

Business 4720 - Class 23

Managing Machine Learning Operations (MLOps)

Joerg Evermann

Faculty of Business Administration
Memorial University of Newfoundland
jevermann@mun.ca



Unless otherwise indicated, the copyright in this material is owned by Joerg Evermann. This material is licensed to you under the [Creative Commons by-attribution non-commercial license \(CC BY-NC 4.0\)](#)



This Class

What You Will Learn:

- ▶ MLOps Principles
- ▶ MLOps Challenges
- ▶ MLOps Lifecycle
- ▶ MLOps Participants
- ▶ MLOps Governance

Based On

Treveil, M. and the Dataiku Team (2020) *Introducing MLOps*, O'Reilly Media, Sebastopol, CA (T)

Gift, N. and Deza, Al. (2021) *Practical MLOps*, O'Reilly Media, Sebastopol, CA (GD)

Resources

Implementations are available on the following GitHub repo:

<https://github.com/jevermann/busi4720-mlops>

The project can be cloned from this URL:

<https://github.com/jevermann/busi4720-mlops.git>



MLOps Purpose

- ▶ Improve Operational Efficiency
 - ▶ Formalized and automated processes
 - ▶ Reliable and repeatable processes
 - ▶ Manageable, adaptable, and understandable processes
- ▶ Mitigate Risk
 - ▶ Availability of service
 - ▶ Model quality and model impacts
 - ▶ Prediction fairness
 - ▶ Skill loss
- ▶ Establish accountability, auditability, and traceability



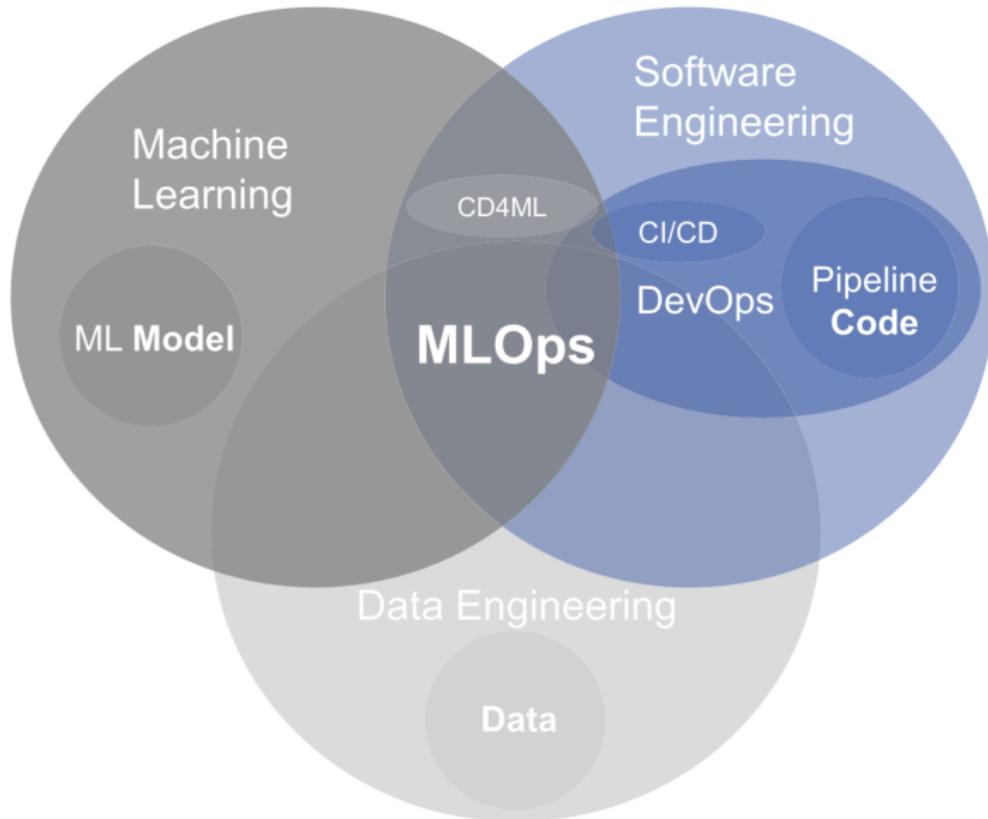
ML Challenges

- ▶ Increasing number of machine learning models and applications
- ▶ Data is constantly changing
- ▶ Business needs can change rapidly
- ▶ Mixed teams of business professionals, data scientists, software engineers and IT staff
- ▶ Data scientists have little expertise in software engineering

MLOps Principles

- 1 Reliability & Reproducability
- 2 Robust automation
- 3 Management and versioning of data and models
- 4 Continuous model (re-) development and continuous model delivery to production
- 5 Continuous monitoring in production

MLOps – Relationship to other disciplines

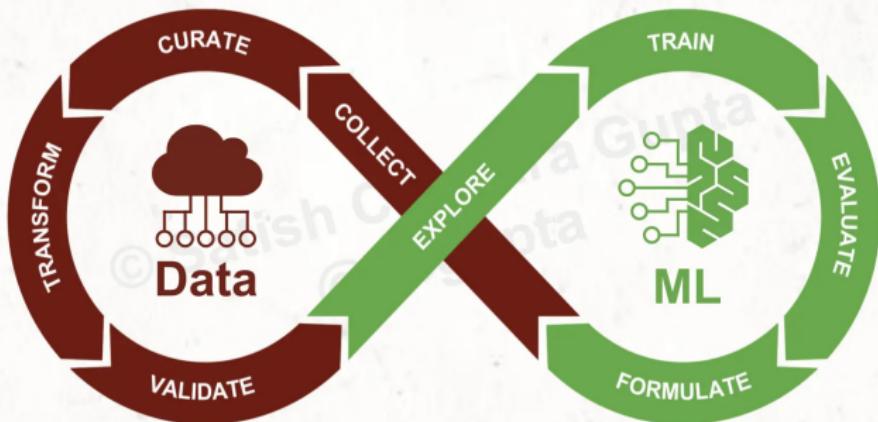


Source:
Kreuzberger, D.,
Kühl, N., &
Hirschl, S.
(2023). Machine
learning
operations
(mlops):
Overview,
definition, and
architecture.
IEEE access, 11,
31866-31879.

Model Development Lifecycle

Model Development

ml4devs.com/mlops-lifecycle 



© Satish Chandra Gupta



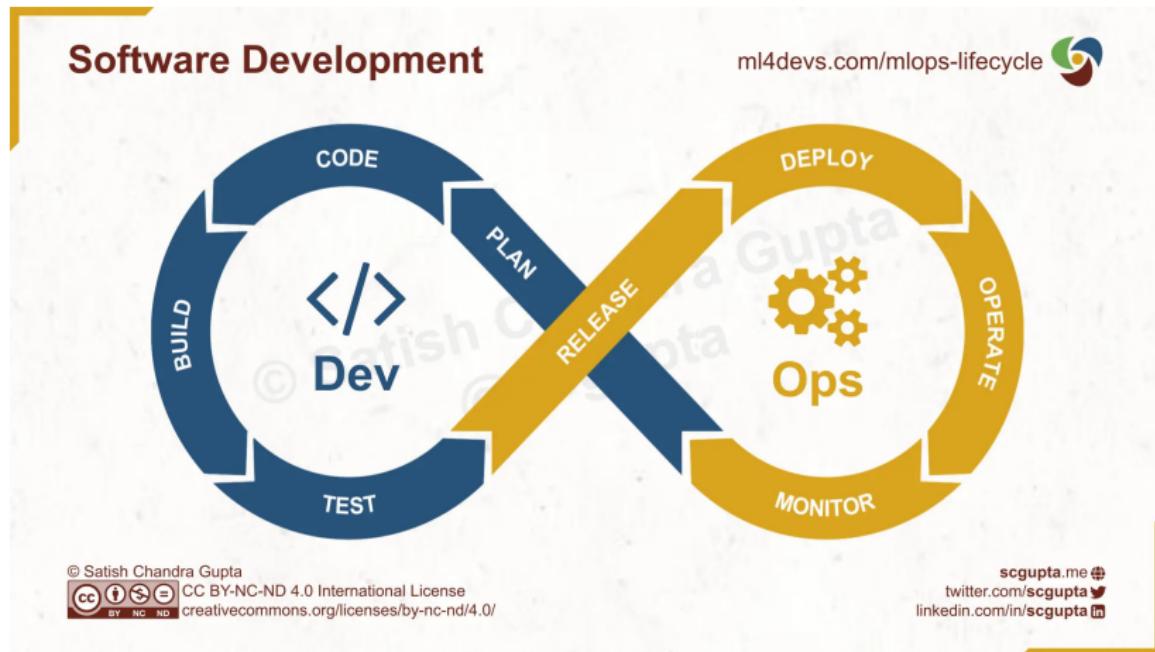
CC BY-NC-ND 4.0 International License
creativecommons.org/licenses/by-nc-nd/4.0/

scgupta.me 

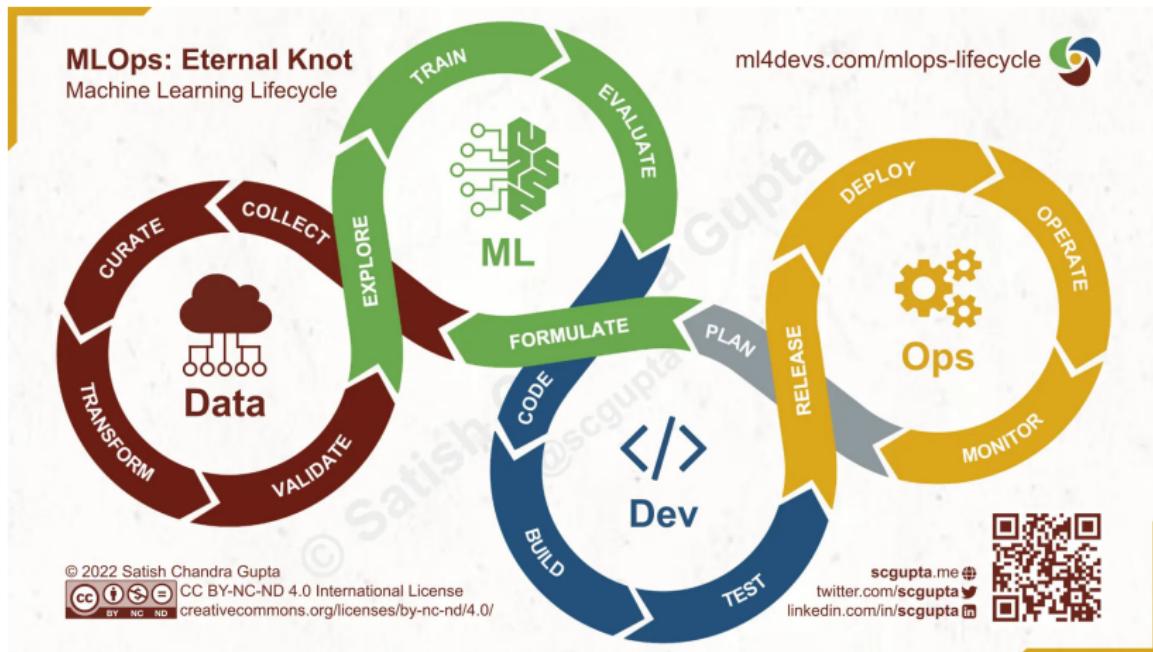
twitter.com/scgupta 

linkedin.com/in/scgupta 

Software Development Lifecycle



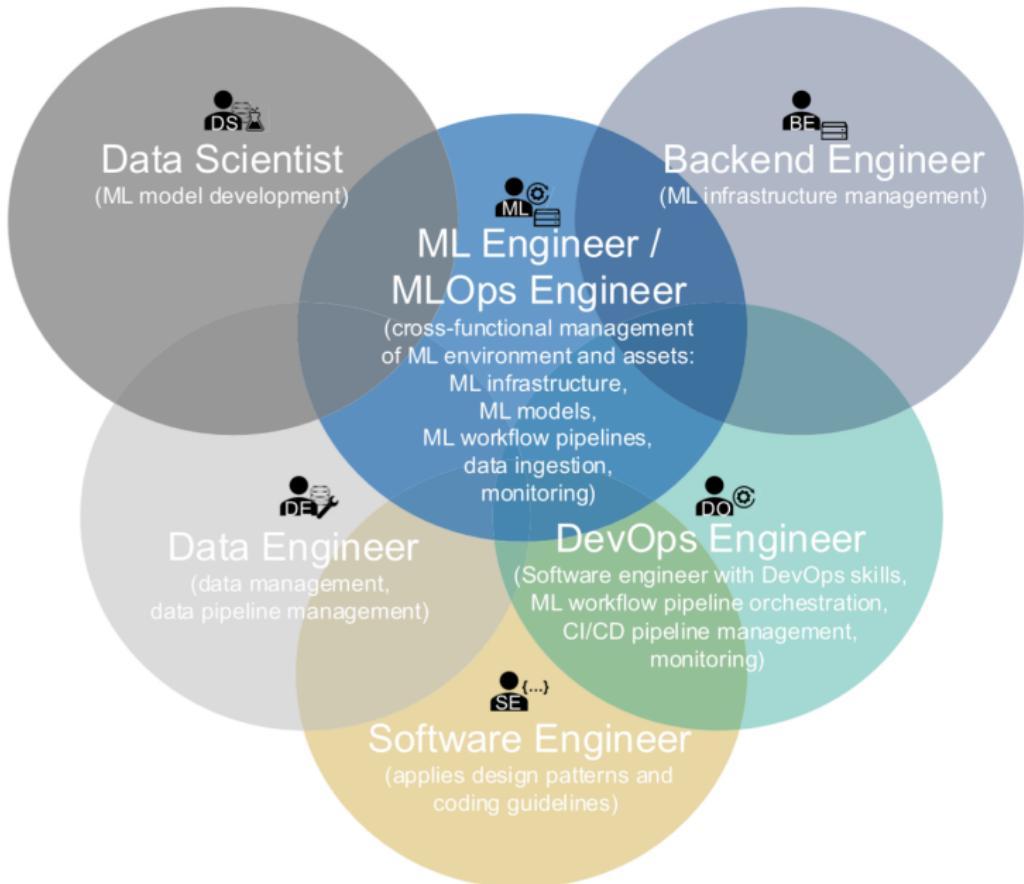
MLOps Lifecycle



MLOps Participants

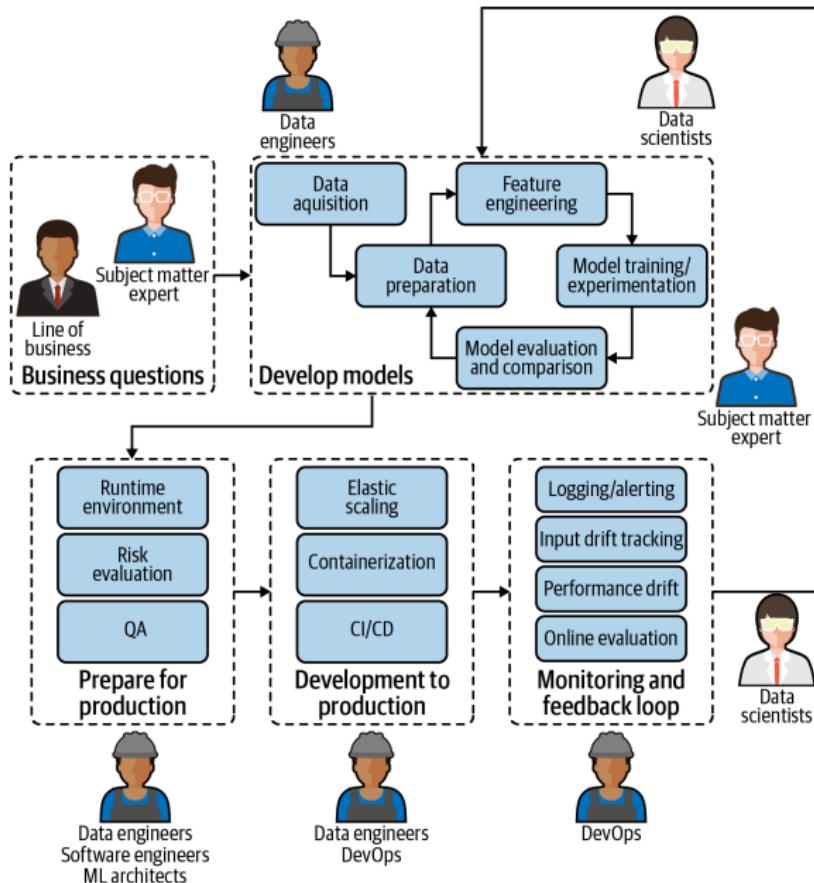
- ▶ Subject matter experts
 - ▶ Provide business questions, goals and KPIs for models
 - ▶ Evaluate model performance against business needs
- ▶ Data scientists
 - ▶ Develop and evaluate models
 - ▶ Deliver operationalizable models
- ▶ Data engineers
 - ▶ Optimize retrieval and use of data
- ▶ Software engineers
 - ▶ Integrate models into applications
- ▶ DevOps engineers
 - ▶ Build systems and test for security, performance, availability
 - ▶ CI/CD
- ▶ Model risk managers and model auditors
 - ▶ Minimize risk and ensure compliance
- ▶ ML engineer/ML architects
 - ▶ Ensure scalable and flexible environment

MLOps Participants – Overlapping Roles



Source:
Kreuzberger, D.,
Kühl, N., &
Hirschl, S.
(2023). Machine
learning
operations
(mlops):
Overview,
definition, and
architecture.
IEEE access, 11,
31866-31879.

MLOps Participants – Roles in MLOps lifecycle



MLOps – Requirements

- ▶ Subject matter experts:
 - ▶ Understandability of deployed models in business terms
 - ▶ Feedback mechanism for models
- ▶ Data scientists and data engineers:
 - ▶ Automated model packaging and delivery
 - ▶ Ability to automatically test model quality
 - ▶ Visibility into model performance (dev, stage, production)
 - ▶ Visibility into data pipelines for each model
- ▶ Software engineers:
 - ▶ Versioning and automatic testing
- ▶ DevOps engineers:
 - ▶ Integration with wider DevOps strategies
 - ▶ Seamless deployment pipeline

MLOps Requirements [cont'd]

- ▶ Model risk managers and model auditors:
 - ▶ Automated reporting on all models (past and present), including data provenance
- ▶ ML engineer/architect:
 - ▶ Ability to assess and adjust infrastructure capacities

MLOps – Tooling

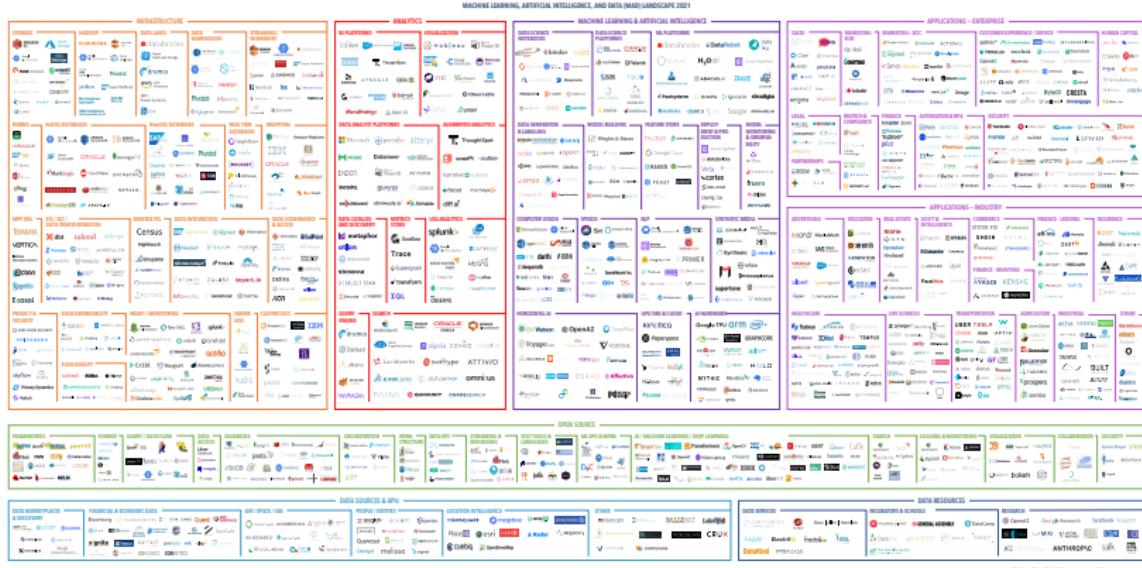
- ▶ Source Code Repository
 - ▶ Training, inference and application source code
 - ▶ Versioning
 - ▶ Examples: GitHub, GitLab
- ▶ CI/CD
 - ▶ Build, test, deploy
 - ▶ Examples: Jenkins, GitHub actions
- ▶ Workflow Orchestration
 - ▶ Defines execution and artifact usage
 - ▶ Data extraction, training, inference, deployment
 - ▶ Examples: Apache Airflow, AWS SageMaker Pipelines, Azure Pipelines
- ▶ Feature Store
 - ▶ Central storage of feature data
 - ▶ Examples: Google Feast, AWS Feature Store, Tecton.ai
- ▶ Model Training Infrastructure
 - ▶ CPU and GPU for training

MLOps – Tooling [cont'd]

- ▶ Model Registry
 - ▶ Store trained model and metadata
 - ▶ Versioning
 - ▶ Examples: MLflow, AWS SageMaker Model Registry, Azure ML Model Registry
- ▶ ML Metadata Stores
 - ▶ ML Pipeline execution, model training, model lineage, etc.
 - ▶ Examples: MLFlow
- ▶ Model Serving
 - ▶ Online inference, real-time predictions
 - ▶ Examples: Flask, TensorFlow Serving, AWS SageMaker Endpoints
- ▶ Monitoring
 - ▶ Performance monitoring
 - ▶ Input drift detection
 - ▶ Examples: TensorBoard, AWS SageMaker model monitor

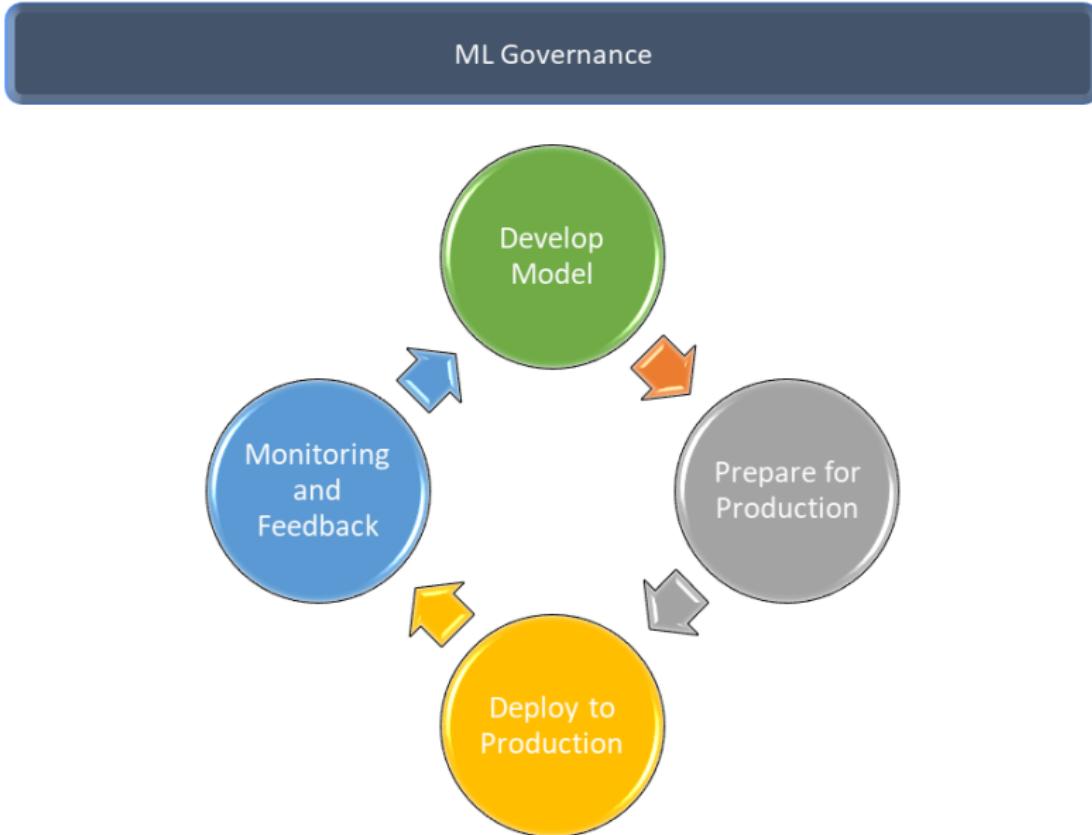
MLOps Tooling [cont'd]

Commercial Offerings

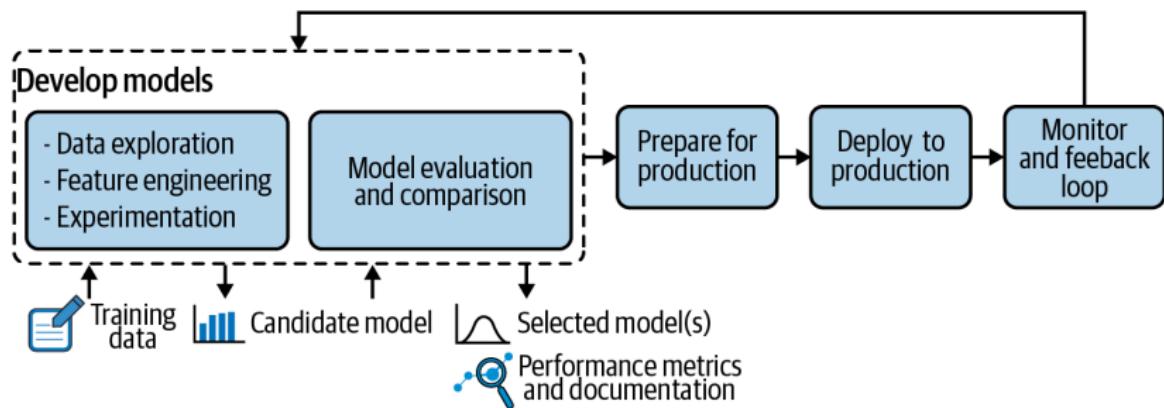


Source: Turck, Matt. *Red Hot – The 2021 Machine Learning, AI and Data (MAD) Landscape*. September 28, 2021. <https://mattturck.com/data2021/> (last accessed July 22, 2024)

MLOps Lifecycle



Develop Models



Source: Treveil et al. (2020), Figure 4-1

Develop Models [cont'd]

Data

- ▶ What data are available? What is the quality of that data?
- ▶ Can the data legally be used for this purpose? What are the terms of use of the data?
- ▶ How can the data be accessed?
- ▶ What features can be created by combining data sets?
- ▶ Must the data be redacted or anonymized?
- ▶ Are there features that cannot be used legally (age, gender, race, etc.)?
- ▶ Is the data representative of minority classes/populations?

Automation and Tools

- ▶ ETL Pipelines (extraction from source)
- ▶ Data Lakes (centralized storage)
- ▶ Feature Stores (engineered features)

Develop Models [cont'd]

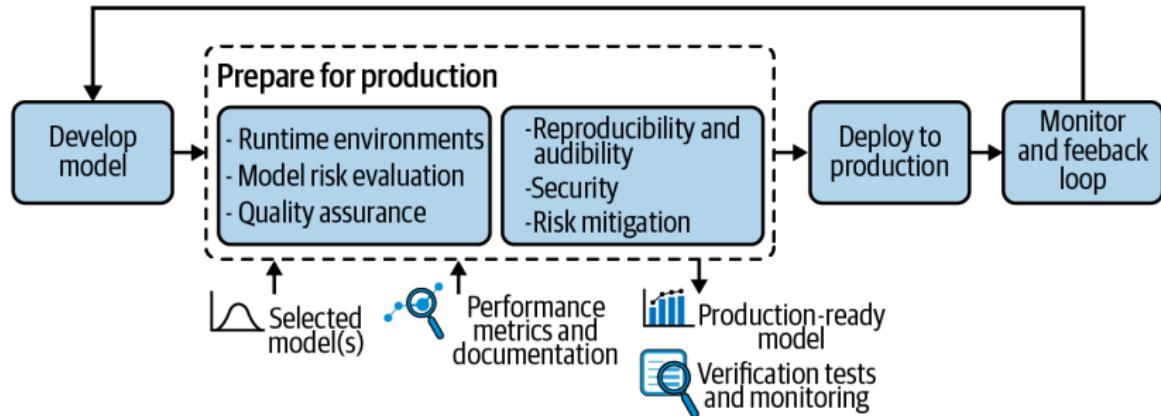
Model Training

- ▶ What are appropriate evaluation metrics?
- ▶ Is the model performance acceptable for sub-populations?
- ▶ Does the model need to be interpretable or explainable?
- ▶ Are the model outcomes fair?

Automation and Tools

- ▶ Model registries and repositories (weights, biases, hyperparameters, random seeds, results, etc.)
- ▶ Container makefiles and container registries (fixing software versions and environment)
- ▶ Feature Stores (training and test data versioning and update processes)

Prepare for Production



Source: Treveil et al. (2020), Figure 5-1

Prepare for Production [cont'd]

Technical Questions

- ▶ What is the runtime environment? (e.g. Flask containers, Tensorflow Serving, Kubernetes Clusters, Edge Devices, JavaScript)
- ▶ Does the model need to be adapted? (e.g. transformation, quantization, pruning)
- ▶ How are data features accessed or provided?

Risk Assessment Questions

- ▶ What if the model acts in the worst possible way?
- ▶ What if a client extracts training data or model details?
- ▶ What are financial, business, legal, and reputational risks?

Sources of Risk

- ▶ Errors in model design or training (incl. data prep)
- ▶ Errors in runtime environment
- ▶ Data quality problems
- ▶ Differences btw training & production data ("input drift")
- ▶ Abuse of model or misuse of outputs
- ▶ Adversarial attacks
- ▶ Legal risk from training data use or model results
- ▶ Reputational risk

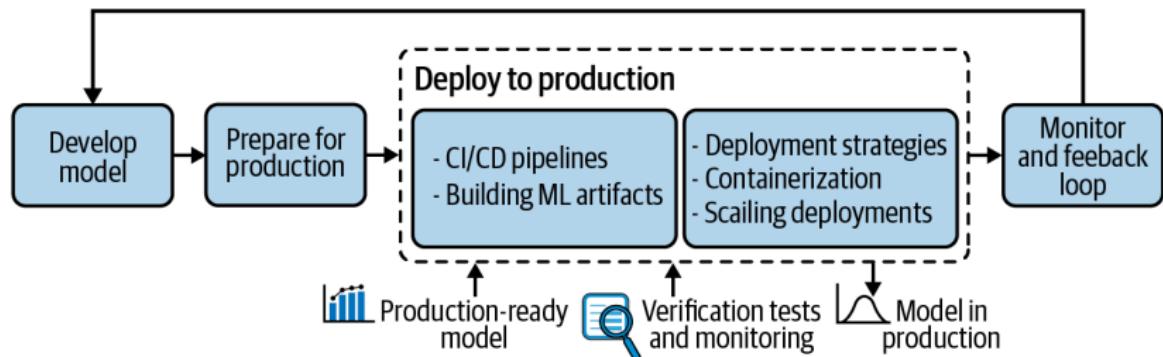
Risk Mitigation

- ▶ Shadow testing
- ▶ Progressive rollouts
- ▶ Continuous logging and monitoring
- ▶ Input and output checks
- ▶ Failover to simpler model
- ▶ Periodic retraining

Automation and Tools

- ▶ Continuous integration and automated testing
- ▶ Model registries to document artifact
 - ▶ Input data sources and provenance
 - ▶ Model assumptions
 - ▶ Software dependencies
 - ▶ Test results (incl. explanations and bias evaluation)
 - ▶ Training and test logs

Deploy to Production



Source: Treveil et al. (2020), Figure 6-1

Deploy to Production [cont'd]

Automated CI/CD Pipeline

1 Build model

- 1.1 Build model artifacts (model code, configuration, data, trained model, environment, documentation, test code and test data)
- 1.2 Archive and register model
- 1.3 Basic checks
- 1.4 Evaluate bias and interpretability

2 Deploy to test environment

- 2.1 Evaluate predictive performance
- 2.2 Evaluate computational performance

3 Deploy to production environment

- 3.1 Limited deployment (parallel or "canary")
- 3.2 Full deployment

Adapted from Trevail et al. (2020) (pg. 74, 75)

Deploy to Production

Scalability and Reliability

- ▶ Deployment targets (models to servers)
- ▶ Automatic Workload balancing
- ▶ Automatic Failover (detection, reprovisioning)
- ▶ Model upgrades

Maintenance

- ▶ Continuous Resource monitoring
- ▶ Continuous Health checks
- ▶ Continuous ML metrics monitoring



Deploy to Production

Automation and Tools

- ▶ Source code repositories (e.g. GitHub)
- ▶ Continuous integration (e.g. Jenkins)
- ▶ Model registries (e.g. MLflow)
- ▶ Model serving (e.g. Flask, Tensorflow Serving)
- ▶ Log data storage and analysis

Deployment Options

- ▶ Microservice (e.g. Flask)
- ▶ Tensorflow TFX and Tensorflow Serving
- ▶ Tensorflow JS (for browser deployment)
- ▶ Tensorflow Lite (for edge devices and mobile apps)

Flask Example – Step 1: Create a Trained Model

Complete file is available on [GitHub](#).

Define the model and train it:

```
import keras.utils
import pandas as pd
import tensorflow as tf
import tensorflowjs as tfjs

keras.utils.set_random_seed(42)
boston_data = \
    pd.read_csv("https://evermann.ca/busi4720/boston.csv")

boston_features = boston_data[['rm', 'tax', 'age']]
boston_labels = boston_data['medv']

# Linear regression model
norm_boston_model=keras.models.Sequential([
    keras.layers.Input(shape=(3,), dtype=tf.float32),
    keras.layers.Dense(1, activation=None) ])
```

Flask Example [cont'd]

Fit the model and save it:

```
stop_callback = keras.callbacks.EarlyStopping()
norm_boston_model.compile(
    loss = tf.keras.losses.MeanSquaredError())
norm_boston_model.fit(
    boston_features, boston_labels,
    epochs=100, validation_split=0.33,
    callbacks=[stop_callback])

# Save model for use in Keras
norm_boston_model \
    .save('norm.boston.model.trained.save')
# Export model for use in TF Serving
norm_boston_model \
    .export('norm.boston.model.trained.export')
# Convert model for use in TFJS
tfjs.converters.save_keras_model(norm_boston_model, \
    'norm.boston.model.trained.tjfs')
```



Flask Example – Step 2: Serve the Model

Complete file is available on [GitHub](#).

Load the model for prediction:

```
import keras
import flask
from flask import request
import pandas as pd

# Load the trained model
norm_boston_model = keras.saving. \
    load_model('norm.boston.model.trained.save')

# A predict function for the model
def predict(inputs):
    return norm_boston_model. \
        predict_on_batch(inputs)[0][0]

app = flask.Flask(__name__)
```

Flask Example [cont'd]

Define the URL handler and run app:

```
@app.route("/predict_json", methods=["POST"])
def predict_json():
    reply = {}
    # TODO: Input checking goes here
    # TODO: Input logging goes here
    inputs = pd.DataFrame \
        .from_dict(request.json) \
        .transpose()
    prediction = predict(inputs)
    # TODO: Output checking goes here
    # TODO: Output logging goes here
    reply["prediction"] = str(prediction)
    reply["success"] = True
    return flask.jsonify(reply)

app.run()
```



Flask Example – Step 3: Access the Service

Complete file is available on [GitHub](#).

Access the prediction with JSON POST request:

```
#!/usr/bin/bash
curl -X POST \
  -H "Content-Type: application/json" \
  --data '[6, 250, 66.5]' \
  http://localhost:5000/predict_json
```



Flask Example – Web Forms

Complete file is available on [GitHub](#).

Access using a JSON POST request from a Web Form:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Boston Housing Data Prediction Service</title>
<script>
  async function predict() {
    // Get the values from the text inputs
    const rooms =
      parseFloat(document.getElementById('rooms').value);
    const tax =
      parseFloat(document.getElementById('tax').value);
    const age =
      parseFloat(document.getElementById('age').value);
    // Create a JSON payload
    const payload = JSON.stringify([rooms, tax, age]);
  }
</script>
</head>
<body>
  <h1>Boston Housing Data Prediction Service</h1>
  <form>
    <label>Rooms</label>
    <input type="text" id="rooms" value="5" />
    <label>Tax</label>
    <input type="text" id="tax" value="10000" />
    <label>Age</label>
    <input type="text" id="age" value="50" />
    <button type="button" onclick="predict();">Predict</button>
  </form>
</body>

```

Flask Example [cont'd]

```
// Make a POST request to the server
const response = await fetch('/predict_json', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: payload
});
// Parse the JSON response
const result = await response.json();
// Display the result
document.getElementById('output-div').textContent
  = result.prediction;
}
</script>
</head>
```



Flask Example [cont'd]

```
<body>
<h1>Boston Housing Data Inputs</h1>
<form onsubmit="event.preventDefault(); predict();">
  <p>
    <label for="rooms">Number of Rooms</label>
    <input name="rooms" id="rooms" required>
  </p>
  <p>
    <label for="tax">Tax Rate per $10,000</label>
    <input name="tax" id="tax" required>
  </p>
  <p>
    <label for="age">Prop bldg older than 1940</label>
    <input name="age" id="age" required>
  </p>
  <input type="submit" value="Submit">
</form>
<p>Prediction is: <div id="output-div">...</div></p>
</body>
</html>
```

Tensorflow JS Example

Complete file is available on [GitHub](#).

```
<!DOCTYPE html>
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/\
@tensorflow/tfjs@latest/dist/tf.min.js"></script>
<script>
async function predict() {
  // Load the model
  const model = await \
    tf.loadLayersModel('https://raw.githubusercontent.com/jevermann/busi4720-mlops/main/model.json');
  // Get the values from the text inputs
  const rooms =
    parseFloat(document.getElementById('rooms').value);
  const tax =
    parseFloat(document.getElementById('tax').value);
  const age =
    parseFloat(document.getElementById('age').value);
```

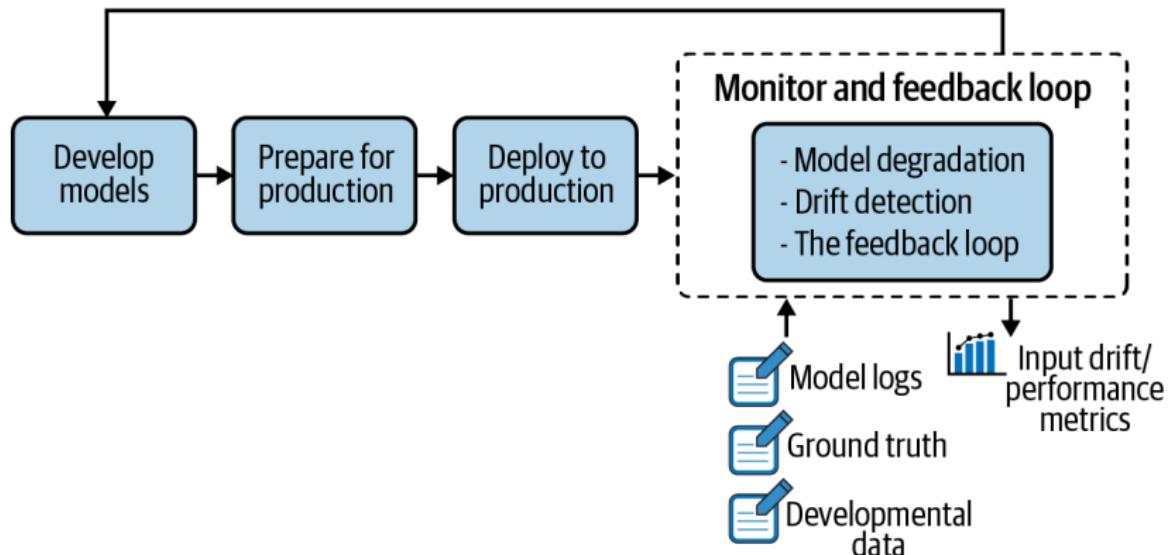
Tensorflow JS Examples [cont'd]

```
// Package the values into a Tensor
const inputs = tf.tensor2d([rooms, tax, age],[1, 3]);
// Get the prediction from the model
document.getElementById('output-div').innerText =
    model.predict(inputs).dataSync();
}
</script>
</head>
```

Remainder of the Web form as above.



Monitoring and Feedback



Source: Treveil et al. (2020), Figure 7-1

Monitoring and Feedback [cont'd]

Model Retraining Considerations

- ▶ Domain changes
- ▶ Training cost
- ▶ Model performance
- ▶ Ground truth availability

Ground Truth for Monitoring and Retraining

- ▶ Not always immediately or imminently available (e.g. loan repayment)
- ▶ Ground truth and prediction are decoupled (e.g. missing or mismatched identifiers)
- ▶ Ground truth not available for all classes (e.g. fraud detection)

Input Drift Causes

- ▶ Selection bias
- ▶ Non-stationary environment

Input Drift Detection

- ▶ Univariate statistical tests (e.g. χ^2 or Kolmogorov-Smirnov)^{a b}
- ▶ Domain classifier approach (train classifier to predict old or new sample domain)

^ahttps://en.wikipedia.org/wiki/Chi-squared_test

^bhttps://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test

Feedback Loop Requirements

- ▶ Logging (metadata, inputs, outputs, actions taken, explanations)
- ▶ Model store (features, preprocessing, train and test data, algorithm, eval metrics)
- ▶ Online evaluation (shadow testing or A/B testing)

Basic Python Logging

Complete file is available on [GitHub](#).

Set up the logger:

```
import logging.handlers

req_logger=logging.getLogger(model_name+'.requests')
req_logger.setLevel(logging.INFO)
req_logger.addHandler(
    logging.FileHandler(
        model_name+'.requests.log'))
# req_logger.addHandler(
#     logging.handlers.RotatingFileHandler(
#         model_name+'.requests.log',
#         maxBytes=1000000,
#         backupCount=5))
```



Basic Python Logging [cont'd]

Use the logger:

```
@app.route("/predict_json", methods=["POST"])
def predict_json():
    req_logger.info('%s TIME %s IP %s JSON %s',
                    model_name,
                    time.ctime(),
                    request.remote_addr,
                    request.json)
    ...
def predict_form():
    req_logger.info('%s TIME %s IP %s FORM %s',
                    model_name,
                    time.ctime(),
                    request.remote_addr,
                    request.form)
    ...
```

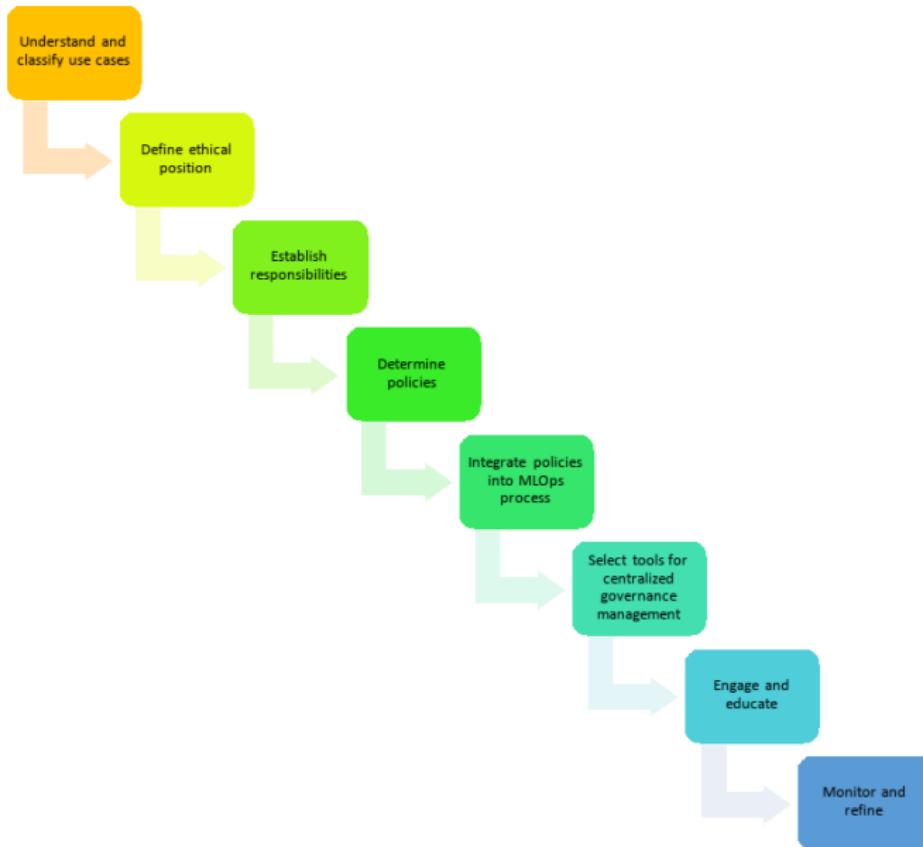


Hands-On Exercise

- 1 Download the complete file from [GitHub](#).
- 2 Define a second logger that writes to a different log file
 - ▶ You do not need to rotate this log file
 - ▶ The definition of the second logger is analogous to that of the request logger
- 3 Add logging to the `predict_json()` and the `predict_form()` functions to capture the time, the three inputs, and the prediction in the log.
 - ▶ Replace the `# TODO: Output logging goes here` comments with your code
 - ▶ To make the log easy to analyze, write the information in CSV format. Make sure you quote the fields that need quoting.



MLOps Governance



1. Understand ML Uses

- ▶ Who is the consumer of the model output?
- ▶ What regulations and legal constraints apply?
- ▶ What are the legal, financial, reputational risks of errors?
- ▶ What is need for explainability or interpretability?
- ▶ What are the availability requirements?
- ▶ What is the model lifetime and likely rate of model decay?

2. Define Ethical Position

- ▶ How important are aspects like equality, privacy, human rights, democracy, bias?
- ▶ How transparent should decision making be?
- ▶ What level of responsibility for errors will the business assume?
- ▶ What is the potential for deception, manipulation, exploitation?

3. Establish Responsibilities ("Who will do what?")

- ▶ Strategic, tactical, and operational
- ▶ Senior management sponsorship
- ▶ Integrate into existing governance mechanisms

| Tasks | Business stakeholders | Business analysis/citizen DS | Data scientists | Risk audit | Data ops | Production/exploitation | Resources admin/architect |
|---------------------------------|-----------------------|------------------------------|-----------------|------------|----------|-------------------------|---------------------------|
| Identification | A/R | C | | I | | | |
| Data preparation | C | A/R | C | | | | |
| Data modeling | C | A | R | | | | |
| Model acceptance | I | C | C | A/R | | | |
| Productionalization | | C | A/R | I | C | | |
| Capitalization | | | R | | R | | A |
| Integration to external systems | | | | | A/R | | |
| Global orchestration | | C | | | R | A | |
| User acceptance tests | A/R | R | C | | I | | |
| Deployments | | | | | R | A | I |
| Monitoring | I | C | | | | A/R | I |

Source: Treveil et al. (2020), Figure 8-4

4. Define Policies ("How will we do this?")

Establish rules for:

- ▶ Reproducibility and traceability
- ▶ Auditability and documentation
- ▶ Sign-off between stages
- ▶ Model verification
- ▶ Model explainability
- ▶ Model bias and bias testing
- ▶ Model deployment mechanisms
- ▶ Model monitoring
- ▶ Data quality and data compliance

Adapted from Treveil et al. (2020), Chapter 8

5. Integrate Policies into MLOps Process ("When will we do this?")

- ▶ Formalize and automate MLOps processes
- ▶ Define controls
- ▶ Define monitoring of controls

6. Implement Governance Tools

- ▶ Automate controls
- ▶ Logging of control violations
- ▶ Auditing of control effectiveness
- ▶ Policy and procedure maintenance

7. Engage and Educate

- ▶ Communicate
- ▶ Awareness
- ▶ Training
- ▶ Buy-in & commitment
- ▶ Culture

8. Monitor and Refine

- ▶ Evaluate risk exposure
- ▶ Evaluate policy adequacy
- ▶ Evaluate control effectiveness
- ▶ Evaluate MLOps process performance