DATA SCIENCE & ML

Practical MLOps: From Notebook to Production

Packaging, deployment, monitoring and retraining that lasts

Houssam Kodad

PDF · DATAFORGE BOOKS

© 2026 DataForge Books. All rights reserved.

“Practical MLOps: From Notebook to Production” and this sample are published by DataForge Books, operated by Houssam Kodad, France. The author asserts the moral right to be identified as the author of this work.

This document is a free promotional sample containing the opening chapter of the full title. It is provided for evaluation only. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher, except as permitted by applicable copyright law.

The information in this book is provided on an “as is” basis for general educational purposes. While every effort has been made to ensure accuracy, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Questions about this sample or the full edition: **support@dataforgebooks.com**

CONTENTS

Table of Contents

CHAPTER 01

What MLOps Is Really Solving

A model that wins offline is worth nothing until it serves reliably and keeps working. The leaderboard score is the start of the job, not the end of it. Everything that happens afterward — packaging, deployment, monitoring, retraining — is where most of the value, and almost all of the failures, actually live. MLOps is the discipline of doing that well, repeatedly, without heroics.

This chapter frames what MLOps is really solving: the ongoing maintenance cost of a model in production, which is unlike anything in ordinary software. We look at why models decay even when no code changes, the maturity levels teams pass through on the way to automation, and why you should think of a model as one replaceable component in a larger system rather than as a finished artefact.

Adopt that systems view and the practices in this book stop feeling like bureaucracy and start feeling like basic engineering hygiene. A deployed model is a liability that must be monitored, versioned and rolled back like any other production dependency — and treating it that way is what lets a small team operate many models without being woken at night by every one of them.

Why Models Decay

Software, once correct, tends to stay correct until someone changes it. Models are different, and the difference is the heart of MLOps. The world a model was trained on keeps moving: customer behaviour shifts, a competitor enters the market, an upstream feature changes meaning, a seasonal pattern turns. The model does not change, but the data flowing through it does, and its accuracy quietly erodes.

This is why a deployed model is a liability rather than a finished asset. It needs monitoring the way a bridge needs inspection — not because it was built wrong, but because the conditions around it change. The entire discipline of MLOps exists to make detecting and responding to that decay cheap, routine and observable, instead of a series of unpleasant surprises discovered by angry stakeholders.

A System View of ML

The model is a small box in a large diagram. Around it sit feature pipelines that feed it, a serving layer that exposes it, a monitoring system that watches it, a model registry that versions it, and a data flywheel that turns predictions and outcomes back into training data. The famous observation that ML systems are mostly not-ML code is exactly right, and it is liberating: most of your reliability

problems are ordinary engineering problems.

Adopting this view reframes everything. Most production failures happen in the wiring between the boxes — a feature that drifted, a serving contract that changed, a pipeline that silently stopped — not inside the model's mathematics. Once you see the model as one replaceable component in an operated system, the rest of this book reads as a tour of how to make each connection robust.

```
# Treat a model's inputs and outputs as versioned contracts.
@dataclass
class PredictionRequest:
    user_id: str
    features: dict[str, float]

@dataclass
class PredictionResponse:
    score: float
    model_version: str    # always know which model produced a prediction
```

The Maturity Levels

Teams progress through recognisable stages, and knowing where you sit tells you what to build next. At level zero, everything is manual: a notebook trains a model and someone copies a file to a server. At the next level, training and deployment are automated and reproducible, so a model can be rebuilt and redeployed without human steps. At the highest level, monitoring detects drift, triggers retraining, and validates the new model before it replaces the old — a closed loop with humans on the decisions that matter.

The point of the ladder is to climb it deliberately. There is no value in building automated retraining if you cannot yet reproduce a single training run, and no value in elaborate serving infrastructure for a model nobody monitors. This book is structured to move you up the levels in order, each chapter adding the capability that the next one depends on.

Reproducible Training

Reproducibility is the foundation everything else rests on. If you cannot rebuild the exact model you have in production — same data, same code, same dependencies, same result — then you cannot debug it, audit it, or safely improve it. Achieving this means versioning not just code but data, pinning the environment, and removing the hidden sources of nondeterminism that creep into ML pipelines.

In practice this is unglamorous and decisive. Snapshot or version the training data so a run can be repeated months later. Pin library versions and random seeds. Record the exact configuration that produced each model. The payoff is enormous: every later capability — experiment tracking, automated retraining, rollback — assumes you can reproduce a run, and teams that skip this step pay for it indefinitely.

Experiment Tracking and the Registry

Once training is reproducible, you need to remember what you tried. Experiment tracking records each run's parameters, metrics, and artefacts so you can compare approaches honestly instead of relying on memory and screenshots. It turns model development from a series of forgotten notebooks into a searchable history you can reason about and defend.

The model registry is the next link: a catalogue of trained models with versions, stages and approval status. It is the bridge between experimentation and production — the place a model is promoted from candidate to staging to production, with a record of who approved it and why. Together, tracking and registry give you the lineage that makes a model auditable, which matters more every year as ML enters regulated decisions.

```
import mlflow

with mlflow.start_run():
    mlflow.log_params({"max_depth": 6, "lr": 0.03})
    model = train(X, y)
    mlflow.log_metric("val_auc", evaluate(model, X_val, y_val))
    mlflow.sklearn.log_model(model, "model", registered_model_name="churn")
```

Monitoring in the Wild

A model in production must be watched on several axes at once. Data drift asks whether the inputs look like the data the model was trained on. Prediction drift asks whether the distribution of outputs has shifted. And, where ground truth eventually arrives, performance monitoring asks the only question that ultimately matters: is the model still accurate? Each catches a different kind of failure, and you need all three.

The complication unique to ML monitoring is delayed feedback. You often learn whether today's predictions were correct only days or weeks later, when outcomes materialise. So drift detection on the inputs becomes an early-warning system for problems you cannot yet confirm with accuracy. Designing monitoring around that delay — leading indicators now, confirmation later — is a theme

we develop in depth.

Safe Rollouts and Retraining

Replacing a model in production is as risky as any deployment, and the same safety patterns apply. Shadow deployments run the new model alongside the old without acting on its output, so you can compare them on live traffic. Canary releases route a small slice of traffic to the new model first. And automatic rollback returns to the previous version the moment a guardrail metric degrades.

Retraining closes the loop, but it must be disciplined rather than reflexive. A retraining pipeline needs clear triggers, validation gates that compare the challenger against the current champion on held-out data, and a human approval where the stakes warrant it. The rest of this book builds these capabilities one at a time, until you have a system that keeps models healthy with minimal heroics — which is the whole promise of MLOps.

This is a free sample

You've reached the end of the sample chapter.

Get the complete book — every chapter, fully worked — at **dataforgebooks.com**.

FULL EDITION · 312 PAGES · PDF

Read the full title at **dataforgebooks.com**

Questions? support@dataforgebooks.com