")

    if challenger_f1 > champion_f1:
        print("Challenger wins! Promoting...")
        client.set_registered_model_alias("mlops_churn", "champion", version)
except:
    print("No champion found. Challenger becomes champion.")
```

```
    client.set_registered_model_alias("mlops_churn", "champion", version)
```

Listing 4: Promoting Challenger to Champion

9.6 Step 5: Batch Inference

```
# Load champion model
champion_model = mlflow.pyfunc.load_model("models:/mlops_churn@champion")

# Create Spark UDF for distributed scoring
predict_udf = mlflow.pyfunc.spark_udf(spark, "models:/mlops_churn@champion")

# Score new customer data
inference_df = spark.table("mlops_churn_inference")
scored_df = inference_df.withColumn(
    "prediction",
    predict_udf(*feature_columns)
)

# Result: DataFrame with 'prediction' column (1=churn, 0=stay)
scored_df.write.format("delta").saveAsTable("churn_predictions")
```

Listing 5: Scoring New Data

10 Web Hooks and Automation

MLOps pipelines can be automated using **webhooks**—HTTP callbacks that trigger actions when events occur.

Example: Webhook Use Cases

- **Model registered:** Send Slack notification to ML team
- **Model promoted to staging:** Trigger automated testing pipeline
- **Drift detected:** Trigger retraining job
- **Model promoted to production:** Update documentation

```python
# When a model is registered, trigger testing
@webhook.on("model_registered")
def run_tests(event):
    model_name = event["model_name"]
    version = event["version"]

    # Trigger Databricks job
    databricks_api.run_job(
        job_id="test_model_job",
        parameters={"model": model_name, "version": version}
    )

    # Send Slack notification
    slack.send(f"New model {model_name} v{version} registered!")
```

Listing 6: Webhook Example (Conceptual)

11 Summary: One-Page Quick Reference

MLOps = ML + Dev + Ops

The practice of reliably and repeatedly developing, deploying, and monitoring ML models in production.

The 5 Personas

- **Business Analyst**: Define the problem, set KPIs
- **Data Engineer**: Build data pipelines, ensure quality
- **Data Scientist**: Build and validate models
- **ML Engineer**: Deploy, monitor, scale
- **Governance Officer**: Security and compliance

The 3 Pipelines

- **Training**: Data → Model artifact
- **Inferencing**: Model + New data → Predictions
- **Monitoring**: Track drift, trigger retraining

Data-Centric AI

Improving **data quality** yields more performance gain than tweaking model algorithms.
Good Data = Unambiguous labels + Good coverage + Timely feedback + Right size

Deploy Strategy

Deploy Code (preferred): Copy training code to each environment, retrain with that environment's data

- Better data access control (Prod data stays in Prod)
- Catches production data issues
- Forces modular code

Champion-Challenger Pattern

Don't blindly replace production models. Compare:

- Champion = current production model
- Challenger = new candidate model
- If Challenger F1 > Champion F1 → Promote Challenger

Key MLflow Commands

```
mlflow.log_input(dataset, context="training")  # Data lineage
mlflow.autolog()                                # Auto-log everything
mlflow.register_model(uri, "model_name")        # Register to registry
client.set_registered_model_alias(name, alias, version)  # Set alias
mlflow.pyfunc.load_model("models:/name@alias")  # Load by alias
mlflow.pyfunc.spark_udf(spark, uri)             # Batch scoring
```