The MLOps Lifecycle

The set of practices for running machine-learning models reliably in the real world, think 'DevOps for AI'. The MLOps lifecycle takes a model from raw data, through training and deployment, to ongoing monitoring, and then loops back to retrain it as the world changes. The goal is to make ML systems repeatable and dependable, not one-off science experiments.

Best for: Productionising AI · ML reliability · Scaling models · Avoiding model decay

01 — AT A GLANCE

Models move through a continuous cycle: prepare data, train, deploy, monitor, and retrain as things drift.

1 **Prepare data**
Collect, clean and organise the data, the foundation. Bad data in means a bad model out.

▼

2 **Train & evaluate**
Build, train and test candidate models, then pick the best one for the job.

▼

3 **Deploy**
Put the chosen model into production so it makes real predictions for users.

▼

4 **Monitor & retrain**
Watch for performance 'drift'; when it slips, retrain on fresh data and redeploy.

▼

↻ *the world keeps changing, so the cycle repeats: monitor, retrain, redeploy, continuously.*

02 — THE CORE COMPONENTS

MLOps connects the stages of a model's life into one repeatable loop.

Data Preparation	Gathering, cleaning and organising data. It's the foundation of everything, and where many ML projects quietly go wrong.

Train & Deploy	Developing and testing models, then deploying the best one to production, ideally with automation so releases are fast and repeatable.
Monitor for Drift	Once live, models decay as real-world data 'drifts' away from training data. Continuous monitoring catches slipping performance early.
Continuous Retraining	When monitoring flags drift (or new data arrives), the model is retrained and redeployed, closing the loop and keeping it accurate.

A model is never 'done'

Unlike normal software, an ML model gets worse over time even if you don't touch it, because the world drifts away from the data it learned on. That's why MLOps is a loop, not a line: deploy, watch, retrain, repeat. Treat models as living systems that need monitoring and care, or they quietly rot in production.

03 — WHEN TO USE IT & WHEN NOT TO

This framework isn't right for every situation. Used at the wrong moment, it can point you in the wrong direction.

✓ USE IT WHEN…	✗ DON'T USE WHEN…
• Moving ML models from experiments into production • Running many models reliably and repeatably • Catching and fixing model 'drift' over time • Scaling AI without it becoming chaos	• For a one-off, throwaway analysis (full MLOps is overkill) • Before you even have a working model to deploy • As tools alone, it's also practices and teamwork • Skipping monitoring, the part that prevents silent failure

04 — IN ACTION

CartIQ — Why the Model Got Worse

CartIQ built a great product-recommendation model, but months later sales from it dropped. MLOps reveals why, and fixes it for good.

How the framework was applied:

Step 1	**The mystery —** The once-accurate model is now recommending poorly, though nobody changed its code.

Step 2	**Spot the drift —** Monitoring shows shopping habits shifted (new trends, seasons); the model's training data is stale.
Step 3	**Retrain on fresh data —** CartIQ retrains the model on recent behaviour, restoring its accuracy.
Step 4	**Automate the loop —** It sets up monitoring and scheduled retraining so drift is caught and fixed automatically.
Step 5	**Stay sharp —** Now the model updates itself as habits change, instead of silently rotting.

The numbers at a glance

Problem	Detect	Fix	Loop
Drift	**Monitor**	**Retrain**	**Automated**
world changed	caught early	fresh data	stays sharp

Why the loop saved the model

CartIQ's model didn't break, the world moved on, and its training data went stale. MLOps monitoring caught the drift and retraining restored accuracy, then automation kept it fresh. Treating the model as a living loop, not a finished product, is what keeps AI working in the real world.

05 — KEEP IN MIND

Even experienced teams slip up here. Keep these in mind to keep your analysis honest and useful.

1. Treating a deployed model as 'finished' and never monitoring it.
2. Ignoring data drift until performance has badly slipped.
3. Skipping automation, so retraining is slow and manual.
4. Letting messy data pipelines undermine the whole lifecycle.
5. Buying MLOps tools but not changing how teams actually work.