End-to-End Machine Learning Systems

Lecture 3: ML Workflows

Prof. Tanu Malik, University of Missouri

Acknowledgements: These lecture materials are adapted from the original notes provided by Prof. Vijaya Reddi and Chip Huyen. Any deviations, errors, or supplementary content in these slides are solely my responsibility.

Why ML Systems?

Your team spent four months building a diabetic-retinopathy classifier.
Test accuracy: 95%; AUC: 0.98

This classifier is now deployed on a tablet for use in rural clinics.

Why ML Systems?

Your team spent four months building a diabetic-retinopathy classifier.
Test accuracy: 95%; AUC: 0.98

This classifier is now deployed on a tablet for use in rural clinics.
Deployment engineer: *The tablet only has 512 MB of available memory, and the model requires 4 GB.*
An 8× violation of memory constraints!

Why ML Systems?

Your team spent four months building a diabetic-retinopathy classifier.
Test accuracy: 95%; AUC: 0.98

This classifier is now deployed on a tablet for use in rural clinics.
Deployment engineer: *The tablet only has 512 MB of available memory, and the model requires 4 GB.*
An 8× violation of memory constraints!
The team is now faced with two questions:

- "Is the model good?"

Why ML Systems?

Your team spent four months building a diabetic-retinopathy classifier.
Test accuracy: 95%; AUC: 0.98

This classifier is now deployed on a tablet for use in rural clinics.
Deployment engineer: *The tablet only has 512 MB of available memory, and the model requires 4 GB.*
An 8× violation of memory constraints!
The team is now faced with two questions:

- "Is the model good?"
- "Is the ML system good?"

Why ML Systems?

Your team spent four months building a diabetic-retinopathy classifier.
Test accuracy: 95%; AUC: 0.98

This classifier is now deployed on a tablet for use in rural clinics.
Deployment engineer: *The tablet only has 512 MB of available memory, and the model requires 4 GB.*
An 8× violation of memory constraints!
The team is now faced with two questions:

- "Is the model good?"
- "Is the ML system good?"

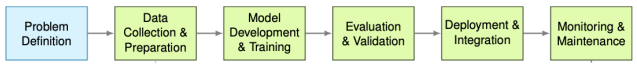
Q: At what point should the 512-MB constraint have affected the project?

The ML Pipeline

ML workflow

The ML pipeline comprises of five stages:

1. Data collection
2. Model development
3. Evaluation
4. Deployment
5. Monitoring

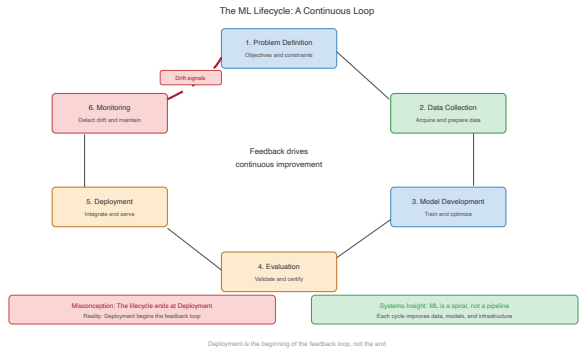


Pipeline = the stages

The ML Lifecycle

ML lifecycle

The ML lifecycle is the iterative loop through these five stages. Monitoring feeds back into data collection and model development so the system can improve over time.



Workflow = the stages; lifecycle = repeating the stages.

D.A.M

A constraint disovered during Deployment must be addressed in earlier stages.

However, changes made in earlier stages may introduce new constraints that must be addressed in later stages.
This is because the stages of the ML workflow are *interdependent*.

$$SystemPerformance = f(Data, Algorithm(Model, Evaluation), Machine(Deployment), Monitoring) \quad (1)$$

Constraint Propagation: Backwards and Forwards

How do we resolve the 512-MB constraint to reach a stable, feasible system state? **Constraint Propagation** means tracing a requirement backward to change upstream stages.

- **Step 1 (Hardware Constraint):** Tablet has 512 MB available memory.

Constraint Propagation: Backwards and Forwards

How do we resolve the 512-MB constraint to reach a stable, feasible system state? **Constraint Propagation** means tracing a requirement backward to change upstream stages.

- **Step 1 (Hardware Constraint):** Tablet has 512 MB available memory.
- **Step 2 (Deployment $\rightarrow$ Model):** Quantize model weights from FP32 to INT8 to reduce memory.

Constraint Propagation: Backwards and Forwards

How do we resolve the 512-MB constraint to reach a stable, feasible system state? **Constraint Propagation** means tracing a requirement backward to change upstream stages.

- **Step 1 (Hardware Constraint):** Tablet has 512 MB available memory.
- **Step 2 (Deployment $\rightarrow$ Model):** Quantize model weights from FP32 to INT8 to reduce memory.
- **Step 3 (Model $\rightarrow$ Evaluation):** Lower bit-width drops accuracy by 4%. Is the new model acceptable?

Constraint Propagation: Backwards and Forwards

How do we resolve the 512-MB constraint to reach a stable, feasible system state? **Constraint Propagation** means tracing a requirement backward to change upstream stages.

- **Step 1 (Hardware Constraint):** Tablet has 512 MB available memory.
- **Step 2 (Deployment $\rightarrow$ Model):** Quantize model weights from FP32 to INT8 to reduce memory.
- **Step 3 (Model $\rightarrow$ Evaluation):** Lower bit-width drops accuracy by 4%. Is the new model acceptable?
- **Step 4 (Evaluation $\rightarrow$ Upstream Design):** If not, revisit training, preprocessing, architecture, or data.

Constraint Propagation: Backwards and Forwards

How do we resolve the 512-MB constraint to reach a stable, feasible system state? **Constraint Propagation** means tracing a requirement backward to change upstream stages.

- **Step 1 (Hardware Constraint):** Tablet has 512 MB available memory.
- **Step 2 (Deployment $\rightarrow$ Model):** Quantize model weights from FP32 to INT8 to reduce memory.
- **Step 3 (Model $\rightarrow$ Evaluation):** Lower bit-width drops accuracy by 4%. Is the new model acceptable?
- **Step 4 (Evaluation $\rightarrow$ Upstream Design):** If not, revisit training, preprocessing, architecture, or data.

The Core Lesson: A constraint at one stage can force changes throughout the system.

Local optimization is not enough; **the system must satisfy all constraints simultaneously.**

Case Study: The Autonomous Food-Spoilage Detector

The Scenario

Your startup is building an automated system for supply-chain trucks: a smartphone camera + edge computer that detects if fresh produce is spoiling in real time.

System Constraints:

- **Machine:** $5 ARM-based embedded processor
- **Network:** Completely offline in rural transit zones
- **Input:** 4K camera stream at 30 frames/second
- **Latency:** Alert must be triggered within 200 ms

The Initial Model

A data-science team gives you a model with:

- 99% offline accuracy
- 2 GB memory footprint
- 400 ms inference time on the target device
- Designed and tested on a cloud GPU

Case Study: The Autonomous Food-Spoilage Detector

The Scenario

Your startup is building an automated system for supply-chain trucks: a smartphone camera + edge computer that detects if fresh produce is spoiling in real time.

System Constraints:

- **Machine:** $5 ARM-based embedded processor
- **Network:** Completely offline in rural transit zones
- **Input:** 4K camera stream at 30 frames/second
- **Latency:** Alert must be triggered within 200 ms

The Initial Model

A data-science team gives you a model with:

- 99% offline accuracy
- 2 GB memory footprint
- 400 ms inference time on the target device
- Designed and tested on a cloud GPU

Team Challenge:
Trace the 200-ms and memory constraints backward.
What must change in the **Data, Algorithm, and Evaluation**?

Case Study Solution: Backward Constraint Propagation

To make the system feasible, the deployment constraints propagate backward into the upstream pipeline.

Case Study Solution: Backward Constraint Propagation

To make the system feasible, the deployment constraints propagate backward into the upstream pipeline.

1. **Machine $\rightarrow$ Algorithm:**
   Replace the heavy cloud model with a compact edge architecture and use INT8 quantization to reduce memory and computation.

Case Study Solution: Backward Constraint Propagation

To make the system feasible, the deployment constraints propagate backward into the upstream pipeline.

1. **Machine $\rightarrow$ Algorithm:**
   Replace the heavy cloud model with a compact edge architecture and use INT8 quantization to reduce memory and computation.
2. **Algorithm $\rightarrow$ Data:**
   Reduce the computational burden of the input: lower resolution, reduce frame rate, or extract a region-of-interest before inference.

Case Study Solution: Backward Constraint Propagation

To make the system feasible, the deployment constraints propagate backward into the upstream pipeline.

1. **Machine $\rightarrow$ Algorithm:**
   Replace the heavy cloud model with a compact edge architecture and use INT8 quantization to reduce memory and computation.
2. **Algorithm $\rightarrow$ Data:**
   Reduce the computational burden of the input: lower resolution, reduce frame rate, or extract a region-of-interest before inference.
3. **Data $\rightarrow$ Algorithm:**
   Retrain the model on the new input representation. A model trained on 4K/30-FPS inputs cannot simply be assumed to work equally well on the reduced input.

Case Study Solution: Backward Constraint Propagation

To make the system feasible, the deployment constraints propagate backward into the upstream pipeline.

1. **Machine $\rightarrow$ Algorithm:**
   Replace the heavy cloud model with a compact edge architecture and use INT8 quantization to reduce memory and computation.
2. **Algorithm $\rightarrow$ Data:**
   Reduce the computational burden of the input: lower resolution, reduce frame rate, or extract a region-of-interest before inference.
3. **Data $\rightarrow$ Algorithm:**
   Retrain the model on the new input representation. A model trained on 4K/30-FPS inputs cannot simply be assumed to work equally well on the reduced input.
4. **Algorithm $\rightarrow$ Evaluation:**
   Evaluate the complete pipeline on the **target hardware**, including preprocessing and inference.

Case Study Solution: Backward Constraint Propagation

To make the system feasible, the deployment constraints propagate backward into the upstream pipeline.

1. **Machine $\rightarrow$ Algorithm:**
   Replace the heavy cloud model with a compact edge architecture and use INT8 quantization to reduce memory and computation.
2. **Algorithm $\rightarrow$ Data:**
   Reduce the computational burden of the input: lower resolution, reduce frame rate, or extract a region-of-interest before inference.
3. **Data $\rightarrow$ Algorithm:**
   Retrain the model on the new input representation. A model trained on 4K/30-FPS inputs cannot simply be assumed to work equally well on the reduced input.
4. **Algorithm $\rightarrow$ Evaluation:**
   Evaluate the complete pipeline on the **target hardware**, including preprocessing and inference.

The key idea:

When Constraints Conflict: Change the System or the Requirement

Suppose reducing the input resolution makes small mold spots too difficult to detect.

When Constraints Conflict: Change the System or the Requirement

Suppose reducing the input resolution makes small mold spots too difficult to detect.

We now have a constraint conflict.

When Constraints Conflict: Change the System or the Requirement

Suppose reducing the input resolution makes small mold spots too difficult to detect.

We now have a constraint conflict.

1. Change the System

Keep the 200-ms requirement.

- Use a lightweight ROI extraction step
- Use a tiny trigger model
- Run a larger model only on suspicious regions
- Use a hardware accelerator
- Change the model architecture

When Constraints Conflict: Change the System or the Requirement

Suppose reducing the input resolution makes small mold spots too difficult to detect.

We now have a constraint conflict.

1. Change the System

Keep the 200-ms requirement.

- Use a lightweight ROI extraction step
- Use a tiny trigger model
- Run a larger model only on suspicious regions
- Use a hardware accelerator
- Change the model architecture

2. Change the Requirement

If the system cannot satisfy all constraints:

- Relax the latency requirement
- Detect events rather than every frame
- Allow occasional cloud processing
- Introduce human confirmation

Stage Interface Contracts

To prevent silent, cascading failures across teams, every lifecycle stage must operate under a strict **Stage Contract** containing inputs, outputs, and invariants.

Stage	Output Deliverable	Quality Invariant
1. Problem	Measurable objectives, explicit deployment targets.	Success criteria are fully quantified.
2. Data	Versioned dataset with schema & validation rules.	Distribution mirrors production environment.
3. Model	Reproducible artifact (weights, code, environment).	Metric goals met within target compute budget.

The Model Contract & Reproducible Artifacts

To satisfy the Model Development Contract, a team cannot just hand over raw weights. The output must be a unified, **Reproducible System Artifact** containing four coupled pieces:

The Model Contract & Reproducible Artifacts

To satisfy the Model Development Contract, a team cannot just hand over raw weights. The output must be a unified, **Reproducible System Artifact** containing four coupled pieces:

1. **Model Weights:** The learned statistical parameters ($\mathcal{O}$ operations cost floor).

The Model Contract & Reproducible Artifacts

To satisfy the Model Development Contract, a team cannot just hand over raw weights. The output must be a unified, **Reproducible System Artifact** containing four coupled pieces:

1. **Model Weights:** The learned statistical parameters ($\mathcal{O}$ operations cost floor).
2. **Inference Code:** The exact code executing forward passes and preprocessing logic.

The Model Contract & Reproducible Artifacts

To satisfy the Model Development Contract, a team cannot just hand over raw weights. The output must be a unified, **Reproducible System Artifact** containing four coupled pieces:

1. **Model Weights:** The learned statistical parameters ($\mathcal{O}$ operations cost floor).
2. **Inference Code:** The exact code executing forward passes and preprocessing logic.
3. **Environment Specification:** The complete dependency graph (Docker layer, CUDA version, library variants).

The Model Contract & Reproducible Artifacts

To satisfy the Model Development Contract, a team cannot just hand over raw weights. The output must be a unified, **Reproducible System Artifact** containing four coupled pieces:

1. **Model Weights:** The learned statistical parameters ($\mathcal{O}$ operations cost floor).
2. **Inference Code:** The exact code executing forward passes and preprocessing logic.
3. **Environment Specification:** The complete dependency graph (Docker layer, CUDA version, library variants).
4. **Configuration:** All hyperparameters, runtime flags, and execution settings.

Unpackaged Model Deployments

The Non-Contract Approach (Naive Weight Dumping)

```
import torch

# Naive approach: Dump raw weights to a shared folder
model = DiabeticRetinopathyCNN()
train_loop(model)
torch.save(model.state_dict(), "/shared/models/latest_weights.pt")

# DANGER: What preprocessing code was used? What version of PyTorch
# was this? What runtime configuration flags were toggled?
# If the serving script uses a different image resize library, it fails silently.
```

During Production Deployment:

When deployment engineers load `latest_weights.pt` in production, a slight version difference in the deep learning framework or underlying GPU kernels yields different predictions. The system breaks without throwing an error.

Implementing the Model Contract: Artifact Tools

We enforce reproducibility by using tools that automatically couple weights, code, configurations, and environment specs into a single immutable block.

The Contract Approach: MLflow Models & Docker Containerization

```python
import mlflow.pyfunc

# Log weights, configuration, and inference code simultaneously
with mlflow.start_run():
    mlflow.log_params({"learning_rate": 0.001, "batch_size": 32})
    mlflow.log_metric("auc", 0.98)

    # Packages weights along with the exact custom preprocessing/serving class
    mlflow.pyfunc.log_model(
        artifact_path="dr_model", python_model=DiabeticRetinopathyModel()
    )
```

```dockerfile
# Seal the entire environment spec, OS layers, and CUDA drivers
FROM pytorch/pytorch:2.4.0-cuda12.1-cudnn8-runtime
COPY requirements.txt /app/requirements.txt
RUN pip install -r /app/requirements.txt --no-cache-dir
```

The Data Contract & Validation Gates

Data collection and preparation is the primary driver of project time (consuming 79% of engineering effort). The **Data Stage Contract** prevents downstream data cascades.

Programmatic Quality Invariants to Enforce:

The Data Contract & Validation Gates

Data collection and preparation is the primary driver of project time (consuming 79% of engineering effort). The **Data Stage Contract** prevents downstream data cascades.

Programmatic Quality Invariants to Enforce:

- **Schema Validation:** Catching missing fields, malformed tensor shapes, or sudden bit-depth drops at point of capture.

The Data Contract & Validation Gates

Data collection and preparation is the primary driver of project time (consuming 79% of engineering effort). The **Data Stage Contract** prevents downstream data cascades.

Programmatic Quality Invariants to Enforce:

- **Schema Validation:** Catching missing fields, malformed tensor shapes, or sudden bit-depth drops at point of capture.
- **Distribution Validation:** Enforcing that feature variations mimic the anticipated real-world production distribution.

The Data Contract & Validation Gates

Data collection and preparation is the primary driver of project time (consuming 79% of engineering effort). The **Data Stage Contract** prevents downstream data cascades.

Programmatic Quality Invariants to Enforce:

- **Schema Validation:** Catching missing fields, malformed tensor shapes, or sudden bit-depth drops at point of capture.
- **Distribution Validation:** Enforcing that feature variations mimic the anticipated real-world production distribution.
- **Provenance Metadata:** Explicitly linking every raw sample to its source camera model, operator profile, and collection environment.

The Data Contract & Validation Gates

Data collection and preparation is the primary driver of project time (consuming 79% of engineering effort). The **Data Stage Contract** prevents downstream data cascades.

Programmatic Quality Invariants to Enforce:

- **Schema Validation:** Catching missing fields, malformed tensor shapes, or sudden bit-depth drops at point of capture.
- **Distribution Validation:** Enforcing that feature variations mimic the anticipated real-world production distribution.
- **Provenance Metadata:** Explicitly linking every raw sample to its source camera model, operator profile, and collection environment.

The Hardware Impact: Validating data schema early shrinks ingest volume before it hits central pipelines, directly minimizing the Iron Law's bandwidth stress term $\left(\frac{D_{\text{vol}}}{\text{BW}}\right)$.

Blind Data Ingestion

The Non-Contract Approach: Blind Data Ingestion

```python
import pandas as pd

def naive_ingest(file_path: str):
    # Blindly load whatever file the data team or clinic dropped
    df = pd.read_csv(file_path)

    # Directly feeds downstream training loops or accelerators
    return df

# CRITICAL VULNERABILITY: If a clinic technician incorrectly updates their camera
# settings, the pixel array dimensions shrink, or labels are written as floats
# instead of strings, this function passes corrupt values straight to training.
```

The Production Impact

Bad or misaligned data feeds downstream model calculations, resulting in a Data Cascade. The model trains on garbage inputs and degrades, but the pipeline throws zero runtime code bugs.

The Data Contract: Programmatic Invariants

We turn data quality invariants into hard code gates using frameworks like **Pandera** or **Great Expectations**. If an update breaks these rules, the pipeline throws an exception.

Concrete Data Quality Gate (Python Example)

```python
import pandas as pd
import pandera as pa

# 1. Define the programmatic contract for incoming image metadata
DataContractSchema = pa.DataFrameSchema({
    "image_id": pa.Column(str, unique=True, nullable=False),
    "camera_model": pa.Column(str, checks=pa.Check.isin(["Fundus-X10", "Fundus-V5"])),
    "resolution_width": pa.Column(int, checks=pa.Check.geq(2048)), # Trap low-res inputs
    "mean_pixel_intensity": pa.Column(float, checks=pa.Check.between(10.0, 245.0)), # Check exposure
})

# 2. Enforce the gate during batch pipeline ingestion
def ingest_clinic_batch(file_path: str):
    df = pd.read_csv(file_path)
    try:
        validated_df = DataContractSchema.validate(df)
        return validated_df
    except pa.errors.SchemaErrors as err:
        print(f"[STAGE CONTRACT VIOLATION]: Ingest halted. \n{err}")
        raise SystemExit("Pipeline locked: Data distribution shifted or schema corrupted.")
```

Monolithic RAM Loading

The Non-System Approach (Out-of-Memory Trigger)

```
import torch

# Naive approach: Load all 100 GB of images directly into RAM
raw_images = load_all_clinic_images_from_disk()
# System crashes here with a fatal Out of Memory (OOM) error!

dataset = torch.tensor(raw_images)
for epoch in range(10):
    for batch in dataset:
        train_step(batch)
```

The Hardware Failure

AI hardware accelerators possess explicit VRAM (Video RAM) capacity boundaries.

Forcing un-chunked array allocations into system storage causes an immediate operating system process termination (‘OOM Kill‘).

Streaming & Memory-Pinned Iterators

We resolve memory bounds by creating a parallelized data delivery network.

The System Optimization (Asynchronous Overlapped I/O)

```
from torch.utils.data import Dataset, DataLoader

class StreamingDataset(Dataset):
    def __getitem__(self, idx):
        # Lazy Loading: Fetch and decompress exactly ONE image from disk at a time
        return load_single_image(idx)

dataloader = DataLoader(
    StreamingDataset(),
    batch_size=32,
    num_workers=4,      # Parallelizes preprocessing across 4 CPU cores
    pin_memory=True,    # Page-locks memory for high-speed PCIe transfer
    prefetch_factor=2   # Proactively pools next batches in cache
)

for batch_x, batch_y in dataloader:
    # Non-blocking transfer keeps the GPU hardware saturated
    bx = batch_x.to("cuda", non_blocking=True)
    gpu_compute(bx)
```

The Blind Production Post-Mortem

The Non-System Approach (Simple Server Metrics Only)

```python
import logging

def predict_and_log(server_request):
    try:
        features = extract_payload(server_request)
        prediction = production_model(features)

        # Logging web health tells you NOTHING about model health
        logging.info(f"Status: 200 | Output Shape: {prediction.shape}")
        return prediction
    except Exception as e:
        logging.error(f"Status: 500 | Error: {str(e)}")
```

The Blind Production Post-Mortem

The Non-System Approach (Simple Server Metrics Only)

```python
import logging

def predict_and_log(server_request):
    try:
        features = extract_payload(server_request)
        prediction = production_model(features)

        # Logging web health tells you NOTHING about model health
        logging.info(f"Status: 200 | Output Shape: {prediction.shape}")
        return prediction
    except Exception as e:
        logging.error(f"Status: 500 | Error: {str(e)}")
```

The Silent Failure Mode

If a field clinic upgrades its imagery scanner, causing input feature values to shift radically, the server will continue returning code 200 `OK`. The model will confidently output junk predictions without ever throwing a code exception.

Automated Statistical Drift Detection

Production pipelines treat telemetry data as an empirical source to identify distribution shifts.

The System Optimization (Statistical Invariant Testing)

```python
import numpy as np
from scipy.stats import ks_2samp

class ProductionDriftMonitor:
    def __init__(self, training_baseline_features):
        # Establish reference data distribution contract
        self.baseline = training_baseline_features

    def evaluate_production_batch(self, current_batch_features):
        # 1. Compute Two-Sample Kolmogorov-Smirnov statistical test
        statistic, p_value = ks_2samp(self.baseline, current_batch_features)

        # 2. Gate system stability based on a threshold alpha significance level
        if p_value < 0.05:
            # Drop an active telemetry alert to trigger upstream automated retraining
            print(f"[SYSTEM SHIFT ALERT] P-value: {p_value:.4f}. Distribution drifted!")
            trigger_pipeline_rollback()
            return "Unstable"

        return "Stable"
```

The Shocklog: When statistical tests identify data drift, the

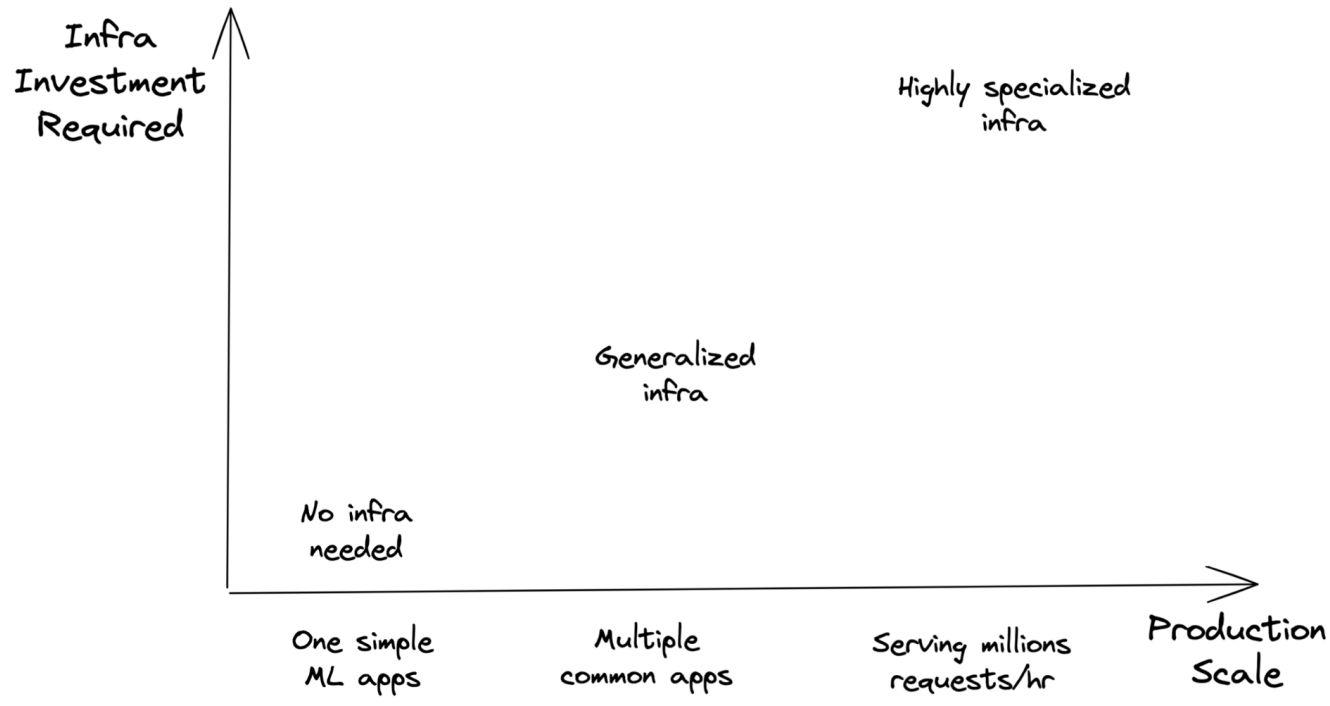
ML Infrastructure

ML infrastructure are additional components that support the ML workflow.
They are not part of the core workflow, but they are essential for making the workflow efficient and reliable.

ML Infrastructure Examples

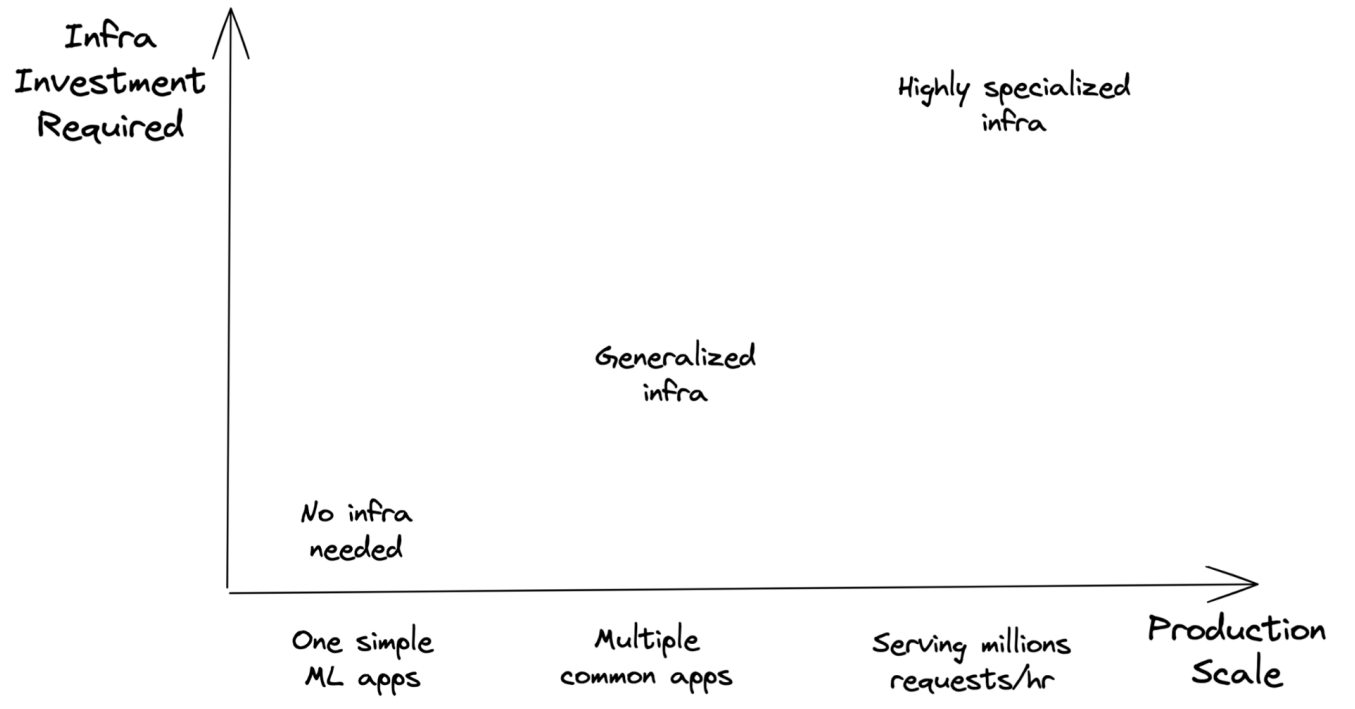
- Data versioning and management systems (e.g., DVC, Pachyderm)
- Model versioning and management systems (e.g., MLflow, Weights and Biases)
- Experiment tracking and logging systems (e.g., TensorBoard, Comet.ml)
- Automated machine learning (AutoML) platforms (e.g., Google AutoML, H2O.ai)
- Cloud-based ML platforms (e.g., AWS SageMaker, Azure ML, Google AI Platform)
- Continuous integration and deployment (CI/CD) pipelines for ML (e.g., Jenkins, GitLab CI/CD)

Different infrastructure needs



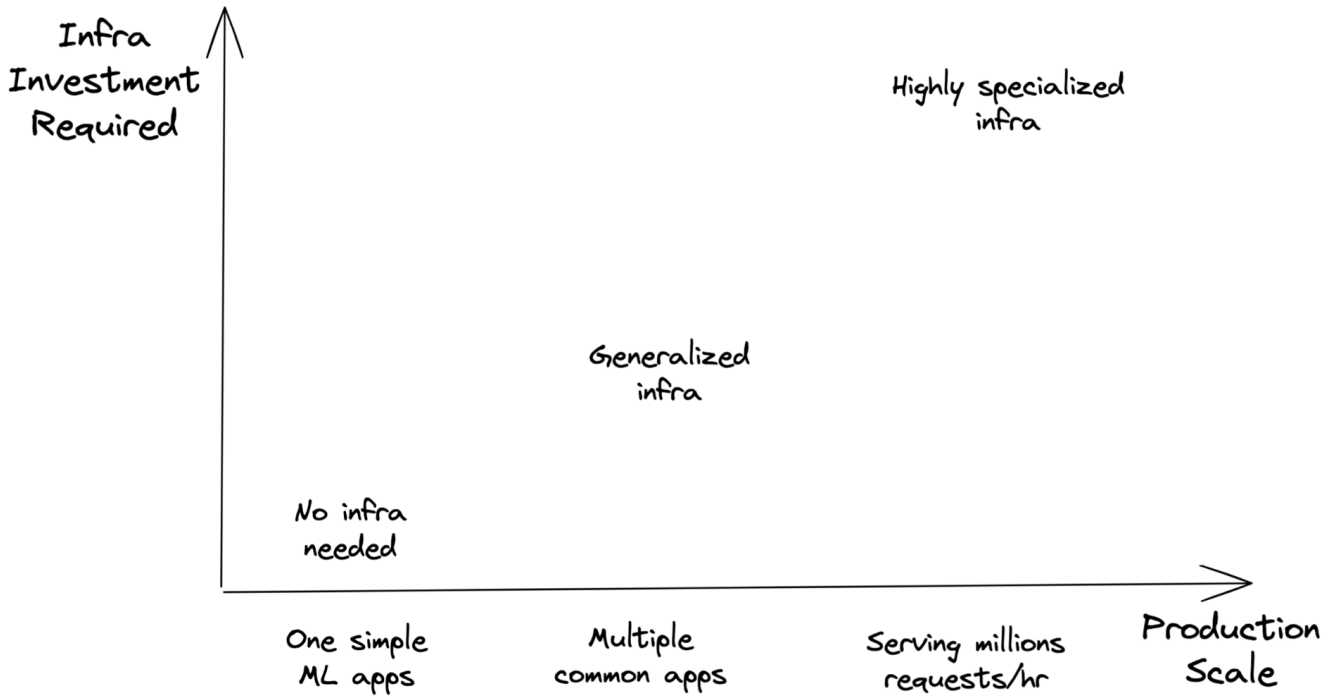
Different infrastructure needs-Big

Google at 63K requests/sec, 234M requests/hr
Most of it resides on the cloud.

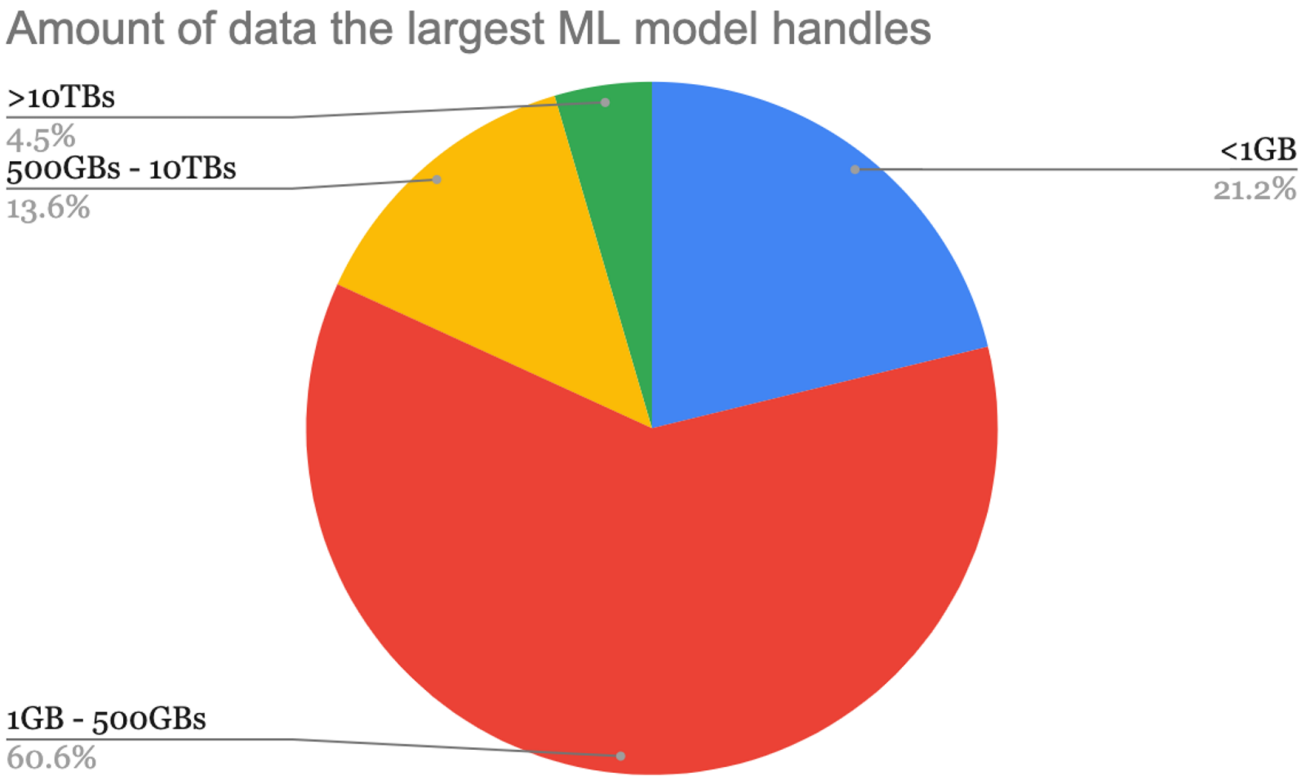


Different infrastructure needs-Medium

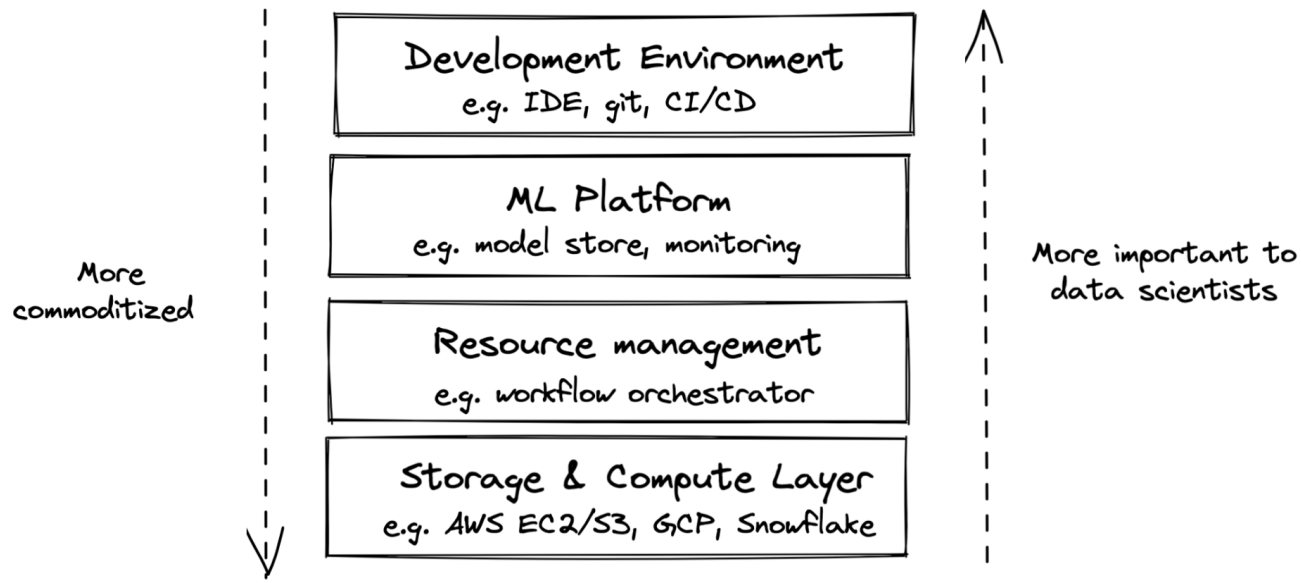
Medium size: GB - TBs of data daily, 10s - 100s data scientists, 3+ models Maybe on the cloud or on the edge



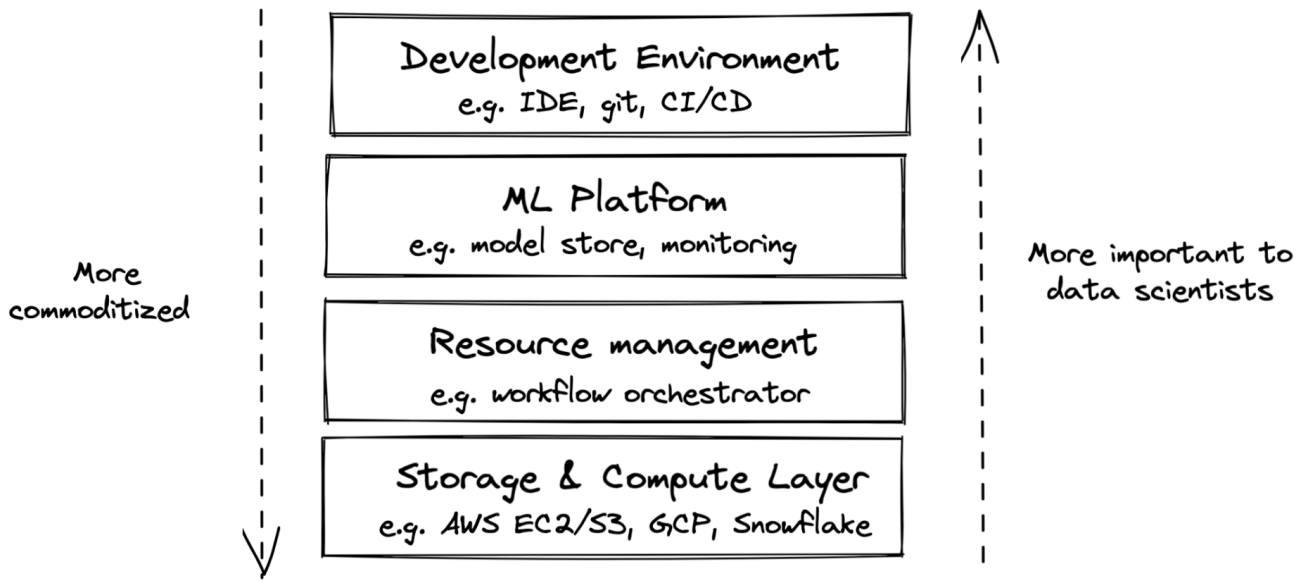
Different infrastructure needs-data wise



Infrastructure Layers



Infrastructure Layers–Order



ML Platform

1. Anna started working on recsys at company X
2. To deploy recsys, Anna's team need to build tool like model deployment, model store, feature store, etc.
3. Other teams at X started deploying models and needed to build the same tools
4. X decided to have a centralized platform to serve multiple ML use cases
5. Key features: Model deployment, Model store, Feature store

Storage Layer-1

Attribute	Conventional Database	Data Warehouse	Data Lake
Purpose	Operational and transactional	Analytical and reporting	Storage for raw and diverse data for future processing
Data type	Structured	Structured	Structured, semi-structured, and unstructured
Scale	Small to medium volumes	Medium to large volumes	Large volumes of diverse data
Performance Optimization	Optimized for transactional queries (OLTP)	Optimized for analytical queries (OLAP)	Optimized for scalable storage and retrieval
Examples	MySQL, PostgreSQL, Oracle DB	Google BigQuery, Amazon Redshift, Microsoft Azure Synapse	Google Cloud Storage, AWS S3, Azure Data Lake Storage

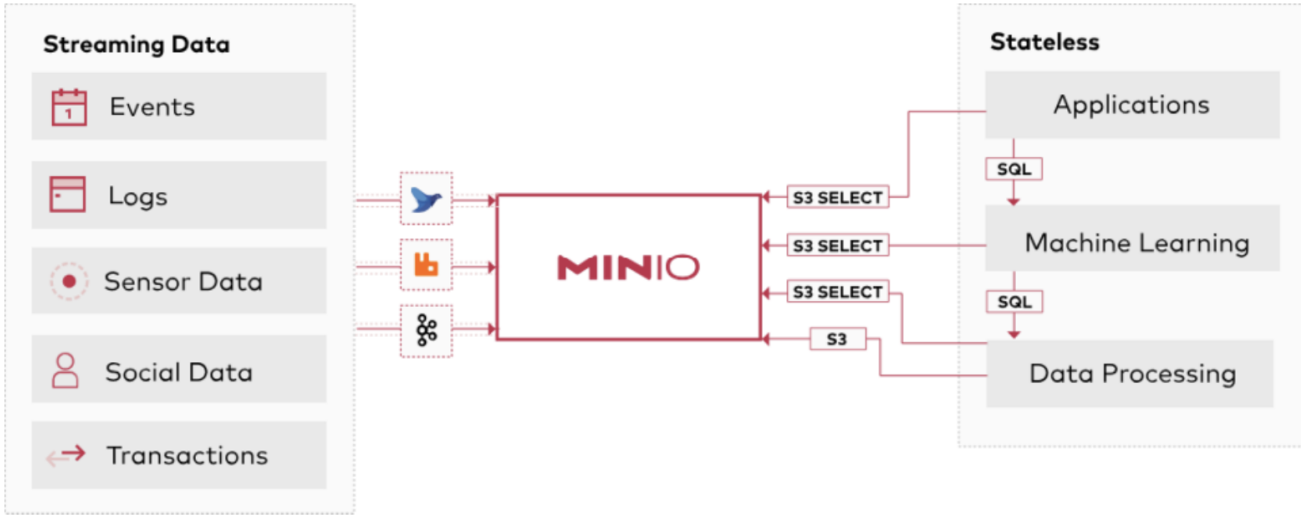
ML pipelines rely on high-throughput reads, large-scale data scans, and evolving schemas.

Storage Layer-ML issues

- Capability to handle large, often dense, numerical arrays efficiently
  e.g.,: GPT3 has 175 billion parameters, requiring approximately 350 GB of storage just for the model weights
- Diversity of data types.
  e.g.,: structured tabular data to unstructured images, audio, and text
- Versioning for both datasets and models
  e.g.,: iterative nature of ML development
- Substantial intermediate data, including partial model updates, gradients, and checkpoints. e.g.,: distributed training

MinIO

Kubernetes-native object storage



MinIO

- The concept of object storage is similar to that of a standard Unix file system, but instead of directories and files, we use buckets and objects.
- Buckets can be nested into a hierarchy just like directories, and objects can be thought of as just a collection of bytes.
- Those collections can be arbitrary byte arrays or normal files like images, PDFs, and more.

```
/
/images/
 imge1.png
 image2.jpg
/videos/
 video1.mp4
/users/
 /john.doe/
  3rd quarter revenue report.docx
```

How much compute for different ML tasks?

Measure of compute is memory and FLOPS
e.g., A Cloud TPU v2 can perform up to 180 teraflops, and the TPU v3 up to 420 teraflops.

MLPerf is a benchmark for ML

https://www.youtube.com/watch?v=MKII0KXDqn4

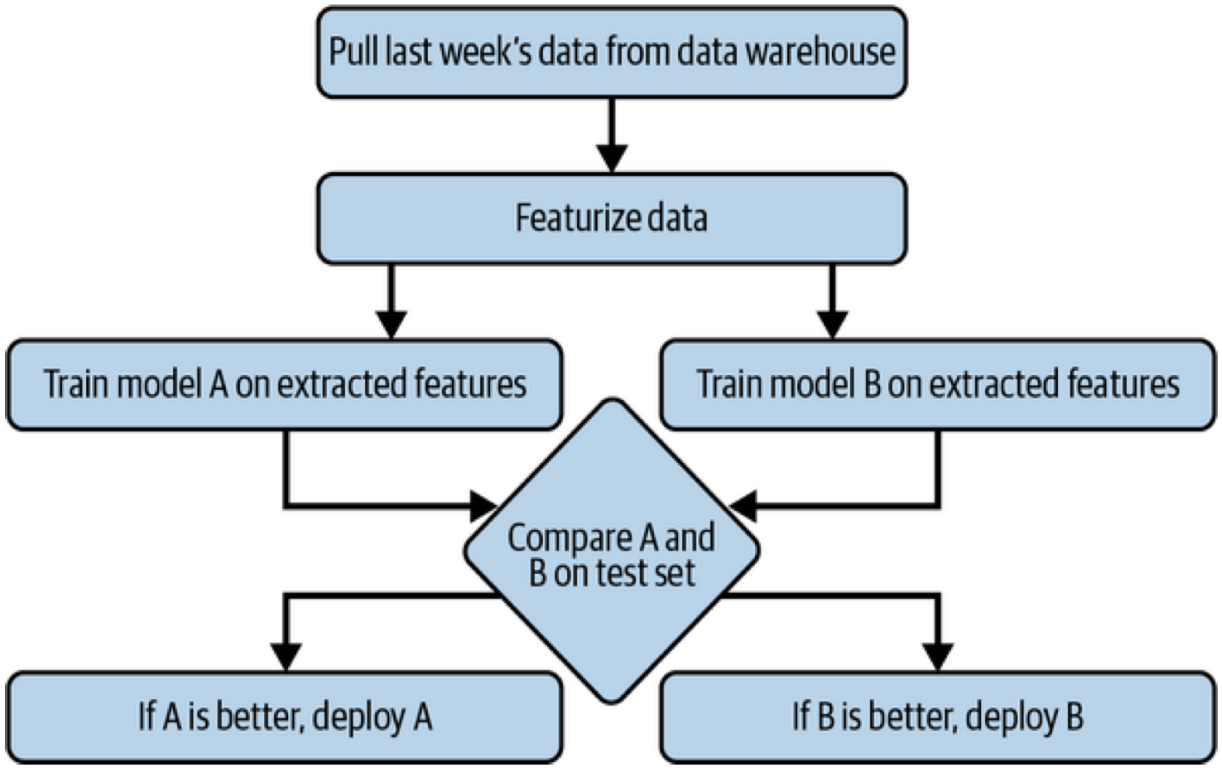
ML Workflow

1. Pull last week's data from data warehouses.
2. Extract features from this pulled data.
3. Train two models, A and B, on the extracted features.
4. Compare A and B on the test set.
5. Deploy A if A is better; otherwise deploy B.

Each step depends on the success of the previous step.

DAGs in Workflows

Most workflow management tools require you to specify your workflows in a form of DAGs.



Schedulers

- Schedulers are cron programs that can handle dependencies. It takes in the DAG of a workflow and schedules each step accordingly.
- You can even schedule to start a job based on an event-based trigger, e.g., start a job whenever an event X happens.
- Keep queues to keep track of jobs, which can be queued or prioritized.
- Important: schedulers need to be aware of the resources available and the resources needed to run each job

Slurm

Slurm, where you specify the job name, the time when the job needs to be executed, and the amount of memory and CPUs to be allocated for the job.

```
#!/bin/bash
#SBATCH -J JobName
#SBATCH --time=11:00:00        # When to start the job
#SBATCH --mem-per-cpu=4096    # Memory, in MB, to be allocated per
CPU
#SBATCH --cpus-per-task=4           # Number of cores per task
```

Ideally scheduler needs to. optimize for resource utilization since they have information on resources available, jobs to run, and resources needed for each job to run.

Orchestrators

Orchestrators are concerned with where to get those resources–machines, instances, clusters, service-level grouping. Orchestrators concern themselves with replication.

Kubernetes

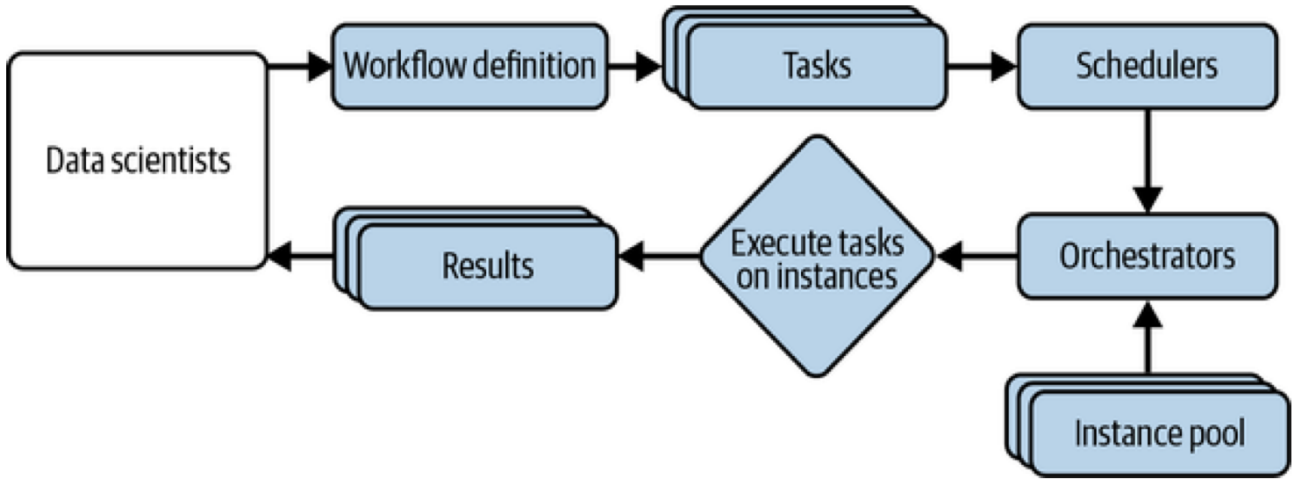
The confusing world of schedulers and orchestrators

Schedulers like Slurm and Google's Borg have some orchestrating capacity
Orchestrators like HashiCorp Nomad and K8s come with some scheduling capacity

separate schedulers and orchestrators: Spark's job scheduler on top of Kubernetes or AWS Batch scheduler on top of EKS.

data science–specific orchestrators including Airflow, Argo, Prefect, and Dagster have their own schedulers.

Data Science Workflow Management



Data Science Workflow Management

Can be defined using: Code (Python) Configuration files (YAML)
Examples: Airflow, Argo, KubeFlow, Metaflow

AirFlow

1st gen data science workflow management
Champion of “configuration-as-code”
Wide range of operators to expand capabilities

Issues: Monolithic
The entire workflow as a container
Non-parameterized
E.g. need to define another workflow if you want to change learning rate
Static DAG
Can’t handle workloads with unknown number of records

```python
dag = DAG(
    'docker_sample',
    default_args={
        'owner': 'airflow',
        'depends_on_past': False,
        'email': ['airflow@example.com'],
        'email_on_failure': False,
        'email_on_retry': False,
        'retries': 1,
        'retry_delay': timedelta(minutes=5),
    },
    schedule_interval=timedelta(minutes=10),
    start_date=days_ago(2),
)

t1 = BashOperator(task_id='print_date', bash_command='date', dag=dag)

t2 = BashOperator(task_id='sleep', bash_command='sleep 5', retries=3, dag=dag)

t3 = DockerOperator(
    api_version='1.19',
    docker_url='tcp://localhost:2375',  # Set your docker URL
    command='/bin/sleep 30',
    image='centos:latest',
    network_mode='bridge',
    task_id='docker_op_tester',
    dag=dag,
)


t4 = BashOperator(task_id='print_hello', bash_command='echo "hello world!!!"', dag=dag)


t1 >> t2
t1 >> t3
```

Argo: next gen

Created to address Airflow's problems
Containerized
Fully parameterized
Dynamic DAG

Argo: cons

YAML-based configs
Can get very messy
Only run on K8s clusters
Can't easily test in dev environment

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: coinflip-
  annotations:
    workflows.argoproj.io/description: |
      This is an example of coin flip defined as a sequence of conditional steps.
      You can also run it in Python: https://couler-proj.github.io/couler/examples/#coin-flip
spec:
  entrypoint: coinflip
  templates:
  - name: coinflip
    steps:
    - - name: flip-coin
        template: flip-coin
    - - name: heads
        template: heads
        when: "{{steps.flip-coin.outputs.result}} == heads"
      - name: tails
        template: tails
        when: "{{steps.flip-coin.outputs.result}} == tails"

  - name: flip-coin
    script:
      image: python:alpine3.6
      command: [python]
      source: |
        import random
        result = "heads" if random.randint(0,1) == 0 else "tails"
        print(result)

  - name: heads
    container:
      image: alpine:3.6
      command: [sh, -c]
      args: ["echo \"it was heads\""]

  - name: tails
```

MLPlatform: MLflow

Software to manage the ML development and deployment process, from data to experimentation to production

MLFlow

Based on an open interface design philosophy: make it easy to connect arbitrary ML code and tools into the platform Simple command-line and REST APIs rather than environment-specific Easy to add to existing software
https://youtu.be/W7Qpu7k0BbM