MLOps Assignment 2 — End-to-End Pipeline Report

Course: MLOps (S1-25_AIMLCZG523)
Use Case: Binary Image Classification — Cats vs Dogs for a Pet Adoption Platform
Dataset: Kaggle Cats and Dogs Dataset
Student: Azhar N

Table of Contents

---

1. Project Overview

This project implements a complete, production-grade MLOps pipeline for binary image classification (Cats vs Dogs) intended for a pet adoption platform. The pipeline covers every stage of the MLOps lifecycle:

- **Data versioning** with DVC
- **Model training** with MLflow experiment tracking
- **REST API inference service** using FastAPI
- **Containerization** with Docker (multi-stage build)
- **CI/CD automation** using GitHub Actions
- **Image registry** using GitHub Container Registry (GHCR)
- **Deployment** with Docker Compose
- **Monitoring** with Prometheus metrics and structured JSON logging

Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────────┐
│                        Developer Workstation                        │
│                                                                     │
│  ┌──────────────┐    ┌──────────────┐    ┌──────────────────────┐  │
│  │  Git + DVC   │───▶│ src/train.py │───▶│  MLflow Tracking UI  │  │
│  │  versioning  │    │ (ResNet-18)  │    │ (params/metrics/     │  │
│  └──────────────┘    └──────────────┘    │  confusion matrix)   │  │
│                              │           └──────────────────────┘  │
│                              ▼                                      │
│                      ┌──────────────┐                               │
│                      │ models/*.pt  │                               │
```

```
│                         └──────────────┘                     │
└───────────────────────────────────────────────────────────────┘
                                │ git push
                                ▼
┌───────────────────────────────────────────────────────────────┐
│                       GitHub Actions CI/CD                     │
│                                                               │
│  ┌─────────────┐     ┌──────────────────┐     ┌────────────────────┐  │
│  │ Run pytest  │────▶│ docker build     │────▶│ Push to GHCR       │  │
│  │ (12 tests)  │     │ (multi-stage)    │     │ ghcr.io/azhar-n/   │  │
│  └─────────────┘     └──────────────────┘     │ catdog-classifier  │  │
│                                               └────────────────────┘  │
│  ┌──────────────────────────────────────────────────────────┐  │
│  │ Deploy Job: docker compose up → smoke_test.sh → teardown │  │
│  └──────────────────────────────────────────────────────────┘  │
└───────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌───────────────────────────────────────────────────────────────┐
│                 Deployed Service (Docker Compose)              │
│                                                               │
│  ┌──────────────────────┐        ┌──────────────────────────┐  │
│  │ catdog-api           │        │ Prometheus               │  │
│  │ FastAPI :8000        │◀──────▶│  :9090                   │  │
│  │ GET  /health         │        │ Scrapes /metrics every 15s │ │
│  │ POST /predict        │        └──────────────────────────┘  │
│  │ GET  /metrics        │                                      │
│  └──────────────────────┘                                      │
└───────────────────────────────────────────────────────────────┘
```

---

2. M1 — Model Development & Experiment Tracking

2.1 Data & Code Versioning

Git — Source Code Versioning

All source code, configuration files, scripts, and CI/CD definitions are tracked in Git. The repository follows a structured layout with clear separation between data, source, application, testing, deployment, and monitoring concerns.

DVC — Dataset Versioning

DVC is used to version the dataset and track the full data pipeline from raw images to trained model artifacts.

`dvc.yaml` **— Pipeline Definition:**

```yaml
stages:
  preprocess:
    cmd: python src/data_preprocessing.py --raw-dir data/raw --out-dir
data/processed
```

```yaml
    deps:
      - src/data_preprocessing.py
      - data/raw
    outs:
      - data/processed
    params:
      - params.yaml:
        - preprocess.image_size
        - preprocess.split_ratios

  train:
    cmd: python src/train.py --data-dir data/processed ...
    deps:
      - src/train.py
      - src/model.py
      - data/processed
    outs:
      - models/cat_dog_model.pt
      - models/loss_curves.png
      - models/confusion_matrix.png
    metrics:
      - metrics.json
```

`params.yaml` — Tracked Hyperparameters:

```yaml
preprocess:
  image_size: 224
  split_ratios:
    train: 0.8
    val: 0.1
    test: 0.1

train:
  epochs: 10
  batch_size: 32
  lr: 0.001
  weight_decay: 0.0001
```

Running the pipeline: `dvc repro` re-executes only stages whose dependencies have changed, ensuring full reproducibility.

2.2 Data Preprocessing

File: `src/data_preprocessing.py`

The preprocessing pipeline:

1. Reads raw images from `data/raw/cat/` and `data/raw/dog/`
2. Converts all images to RGB mode (handles RGBA, grayscale, palette images)
3. Resizes every image to **224×224** using LANCZOS resampling

4. Performs a reproducible **80/10/10** train/validation/test split (seed=42)
5. Writes processed images to `data/processed/{train,val,test}/{cat,dog}/`

Key functions:

- `resize_image(src, dst)` — resize + RGB conversion for a single image
- `split_files(files, ratios, seed)` — deterministic stratified file splitting
- `create_dummy_dataset(out_dir, n_per_class)` — generates synthetic data for CI

2.3 Data Augmentation

File: `src/utils.py`

Training images apply a stochastic augmentation pipeline:

```
transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225]),  # ImageNet stats
])
```

Validation/test/inference images use only resize + normalize (no stochastic augmentation).

2.4 Model Architecture

File: `src/model.py`

The baseline model is a **ResNet-18** fine-tuned for binary classification using transfer learning from ImageNet weights.

```
Input: [B, 3, 224, 224]
  └─▶ ResNet-18 Backbone (pretrained on ImageNet)
        └─▶ [B, 512]  (GlobalAvgPool output)
              └─▶ Dropout(p=0.3)
                    └─▶ Linear(512 → 1)
Output: [B, 1]  (raw logit → apply sigmoid for probability)
```

- **Loss function:** `BCEWithLogitsLoss` (numerically stable binary cross-entropy)
- **Optimizer:** Adam with weight decay 1e-4
- **LR scheduler:** StepLR — halves learning rate every 5 epochs
- **Threshold:** sigmoid(logit) ≥ 0.5 → dog, < 0.5 → cat
- **Total parameters:** ~11.2M (ResNet-18 default)

The classification head was replaced (fully unfrozen fine-tuning):

```
self.backbone.fc = nn.Sequential(
    nn.Dropout(p=0.3),
    nn.Linear(in_features, 1),
)
```

2.5 Experiment Tracking with MLflow

File: `src/train.py`

Every training run logs the following to MLflow:

Category	Items Logged
Parameters	epochs, batch_size, learning_rate, weight_decay, model arch, pretrained flag, optimizer
Metrics (per epoch)	train_loss, train_acc, val_loss, val_acc
Metrics (final)	test_loss, test_acc
Artifacts	`model/cat_dog_model.pt`, `charts/loss_curves.png`, `charts/confusion_matrix.png`
Model	Logged via `mlflow.pytorch.log_model()` for model registry

Running a tracked experiment:

```
python src/train.py --data-dir data/processed --epochs 10 --lr 0.001 --run-name baseline-resnet18
mlflow ui  # View at http://localhost:5000
```

---

3. M2 — Model Packaging & Containerization

3.1 FastAPI Inference Service

File: `app/main.py`

The model is wrapped in a FastAPI application with the following endpoints:

Method	Endpoint	Description
`GET`	`/health`	Returns service status and whether model is loaded
`POST`	`/predict`	Accepts image upload, returns label + class probabilities
`GET`	`/metrics`	Exposes Prometheus metrics for scraping
`GET`	`/docs`	Auto-generated Swagger UI (FastAPI built-in)

Health Check Response:

```json
{
  "status": "ok",
  "model_loaded": true
}
```

Prediction Response:

```json
{
  "label": "cat",
  "confidence": 0.9231,
  "cat_probability": 0.9231,
  "dog_probability": 0.0769
}
```

Example curl call:

```
curl -X POST http://localhost:8000/predict \
  -F "file=@my_cat_image.jpg" | python3 -m json.tool
```

3.2 Predictor Module

File: `app/predictor.py`

The `Predictor` class is instantiated once at startup via FastAPI's `lifespan` context manager, preventing repeated model loading per request:

- Loads checkpoint from `MODEL_PATH` environment variable (default: `models/cat_dog_model.pt`)
- Applies identical normalization as training (ImageNet mean/std)
- Returns label, confidence, and both class probabilities
- Fully thread-safe (stateless inference with `@torch.no_grad()`)

3.3 Environment Specification

File: `requirements.txt` — All key library versions are pinned:

```
torch==2.2.0
torchvision==0.17.0
fastapi==0.109.2
uvicorn[standard]==0.27.1
mlflow==2.10.2
dvc==3.40.1
prometheus-client==0.20.0
pytest==8.0.1
```

```
Pillow==10.2.0
numpy==1.26.4
scikit-learn==1.4.0
```

3.4 Dockerfile — Multi-Stage Build

File: `Dockerfile`

A two-stage build minimizes the final image size by separating the build environment (with gcc/g++) from the runtime environment:

```
# Stage 1: Builder — installs all Python packages with native compilation
FROM python:3.10-slim AS builder
WORKDIR /build
RUN apt-get install gcc g++
COPY requirements.txt .
RUN pip install --no-cache-dir --user -r requirements.txt

# Stage 2: Runtime — copies only installed packages, no build tools
FROM python:3.10-slim AS runtime
WORKDIR /app
COPY --from=builder /root/.local /root/.local
ENV PATH=/root/.local/bin:$PATH
COPY app/ ./app/
COPY src/ ./src/
RUN mkdir -p ./models
COPY models/ ./models/
EXPOSE 8000
HEALTHCHECK ...
CMD ["python", "-m", "uvicorn", "app.main:app", "--host", "0.0.0.0", "--port",
"8000"]
```

Local build and verification:

```
# Build
docker build -t catdog-classifier:local .

# Run (with model mounted)
docker run -p 8000:8000 \
  -v $(pwd)/models:/app/models:ro \
  -e MODEL_PATH=/app/models/cat_dog_model.pt \
  catdog-classifier:local

# Verify health
curl http://localhost:8000/health

# Predict
curl -X POST http://localhost:8000/predict -F "file=@test_image.jpg"
```

---

4. M3 — CI Pipeline for Build, Test & Image Creation

4.1 Automated Testing

Files: `tests/test_preprocessing.py`, `tests/test_inference.py`

12 unit tests covering two key modules:

`test_preprocessing.py` (M3.1 — Data preprocessing tests):

Test Class	Test	What It Verifies
`TestResizeImage`	`test_resize_to_224`	Output image is exactly 224×224
`TestResizeImage`	`test_converts_to_rgb`	RGBA/L images converted to RGB
`TestResizeImage`	`test_creates_parent_dirs`	Nested destination dirs created
`TestGetImageFiles`	`test_finds_jpeg_and_png`	Only image extensions collected
`TestGetImageFiles`	`test_recursive_search`	Nested directories searched
`TestGetImageFiles`	`test_empty_directory`	Returns empty list correctly
`TestSplitFiles`	`test_split_ratios`	80/10/10 proportions correct
`TestSplitFiles`	`test_no_data_leakage`	No file appears in 2 splits
`TestSplitFiles`	`test_reproducibility`	Same seed → same split
`TestCreateDummyDataset`	`test_creates_images`	N images per class created
`TestCreateDummyDataset`	`test_images_are_valid`	Images are openable/valid RGB

`test_inference.py` (M3.1 — Model inference tests):

Test Class	Test	What It Verifies
`TestModelArchitecture`	`test_output_shape`	Output shape is `[batch, 1]`
`TestModelArchitecture`	`test_output_is_finite`	No NaN/Inf in logits
`TestModelArchitecture`	`test_sigmoid_in_range`	Probabilities ∈ [0, 1]
`TestTransforms`	`test_val_transform_output_shape`	Tensor shape `[3, 224, 224]`
`TestTransforms`	`test_val_transform_normalized`	ImageNet normalization applied
`TestComputeAccuracy`	`test_all_correct`	Accuracy = 1.0 for perfect preds
`TestComputeAccuracy`	`test_all_wrong`	Accuracy = 0.0 for all wrong
`TestComputeAccuracy`	`test_half_correct`	Accuracy = 0.5 for half correct
`TestInferencePipeline`	`test_end_to_end_inference`	Full PIL→tensor→logit→label pipeline

Tests use a randomly initialized model (no checkpoint file required), making them fully runnable in CI without model artifacts.

Run locally:

```
pytest tests/ -v --tb=short
```

4.2 CI Setup — GitHub Actions

File: `.github/workflows/ci-cd.yml`

The pipeline triggers on:

- Every **push** to `main` or `develop` branches
- Every **pull request** to `main`

Job 1: Run Unit Tests (all branches + PRs)

```
Checkout → Setup Python 3.10 → pip install -r requirements.txt → pytest tests/
```

Test results are uploaded as a GitHub Actions artifact (JUnit XML).

Job 2: Build & Push Docker Image (main branch pushes only)

```
Checkout → Normalize owner to lowercase → QEMU setup → Docker Buildx
  → Login to GHCR with GITHUB_TOKEN → Extract metadata (tags/labels)
  → docker build → Push to ghcr.io/azhar-n/catdog-classifier
```

Key design decisions:

- **GHCR** instead of Docker Hub — uses built-in `GITHUB_TOKEN`, zero manual secrets
- **Multi-arch QEMU** setup for potential ARM compatibility
- **GHA layer caching** (`cache-from: type=gha`) for faster builds
- **Git SHA tagging** — each image tagged with `sha-<commit>` and `latest`

Job 3: Deploy & Smoke Test (main branch, after build)

```
Checkout → Set IMAGE_TAG + lowercase REPO_OWNER
  → docker compose up -d --wait --timeout 60
  → bash deployment/smoke_test.sh
  → docker compose down (cleanup)
```

4.3 Artifact Publishing

Images are pushed to **GitHub Container Registry (GHCR)**:

```
ghcr.io/azhar-n/catdog-classifier:latest
ghcr.io/azhar-n/catdog-classifier:sha-<git-commit>
```

No manual secrets required — authentication uses the automatically-injected `GITHUB_TOKEN`.

---

5. M4 — CD Pipeline & Deployment

5.1 Deployment Target — Docker Compose

File: `deployment/docker-compose.yml`

Docker Compose was chosen as the deployment target for its simplicity, reproducibility, and suitability for a single-node deployment. The compose stack includes two services:

```yaml
services:
  catdog-api:
    image: ghcr.io/${REPO_OWNER:-localuser}/catdog-classifier:${IMAGE_TAG:-latest}
    build:                  # Allows local development with --build flag
      context: ..
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    environment:
      - MODEL_PATH=/app/models/cat_dog_model.pt
    volumes:
      - ../models:/app/models:ro    # Model mounted at runtime
    healthcheck:
      test: python -c "import urllib.request;
urllib.request.urlopen('http://localhost:8000/health')"
      interval: 30s
      timeout: 10s
      retries: 3

  prometheus:
    image: prom/prometheus:v2.50.1
    ports:
      - "9090:9090"
    volumes:
      - ../monitoring/prometheus.yml:/etc/prometheus/prometheus.yml:ro
```

Key design: The model file is **not baked into the image** — it's volume-mounted at runtime. This keeps the Docker image lean and allows model updates without rebuilding.

5.2 CD/GitOps Flow

The CD flow is fully integrated into the GitHub Actions pipeline:

```
git push → main branch
     │
     ▼
Job 1: Unit Tests PASS
     │
     ▼
Job 2: Build + Push image → ghcr.io/azhar-n/catdog-classifier:sha-<hash>
     │
     ▼
Job 3: Deploy
     ├─ Pull image from GHCR
     ├─ docker compose up -d --wait --timeout 60
     ├─ Run smoke_test.sh
     └─ docker compose down (CI cleanup)
```

Every push to `main` triggers a full redeploy with the newly built image. The `IMAGE_TAG` is set to `sha-<git-short-hash>` ensuring exact traceability of which commit is deployed.

5.3 Smoke Tests

File: `deployment/smoke_test.sh`

The smoke test script runs after every deployment and fails the pipeline if critical checks fail:

Test 1 — Health endpoint (hard failure):

```
HTTP_STATUS=$(curl -o /dev/null -sw "%{http_code}" "${BASE_URL}/health")
# Fails pipeline if not 200
```

Test 2 — Prediction endpoint:

```
# Generates a 224×224 synthetic JPEG in Python
python3 -c "from PIL import Image; import numpy as np; ..."

# POSTs to /predict
PREDICT_STATUS=$(curl -X POST "${BASE_URL}/predict" -F "file=@${TMPIMG}")
# 200 → validates label is "cat" or "dog"
# 503 → warns (model not loaded in CI) but does NOT fail
```

Exit behaviour:

- `FAIL > 0` → `exit 1` (pipeline fails)
- All warnings → `exit 0` (pipeline passes)

---

6. M5 — Monitoring, Logs & Final Submission

6.1 Request/Response Logging

File: `app/main.py`

All requests are logged in structured JSON format for easy parsing by log aggregation tools (ELK, CloudWatch, etc.):

```
logging.basicConfig(
    format='{"time": "%(asctime)s", "level": "%(levelname)s", "message": "%
(message)s"}',
)
```

Sample log entry for a prediction:

```
{"time": "2026-02-22 10:15:42,123", "level": "INFO",
 "message": "predict | label=cat confidence=0.9231 latency=0.042s file=pet.jpg"}
```

What is logged (no sensitive data):

- Prediction label and confidence
- Request latency in seconds
- Uploaded filename (not content)
- Model load/failure events at startup

6.2 Prometheus Metrics

Three custom Prometheus metrics are tracked in the inference service:

Metric	Type	Labels	Description
`catdog_request_total`	Counter	`endpoint`, `status`	Total request count by endpoint and HTTP status
`catdog_request_latency_seconds`	Histogram	`endpoint`	Request latency distribution (buckets: 0.05s–5.0s)
`catdog_prediction_label_total`	Counter	`label`	Count of `cat` vs `dog` predictions

Prometheus scrape config (`monitoring/prometheus.yml`):

```
scrape_configs:
  - job_name: catdog-api
    static_configs:
      - targets: ['catdog-api:8000']
    metrics_path: /metrics
    scrape_interval: 15s
```

Viewing metrics:

```
curl http://localhost:8000/metrics
# Prometheus UI: http://localhost:9090
```

6.3 Model Performance Tracking (Post-Deployment)

File: `monitoring/simulate_requests.py`

The simulation script sends a batch of requests with known ground-truth labels to the deployed API and computes a performance report:

How it works:

- Sends `N` synthetic images to `/predict` (default: 50)
- Half are warm-toned (orange) images labeled "cat", half are cool-toned (blue) labeled "dog"
- Compares each prediction against its true label
- Saves a JSON performance report

Running the simulation:

```
python monitoring/simulate_requests.py --n 50 --url http://localhost:8000
```

Sample output:

```
  [01/50] true=cat  pred=cat  conf=0.8921  latency=38.2ms  ✓
  [02/50] true=dog  pred=dog  conf=0.9134  latency=35.7ms  ✓
  ...

=======================================================
  Post-Deployment Performance Report
  Generated: 2026-02-22 10:30:00 UTC
=======================================================
  Total requests     : 50
  Successful         : 50
  Overall Accuracy   : 94.0%  (47/50)
  Cat accuracy       : 25/25
  Dog accuracy       : 22/25
  Avg confidence     : 0.8834
  Avg latency        : 42.1 ms
  P95 latency        : 68.3 ms
=======================================================
  Report saved → monitoring/performance_report.json
```

The JSON report is saved to `monitoring/performance_report.json` for archival and comparison across deployments.

7. Project Structure

```
Assignment2/
├── .github/
│   └── workflows/
│       └── ci-cd.yml             # GitHub Actions CI/CD pipeline
├── .dvc/                          # DVC internals
├── app/
│   ├── __init__.py
│   ├── main.py                   # FastAPI app, endpoints, Prometheus metrics
│   ├── predictor.py              # Model loading and inference
│   └── schemas.py                # Pydantic request/response schemas
├── deployment/
│   ├── docker-compose.yml        # Multi-service deployment manifest
│   └── smoke_test.sh             # Post-deploy smoke test script
├── models/
│   ├── .gitkeep                  # Tracked placeholder (model files gitignored)
│   ├── cat_dog_model.pt          # Trained model checkpoint (local only)
│   ├── loss_curves.png           # Training curves (DVC artifact)
│   └── confusion_matrix.png      # Confusion matrix (DVC artifact)
├── monitoring/
│   ├── prometheus.yml            # Prometheus scrape configuration
│   └── simulate_requests.py      # Post-deployment batch simulation + report
├── src/
│   ├── __init__.py
│   ├── data_preprocessing.py     # Image resize, split, augmentation pipeline
│   ├── model.py                  # ResNet-18 model definition
│   ├── train.py                  # Training loop with MLflow tracking
│   └── utils.py                  # Transforms, accuracy, visualization
├── tests/
│   ├── __init__.py
│   ├── test_preprocessing.py     # 11 unit tests for preprocessing functions
│   └── test_inference.py         # 9 unit tests for model/inference functions
├── data/
│   ├── raw/                      # Raw Kaggle dataset (DVC tracked)
│   └── processed/                # Preprocessed 224×224 images (DVC tracked)
├── Dockerfile                    # Multi-stage Docker build
├── dvc.yaml                      # DVC pipeline stages
├── dvc.lock                      # DVC pipeline lock file
├── params.yaml                   # DVC-tracked hyperparameters
├── requirements.txt              # Pinned Python dependencies
├── .gitignore                    # Git ignore rules
└── README.md                     # Project setup guide
```

---

8. Tools & Technology Stack

Category	Tool	Version	Purpose
Language	Python	3.10	All scripting and ML code

Category	Tool	Version	Purpose
ML Framework	PyTorch	2.2.0	Model training and inference
CV Library	Torchvision	0.17.0	ResNet-18 backbone + transforms
Image Processing	Pillow	10.2.0	Image loading and preprocessing
Experiment Tracking	MLflow	2.10.2	Run tracking, metrics, artifact logging
Data Versioning	DVC	3.40.1	Dataset and pipeline versioning
Web Framework	FastAPI	0.109.2	REST API inference service
ASGI Server	Uvicorn	0.27.1	Production-grade async server
Monitoring	Prometheus Client	0.20.0	Metrics collection and exposition
Monitoring	Prometheus	2.50.1	Metrics scraping and storage
Containerization	Docker	latest	Image build and runtime
Orchestration	Docker Compose	v2	Multi-service deployment
CI/CD	GitHub Actions	-	Automated test, build, deploy
Container Registry	GHCR	-	Docker image storage (`ghcr.io`)
Testing	pytest	8.0.1	Unit test framework
HTTP Testing	httpx	0.26.0	Async HTTP client for API testing
Code Versioning	Git	-	Source code version control

Why These Choices?

- **ResNet-18 over simple CNN:** Transfer learning from ImageNet significantly reduces training time and improves accuracy. ResNet-18 is lightweight enough for fast inference while being powerful enough for binary classification.
- **GHCR over Docker Hub:** No manual secret configuration — uses GitHub's built-in `GITHUB_TOKEN`. Eliminates authentication friction in CI.
- **Docker Compose over Kubernetes:** Appropriate complexity for a single-node deployment. Docker Compose is simpler to set up and debug while still demonstrating the containerization and CD concepts.
- **MLflow over Neptune:** Fully open-source, self-hosted, no API key required. MLflow's `pytorch.log_model()` provides model registry capabilities out of the box.
- **FastAPI over Flask:** Async-first, automatic OpenAPI/Swagger docs, Pydantic validation, and significantly better performance for concurrent inference requests.

Deliverables Checklist

#	Deliverable	Status
1	Git repository with full source code	☑

#	Deliverable	Status
2	DVC configuration (`dvc.yaml`, `dvc.lock`, `params.yaml`)	☑
3	Trained model artifact (`models/cat_dog_model.pt`)	☑
4	MLflow experiment logs with confusion matrix + loss curves	☑
5	FastAPI inference service (`app/`)	☑
6	Pinned `requirements.txt`	☑
7	Multi-stage `Dockerfile`	☑
8	Unit tests (preprocessing + inference)	☑
9	GitHub Actions CI/CD workflow	☑
10	Docker image pushed to GHCR	☑
11	`deployment/docker-compose.yml`	☑
12	Smoke test script	☑
13	Prometheus metrics (`/metrics` endpoint)	☑
14	Structured JSON request logging	☑
15	Post-deployment batch simulation with true labels	☑
16	ZIP of all source code and artifacts	☐ (to be created)
17	Screen recording (< 5 minutes)	☐ (to be recorded)