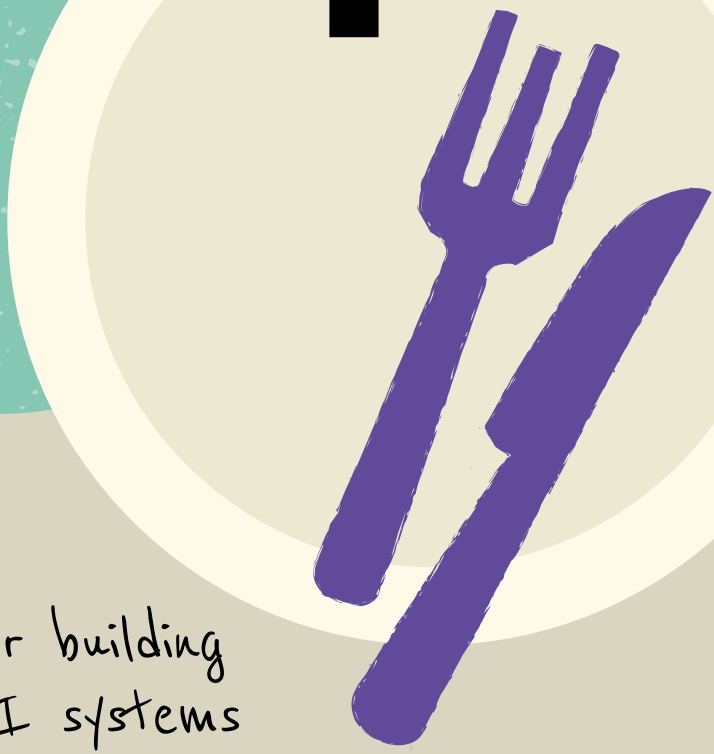
fuzzy labs

COOKING WITH MLOps

First Edition



Recipes for building delicious AI systems

Contents

Welcome to the flavourful world of Fuzzy Labs

This ‘cookbook’ is our attempt to talk about our working process in a way that’s a little more fun, and easier to digest.

We’ve borrowed the language of gastronomy because it captures something important about how well-engineered AI systems are actually built: preparation matters, experimentation is normal, and small changes can lead to deliciously effective outcomes, if everything is cooked properly.

Each ‘recipe’ in this book reflects real work we’ve done. You’ll see familiar ingredients, a clear method, and the occasional serving suggestion. You don’t need to follow every recipe exactly. Our aim is simply to show you why each one works – with repeatable results you’d be happy to serve time and again.

Now, make yourself comfortable, because some of these metaphors are going to be hard to swallow.

Who We Are & What We Do

We're Fuzzy Labs, a Manchester-based MLOps consultancy founded in 2019. We're engineers at heart, and nerds that are passionate about the power of open source.

Our work focuses on helping organisations build and productionise AI systems that have a positive impact in the real world. That means models that can be trained, evaluated, replicated and improved without guesswork, and without locking teams into proprietary platforms they don't control.

We work side by side with our clients' engineers – not simply handing over finished dishes, but cooking together. The aim is always the same: upskill teams, transfer knowledge, all whilst creating systems that are well understood, well documented and ready to be adapted long after we've stepped away from the stove.

WTF Is MLOps?

MLOps is the engineering discipline for building and running AI systems. To extend our culinary metaphor, if machine learning is the act of cooking, then MLOps is the kitchen.

It's everything that sits around the model itself: how data is prepared and stored, how experiments are tracked, how models are trained, evaluated, deployed, monitored, and improved over time. It's the environment that makes the work reliable and repeatable, by different people, under changing conditions.

Treating MLOps as a starting point, rather than an afterthought, helps avoid a familiar problem in AI work: reaching a promising result that can't go any further because the foundations were never put in place. You might get something working once, but you won't know exactly why – and you may be unable to reproduce it.

Good MLOps gives you clean surfaces, labelled cupboards, working timers, and a layout that makes sense. It doesn't guarantee a great dish, but without it, even the best recipes can leave a bitter taste when the heat is turned up.

Our Chefs

Meet the Team

Meet the team of talented chefs, dedicated to dishing up all manner of tasty treats at Fuzzy Labs.

Led by **Matt Squire** (CTO) and **Tom Stockton** (CEO), our brigade spend their days helping organisations build and productionise AI systems using open-source tools.

Writing this MLOps cookbook has been a labour of love. It's our way of passing on the recipes we reach for most often, in the hope they're as useful in your kitchen as they've been in ours.

The Fuzzy Brigade

The chefs who came together to cook up this book:

Matt Squire, Tom Stockton,
Danny Wood, James Stringer,
Misha Lakovlev, Rhiannon Kelly,
Oscar Wong, Alan Thomas

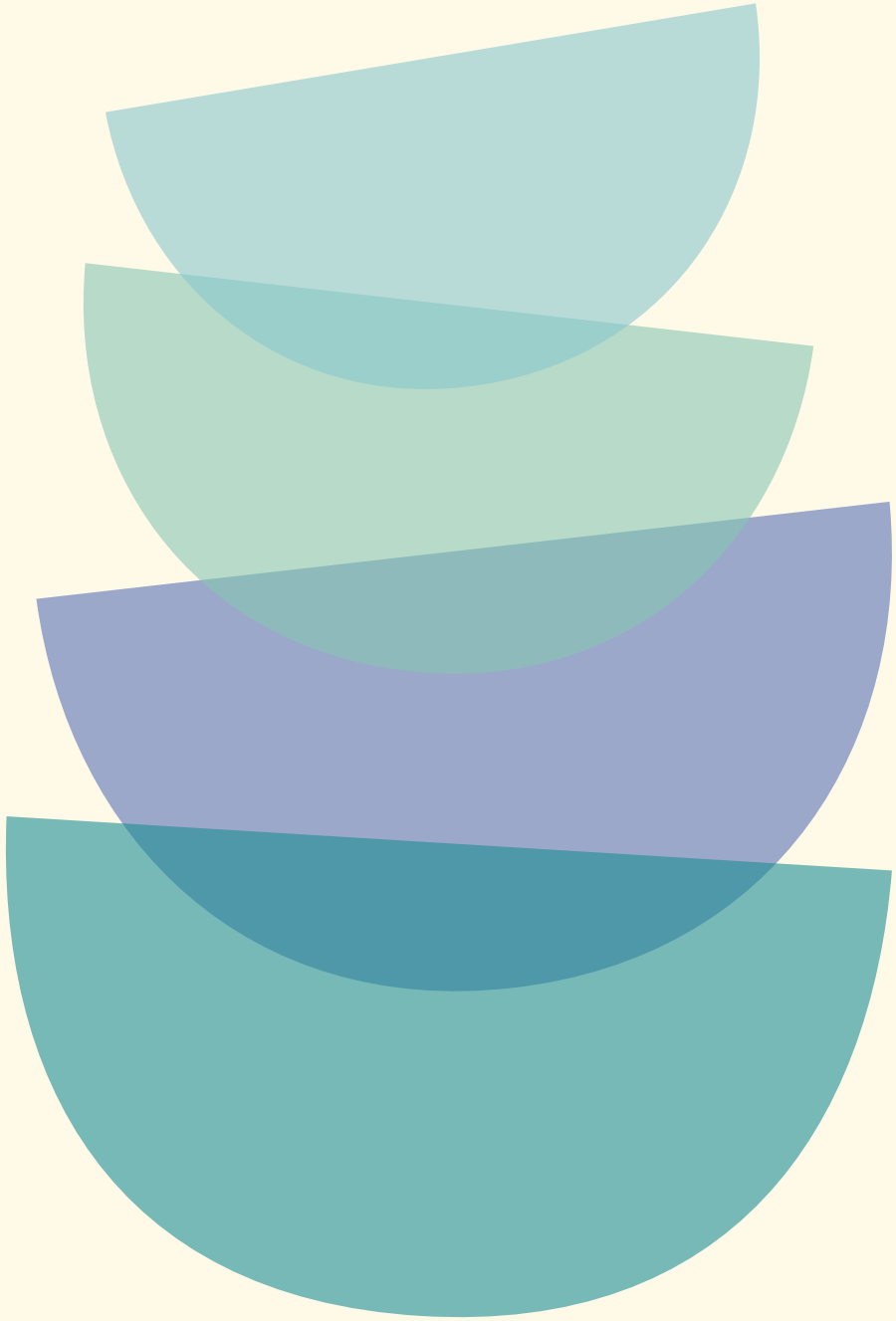
How to Use This Book

Technically this cookbook began in 2023 when, after encountering the same engineering patterns across client projects, we started recording what was repeatable and reusable. From there, we created a set of instructions we named ‘recipes’.

Each recipe is a practical template for a specific AI use case, built on solid MLOps foundations and open-source tools. You can run each as it is to help understand fundamental MLOps principles, or you can adapt them to work with your own AI use case.

We have strong opinions about what good MLOps looks like – how data should be handled, how experiments should be tracked, and how models should make the journey from prototype to production. This book isn’t the full menu, but it should give you a good sense of our approach. Think of it as an amuse-bouche: with six carefully prepared recipes that show the principles at work.

Finished ‘dishes’ (real-world case studies) can be found later in the book under *Here’s One We Made Earlier*, so you can see how different recipes come together in practice.



The Ingredients

(Open-Source Tools)

Like any kitchen, there are certain ingredients in the pantry that get used time and time again. As mainstays of many a recipe, the open-source tools we turn to most frequently are listed here.

Data Version Control

DVC (Data Version Control) keeps track of the data, models, and pipelines behind a machine learning project. It links them back to your Git repository so you can see exactly what changed, when, and why. It's a good fit for smaller teams and earlier-stage projects, where making explicit, reversible decisions about data matters more than heavy automation.

lakeFS brings Git-like operations to large data lakes. It lets you branch, commit, and roll back changes to data stored in object stores, without tying everything directly to Git. It comes into its own once datasets grow to gigabyte or terabyte scale, and teams need production-grade guarantees around data consistency and lineage.

LabelBox isn't a version control tool, but it plays a crucial supporting role. It provides a collaborative way to label data, which is often the most manual and error-prone part of the pipeline. Used alongside data versioning, it helps ensure training data can be reproduced and understood over time.

Experiment Tracking

MLflow records what happened during each experiment run, including parameters, metrics, and artefacts. That makes it possible to compare results properly, rather than relying on memory or scattered notes. It's also useful for versioning and tracking LLM prompts, where the effect of small changes can be difficult to predict.

Model Training

ZenML provides a structured way to define and run machine learning pipelines. It separates pipeline logic from the infrastructure used to execute it, so the same workflow can run locally or in the cloud. It's most useful when you want training pipelines to stay portable, reproducible, and easy to adapt as systems mature.

Optuna handles hyperparameter optimisation by running controlled experiments to find settings that actually improve performance. It's lightweight to adopt, scales from local experiments to distributed runs, and makes it easier to understand which changes helped and which didn't.

Brevitas supports quantisation-aware training in PyTorch, allowing models to be trained with constrained hardware in mind. It's particularly useful when memory, power consumption, or inference latency are hard limits, for example when deploying models to edge devices.

The Ingredients, continued...

Model Deployment

KServe provides a Kubernetes-native way to serve machine learning models. It standardises how models are deployed, scaled, and exposed for inference. It's a solid choice when you want a consistent deployment pattern across different training frameworks.

vLLM is an efficient inference engine for large language models, designed to deliver high throughput with predictable memory usage. It's well suited to self-hosting LLMs where low latency and stable performance on shared GPU infrastructure matter.

Ray Serve sits in front of inference services to handle request routing, autoscaling, and fault tolerance. It's particularly useful for GenAI systems with uneven traffic patterns, where demand can spike unpredictably.

PlatformIO is a build and deployment toolchain for embedded systems. We use it when models need to run on microcontrollers or other constrained devices, where cloud-based serving isn't an option.

Model Monitoring and Evaluation

Evidently monitors deployed models for data drift, concept drift, and performance degradation. It supports both traditional machine learning and LLM deployments, with ready-made metrics and dashboards to surface problems early.

Pydantic Logfire (commercial service; open-source client libraries) Logfire provides observability into application and agent behaviour, capturing structured traces of execution paths, tool calls, and decisions. While the backend service is commercial, it builds on open-source client libraries and integrates cleanly into existing systems.

Pydantic Evals is an open-source framework for evaluating agent and LLM behaviour. It supports repeatable checks so changes don't silently alter how a system behaves.

RAGAS evaluates Retrieval-Augmented Generation systems by measuring retrieval quality and answer faithfulness. It helps teams understand whether responses are grounded in the right source material, rather than sounding plausible but being wrong.

Tools that sit outside the classic MLOps categories

Guardrails AI constrains model outputs by enforcing formatting and content rules. The open-source core covers common safety and consistency needs, with optional commercial features for more advanced governance.

IBM MCP ContextForge acts as an agent gateway, enforcing governance over tool usage and capturing a full audit trail of agent actions. Its plugin system allows teams to validate inputs and outputs, apply policies, and export telemetry, making agent behaviour observable and controllable as systems move towards production.

Recipe One

Experiments in Computer Vision

Theme

AI Research Engineering

Purpose

Cook-up a custom computer vision model from scratch.

URL: fuzzylabs.ai/recipes/r1

Ingredients

- **DVC** – Data Version Control (your pantry and labelling system)
- **MLflow** – Experiment Tracking (your tasting notebook)
- **ZenML** – Pipeline Automation (the prep line)

Chef's Notes

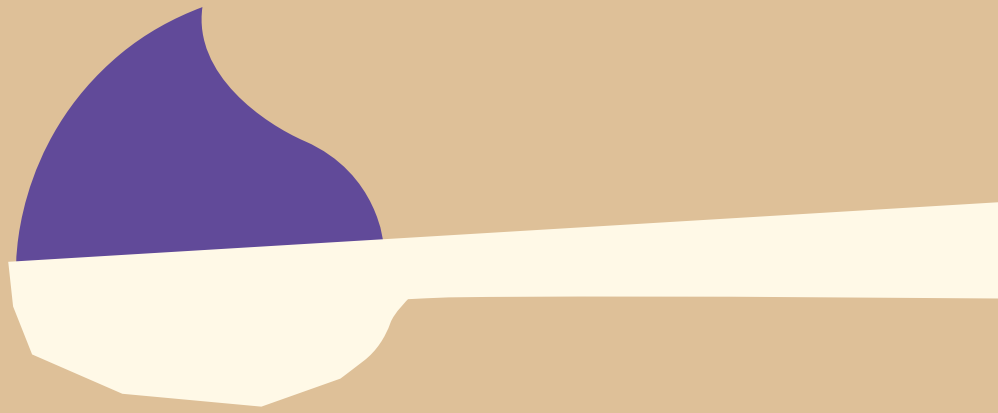
This recipe demonstrates the MLOps tools and techniques required to train a custom computer vision model. We'll be using the Simpsons-MNIST dataset: small, blurry images that are deliberately awkward to classify. Our aim is to train a model that can tell Bart from Lisa from Marge from Maggie.

Training models without structure is like cooking from memory. You might stumble onto something good, but you won't know why – and you probably won't be able to reproduce it. This recipe is designed to remove friction from the model-training process and make every step repeatable.

You'll be able to see exactly what went into the model, and reliably produce the same result again. It also removes the need to manually run each step of training – a process that can be time-consuming and error-prone. To extend our cooking analogy, if there's a tool designed to chop, measure and prepare everything consistently, you wouldn't insist on doing it all by hand, would you?

Here, we're creating a perfect test kitchen for learning how to build, adjust and improve a model without losing track of what changed. We'll do this by demonstrating the MLOps principles of data versioning, experiment tracking and building a model training pipeline.

The specifics of how we do this will be applicable for training other kinds of computer vision models but the general principle also applies more broadly for training other types of models.



Method

1. **Preparation**
   Download the Simpsons image dataset and prepare a test/train set.

2. **Keep a clear record of every batch**
   Version the data using DVC, so every change is tracked and reversible.

3. **Get cooking**
   Run the ZenML training pipeline:
   - Load the data
   - Run the training code
   - Log details of training run into MLflow
   - Evaluate the model
   - Season to taste

At this point, you're cooking on gas (trained one model) and have a solid base recipe (recorded one experiment as a baseline).

Now, your job is to improve the model. Add seasoning by trying rotations, or add layers in your neural network. Each iteration goes through the same pipeline and is recorded alongside the rest.

With ZenML automating the process, and all our experiments tracked in MLflow, we can always reproduce a previous state, restoring the exact data, code and configuration that produced it.

Et voilà.

Serving Suggestions (Variations)

- Swap Simpsons-MNIST for your own dataset
- Scale the training to run on a GPU cluster
- Use the same method to train a language model instead of vision

Recipe Two

Fashion on the Edge

Theme

AI Prototype

URL: fuzzylabs.ai/recipes/r2

Purpose

Train and deploy a computer vision model to an edge device that can identify items of clothing.

Ingredients

- **Optuna** – hyperparameter optimisation (the sous-chef running controlled taste tests)
- **PyTorch** – model training (the camping stove)
- **Brevitas** – quantisation-aware training (the portioning knife)
- **MLflow** – experiment tracking (your tasting notebook)
- **ZenML** – pipeline automation (the prep line)
- **PlatformIO** – device deployment (your mess tin)

Chef's Notes

This recipe shows how to train and deploy a computer vision model that can run on constrained hardware. We shrink the model using quantisation-aware training and use automated experiments to find a workable balance between model size and accuracy.

Running models on edge devices is more like cooking on a camping trip than in a restaurant kitchen. You can only carry so much, power is limited, and every ingredient has to earn its place. The job isn't to make the fanciest dish – it's to make something that fits in the rucksack, cooks reliably, and doesn't have your stove running out of gas halfway through.

Here we use the Fashion-MNIST dataset to train a classifier that runs on an Arduino-compatible ESP32-S3, identifying items such as t-shirts, pullovers, and dresses directly on the device.

Along the way, we demonstrate MLOps principles such as structured experiment tracking, automated pipelines, and repeatable deployment to the ESP32.

The benefits of running a model on the edge are improved privacy (data stays on-device) and lower latency compared with sending data to the cloud.

Method

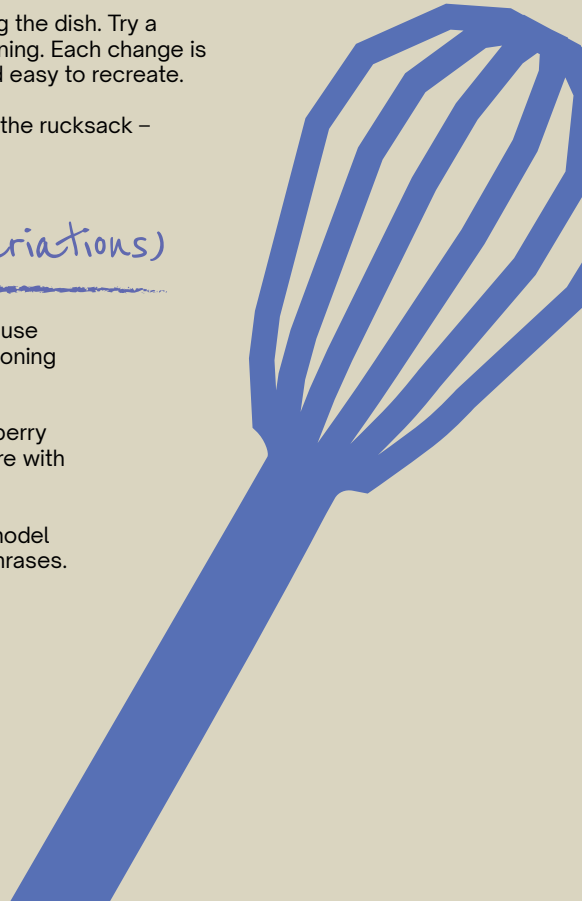
1. **Preparation**
   Download and preprocess the Fashion-MNIST dataset for training.
2. **Set the menu**
   Use Optuna to orchestrate multiple ZenML pipeline runs with different hyperparameters, searching for the best balance between accuracy and model size.
3. **Get cooking**
   Each pipeline run:
   - Trains a model using Brevitas to enable quantisation-aware training
   - Logs training parameters and metrics to MLflow
   - Evaluates model performance consistently
4. **Pick the plate**
   Review experiment results and select the configuration that best fits the hardware constraints.
5. **Portion it down**
   Convert the trained model into a quantised format suitable for the ESP32-S3.
6. **And serve**
   Use PlatformIO to deploy the model onto the ESP32-S3 for on-device inference.

With everything laid out, you can start refining the dish. Try a smaller model. Add a layer. Adjust the seasoning. Each change is written down, compared against the rest, and easy to recreate.

You're no longer guessing what made it fit in the rucksack – you've got a recipe that travels.

Serving Suggestions (Variations)

- Train on a custom dataset specific to your use case, combining this recipe with data versioning from Recipe #1.
- Target more powerful hardware (e.g. Raspberry Pi) and switch to a larger model architecture with higher-resolution images.
- Adapt the pipeline for audio by training a model that recognises short sounds or spoken phrases.



Recipe Three

Part 1

Self-Hosted Agent: Prototype

Theme

AI Prototype

Purpose

Build and deploy self-hosted AI agent to review GitHub Pull Requests.

URL: fuzzylabs.ai/recipes/r3

Ingredients

- **Pydantic AI** – Agent framework (the mixing bowl)
- **MLflow** – Prompt versioning (your tasting notebook)
- **vLLM** – Model serving (the hob)
- **Pydantic Evals** – Agent evaluation (the tasting spoon)
- **Pydantic Logfire** – Agent observability (the thermometer)

Chef's Notes

This recipe shows how to self-host an AI agent – with a Large Language Model (LLM) at its core – without relying on proprietary software or sending any data to third parties.

The agentic use case we're preparing here is a GitHub Pull Request reviewer, similar to Cursor's 'bugbot' feature. It's a dish that excites, so it's no surprise that it takes time and attention to execute in the kitchen: inspecting each change, judging how it fits with what's already there, and offering clear, useful notes, consistently.

Along the way, we demonstrate key MLOps principles, including experiment tracking by versioning LLM prompts, serving the model reliably, and evaluating the agent's output before it's ready to go. Each step is measured, repeatable, and designed to avoid guesswork. It's like baking a cake, but with scales, timers, and a written recipe, rather than winging it with your inner Mary Berry.

Why go to the effort of self-hosting? Because relying on closed platforms is like cooking with pre-packaged ingredients. It's quick, but you lose visibility and control. By hosting your own agent, you avoid proprietary lock-in, keep data within your own network, and retain full control over model choice and cost, adjusting the recipe as requirements change.

Method

This example is hosted in AWS, but we use Kubernetes (k8s) to host the agent and model which makes it easy to modify this recipe to run in any cloud or on prem.

1. **Prepare the base**
   Install MLflow into k8s and register the initial system prompt for the agent. This defines the behaviour of our agent and tells it which 'tools' it can use to do its job as a PR reviewer.

2. **Heat the model**
   Deploy the vLLM server into k8s. This defaults to using the Qwen model and creates an API interface for us to call the model from the agent.

3. **Stir carefully**
   Deploy the agent into k8s. Our open-source agent uses the Pydantic AI framework to load our system prompt, integrate with the LLM and define the tools and workflow needed to communicate with GitHub to review our PRs.

4. **Taste and test**
   Evaluate the agent's response using Pydantic Eval end-to-end tests using a third party LLM 'As-A-Judge'. This tells us if it performs as expected against our known test cases.

5. **Plate up**
   Deploy the Pydantic Logfire backend into k8s and enable logging of our agents' activities. This gives us visibility into how the agent performs in real use.

With everything in place, you can start iterating. Tweak the system prompt. Swap the model. Adjust the tools. Each change is versioned, evaluated and observable. You're no longer guessing which ingredient improved the dish – you know exactly.

And there you have it: something tasty, cooked from scratch and entirely in-house.

Serving Suggestions (Variations)

- Add new tools so the agent can draft suggested fixes alongside its review.
- Swap the Qwen LLM for DeepSeek, Llama, Mistral, or a third-party model if self-hosting isn't a requirement.
- Change the use case entirely – rework the tools and prompt so the agent connects to your CRM and answers questions about sales activity and pipeline.

Recipe Four

Part 2

Self-Hosted Agent: Production Governance

Theme

Production AI

Purpose

Add production features of governance and provenance to the self-hosted agent prototype.

URL: fuzzylabs.ai/recipes/r4

Ingredients

- **IBM MCP ContextForge** – agent gateway (the pass)
- **Schema Guard (ContextForge plugin)** – agent governance (the instruction sheet)
- **Telemetry Exporter (ContextForge plugin)** – agent provenance (food safety)

Chef's Notes

Part 1 gives you a working self-hosted agent for reviewing GitHub pull requests. Part 2 is about being able to trust what it does.

Once an agent is allowed to call tools automatically, the questions change. It's no longer just about whether the result looks right, but about what happened along the way. In Bake Off terms, you've moved out of your home kitchen and into the tent. The rules are clear, the process is visible, and you're expected to explain your reasoning for the showstopper, not just hand over the finished bake.

This recipe adds a governance layer in front of the agent's tool calls. Instead of invoking external tools directly, the agent routes all tool usage through ContextForge. That gives you a single place to enforce policies and capture a complete audit trail of tool usage.

With governance and provenance in place, you can constrain how the agent behaves, inspect its actions after the fact, and adjust policies without changing the agent itself. You move from hoping the agent behaves to being able to prove that it did.

No more worrying about what you've pulled from the oven. The bake still stands up when it's cut.

No soggy bottoms.

Method

Our aim is to add governance without breaking the existing agent workflow or degrading output quality.

1. **Line the tin**
   Install ContextForge in the same Kubernetes cluster as the agent. Configure the agent to send all tool calls through ContextForge.

2. **Adhere to food safety**
   Add Schema Guard rules for the tools used by the agent. If the agent attempts to call an unauthorised GitHub API or internal tool, the request is blocked at the gateway.

3. **Control the mixture**
   In addition to limiting which tools can be called, we also define rules around the results the agent is allowed to receive. This helps prevent unsafe or unexpected outputs from propagating back into the agent's reasoning.

4. **Keep your baking notes**
   Enable the OpenTelemetry exporter to capture tool inputs, outputs, timing, and error status. This gets sent to the existing Pydantic Logfire backend to enhance our observability.

5. **Test the bake**
   Run the existing evaluation suite and a live PR review to confirm that governance policies constrain how tools are used, without changing the quality of the agent's reviews.

With this layer in place, you can prove what the agent did, keep its behaviour within bounds, and evolve governance policies independently of the core agent logic – even when the bake is under scrutiny.

Serving Suggestions (Variations)

- Require manual approval for any tool that writes to GitHub, while allowing read-only tools to run automatically.
- Introduce rate limits to keep tool usage predictable and prevent runaway costs or abuse as more users adopt the agent.
- Extend Schema Guard to enforce business rules, such as blocking comments that reference internal customer names.

Recipe Five

RAG: Part 1 Prototype

Theme

AI Prototype

URL: fuzzylabs.ai/recipes/r5

Ingredients

- **Metaflow** – data pipeline (your prep kitchen)
- **Chroma** – vector database (the pantry)
- **MLflow** – experiment tracking and prompt versioning (your tasting notebook)
- **vLLM** – model serving (the hob)
- **RAGAS** – evaluation (the tasting spoon)
- **Guardrails AI** – hallucination and jailbreak checks (food safety)

Purpose

Build a Retrieval-Augmented Generation (RAG) prototype on Kubernetes using a real-world financial dataset.

Chef's Notes

This recipe shows how to cook up an open-source RAG system. RAG is a way to make AI answers more grounded in real documents. Instead of cooking from scratch, the system works from what's already prepared – pulling the right material together and answering using sources it can point back to.

The approach here is closer to Great British Menu than Ready, Steady, Cook. The documents are known in advance, the choices are deliberate, and you can explain why each part belongs on the plate.

Along the way, we demonstrate four core MLOps principles including data pipelines for consistent document ingestion and indexing, experiment tracking to record prompt and retrieval settings, evaluation to measure retrieval quality and answer faithfulness and inference to serve the system as an API.

The result is a repeatable workflow you can confidently extend in the follow up recipe (Part 2), where we focus on production hardening – tightening controls, not rewriting the recipe.

Method

This example is hosted in AWS, but we use Kubernetes (k8s) to host the agent and model which makes it easy to modify this recipe to run in any cloud or on prem.

1. **Prepare**
   Use Metaflow to pull 50 FinDER documents and prepare them for downstream processing. This gives us a realistic set of financial filings to work with, rather than a toy dataset.
2. **Section it up**
   Split the documents into smaller sections and store them in a vector database (Chroma) so they can be searched semantically. Decisions made here – how large the chunks we chop the text into are and how they're structured – shape how the system behaves later.
3. **Decide what goes in the pan**
   For each user query, retrieve candidate sections and re-rank them so the most relevant context rises to the top.
4. **Put it together**
   Serve the Qwen model using vLLM. The model reads the retrieved context and produces an answer with citations back to the source documents.
5. **Taste, then taste again**
   Use RAGAS to score retrieval and answer quality. Apply Guardrails AI to detect hallucinations and common jailbreak attempts.
6. **Make notes**
   Log prompts, retrieval settings, and evaluation results in MLflow so changes are explicit and comparable over time.

With everything laid out, you can refine the dish with confidence – adjust chunk sizes, embeddings, and prompts – knowing exactly what changed and why. The result is a solid base you can take all the way to the banquet, rather than starting over with every new dish.

Service!

Serving Suggestions (Variations)

- Swap the dataset for support tickets, research papers, or internal product documentation.
- Use a different self-hosted LLM, or call out to a third-party model if hosting isn't a requirement.
- Improve retrieval by adding a query translation step to expand acronyms or company aliases (e.g. "MS Q3 margin" > "Morgan Stanley profit margin between July and September 2024").

Recipe Six

RAG: Part 2 Production

Theme

Production AI

Purpose

Productionise the RAG prototype with scalable serving, monitoring, load testing, and controlled prompt updates.

URL: fuzzylabs.ai/recipes/r6

Ingredients

- **Ray Serve** – model proxy (the pass)
- **Prometheus + Grafana –** monitoring (keeping an eye on the timings)
- **Locust –** load testing (a busy Saturday night run-through)
- **Pydantic AI** – feedback loop (diner feedback)
- **Guardrails AI** – business guardrails (house rules)
- **MLflow** – prompt versioning (tasting notebook)

Chef's Notes

Part 1 gives you a working RAG system. Part 2 is about trusting it under load and with real users.

There's a real difference between getting a dish to taste right and serving it consistently at scale. As demand increases, timing becomes visible, small changes have wider effects, and constraints that were easy to ignore before now need to hold.

This recipe takes the existing retrieval and generation logic and hardens it for production use. We focus on the things that tend to break first: scaling model inference, observing performance, testing the system under pressure, and updating prompts without changing behaviour.

Those additions let you answer practical questions that only emerge in production. How does latency change as traffic increases? What happens when users push the system in unexpected ways? Which prompt change improved answers, and which one made things worse?

Along the way, this recipe demonstrates advanced MLOps principles, including scalable model serving, monitoring, continued prompt versioning, and evaluation of output quality.

The aim isn't to change the dish. It's to make sure it can be served reliably, safely, and consistently, even when demand is out the door.

Method

Our aim is to turn the working prototype into a production ready service.

1. **Set up the pass**
   Install and configure Ray Serve in front of the existing vLLM service. This allows you to scale inference horizontally and improve reliability without changing application logic.
2. **Lay down the house rules**
   Extend Guardrails AI with domain rules. In this example, because the dataset is financial, the system is prevented from giving financial advice.
3. **Keep an eye on the timings**
   Deploy Prometheus and Grafana to collect and visualise performance and reliability metrics such as latency, error rates, and throughput.
4. **Put the kitchen through a busy service**
   Install Locust to generate traffic on demand and observe how the system behaves under stress using the monitoring stack.
5. **Listen to the diners and take notes**
   In production, user feedback becomes a signal. This agent gathers feedback on response quality and proposes prompt updates, which are versioned in MLflow rather than applied silently.
6. **The head chef signs off changes**
   Use the RAGAS evaluation suite to assess proposed prompt changes and decide whether they should be promoted to production.

With these components in place, you can keep the kitchen running under pressure. The service scales, changes can be tested safely, and improvements are made deliberately, without losing sight of what's happening or why.

Serving Suggestions (Variations)

- Replace the financial guardrails with domain-specific rules for healthcare, legal, or customer support use cases.
- Tune the Locust workload to reflect your real traffic mix and add latency thresholds to gate new releases so you don't regress performance.
- Schedule the feedback loop to propose prompt updates on a regular cadence, approving changes only after passing evaluation.

Here's One We Made Earlier

Six real-world case studies for you to feast your eyes on.

Case Study One:

Fotenix

Onboarding new polytunnels in days, not weeks

Tasting Notes:

Experiments in computer vision

Area:

AI Production

MLOps:

- Training Pipeline
- Experiment Tracking
- Data Versioning

Ingredients

- MLflow
- ZenML

URL:

fuzzylabs.ai/casestudy/c1

The Client

Fotenix helps growers monitor and manage crops using advanced imaging and AI. They specialise in indoor farming environments, where crops such as strawberries are grown under tightly controlled conditions.

Their camera systems analyse crop imagery to spot pests, detect early signs of disease, and carry out broader quality checks – flagging areas that need attention before problems spread. The challenge isn't a lack of data, but making sense of it consistently as models are trained, tuned and deployed for each new customer setup.

The Order

As Fotenix rolled out their solution to more customers, each installation required models to be trained and adapted to slightly different growing conditions. Without a consistent way to track experiments, comparing training runs and understanding what had changed from one iteration to the next became slow and manual.

It was a bit like cooking by feel or instinct: workable when you're making one dish, but harder to scale when you're feeding a roomful of guests.

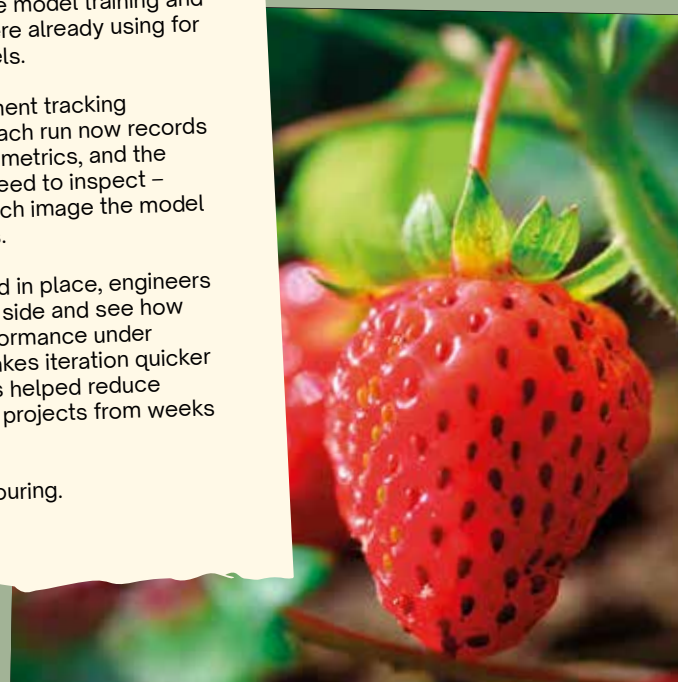
The Final Dish

We worked alongside the Fotenix engineering team to fold MLflow into the model training and evaluation pipeline they were already using for their computer vision models.

From that point on, experiment tracking happened automatically. Each run now records model settings, evaluation metrics, and the visual outputs engineers need to inspect – including the regions of each image the model focuses on during analysis.

With that consistent record in place, engineers can compare runs side by side and see how tuning choices affect performance under similar conditions. This makes iteration quicker and more reliable, and has helped reduce onboarding for new client projects from weeks to days.

All in all, a dish worth savouring.



Case Study Two:

Pebl

Enabling sustainable farming through computer vision on the edge

URL:

fuzzylabs.ai/casestudy/c2

The Client

Pebl design and supply low-cost marine monitoring kit and field support, helping teams collect usable data from coastal and offshore sites. Their work supports sustainable seaweed farming, often in remote locations where infrastructure is minimal and conditions are unpredictable.

Our job was to explore how AI could help monitor these farms automatically. Any solution would need to operate far from shore, strapped to a buoy, running on battery power, and with no live communication back to land. Like a slow cooker full of braised brisket, happily left unattended.

Tasting Notes:

Vision on the edge

Area:

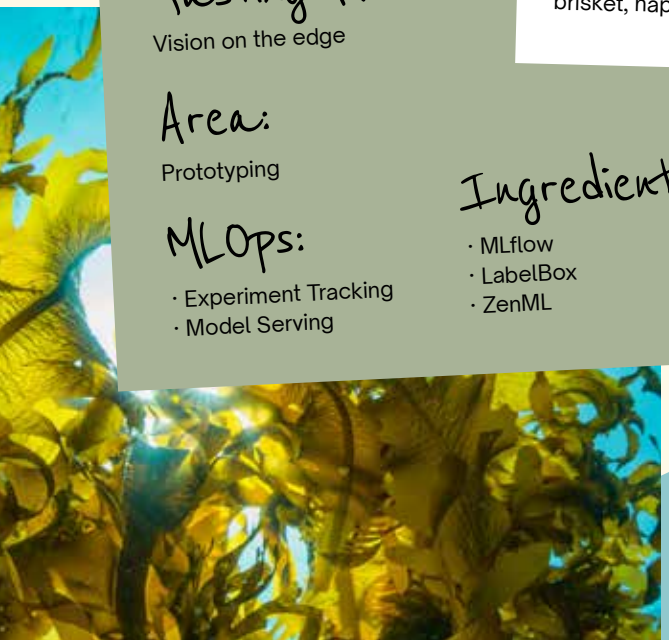
Prototyping

MLOps:

- Experiment Tracking
- Model Serving

Ingredients

- MLflow
- LabelBox
- ZenML



The Order

Monitoring activity around seaweed farms is important, particularly spotting nearby vessels that could interfere with equipment or operations. Doing this manually isn't practical, and streaming video back to shore isn't an option.

The challenge was to design a system that could watch continuously, decide when something mattered, and start documenting, all while working with limited power and compute. Everything had to be efficient, resilient, and simple enough to deploy repeatedly in the field.

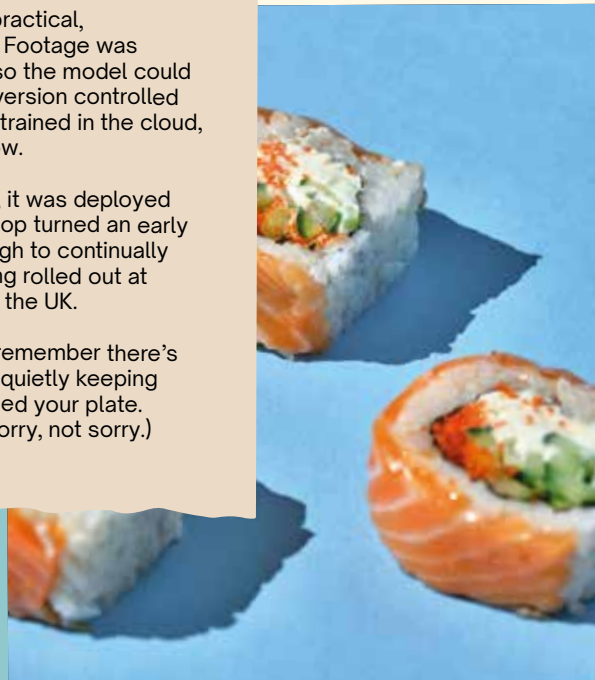
The Final Dish

On site, the setup consisted of a battery-powered Raspberry Pi fitted with a Pi camera module for capturing images. We ran a YOLOv8 computer vision model directly on the device, enabling it to recognise ships and begin recording as soon as a vessel appeared, without anyone needing to monitor the feed.

Because training models at sea isn't practical, improvement happened back on land. Footage was brought ashore, labelled in LabelBox so the model could learn what qualifies as a vessel, then version controlled using DVC. New model versions were trained in the cloud, with each experiment tracked in MLflow.

Once a stronger model was identified, it was deployed back to the device. That repeatable loop turned an early prototype into something robust enough to continually reuse. The result is a system now being rolled out at around 30 seaweed farm sites across the UK.

So, the next time you're eating sushi, remember there's a good chance a system like this was quietly keeping watch long before the seaweed reached your plate. We recommend the CalifornAI Roll. (Sorry, not sorry.)



Case Study Three:

Fuzzy Media

Productionising an AI support assistant that learns from your team

Tasting Notes:

Self-hosted agent

Area:

AI Production

MLOps:

- Data Version Control
- Experiment Tracking
- Model Serving

Ingredients

- lakeFS
- MLflow
- vLLM

URL:
fuzzylabs.ai/casestudy/c3

The Client

Fuzzy Media is a major U.S. sports broadcaster, delivering live streams, real-time stats, and breaking news to tens of millions of users each month.

When something breaks, it can be mid-game and very public. Internal engineering support is part of keeping the show on the road, but the knowledge needed to fix issues is often scattered across Slack threads, docs, and a handful of senior engineers.

The Order

To make that knowledge easier to reach, the Fuzzy Media engineering team built a prototype Slackbot called NORA. NORA's job was simple in theory. When an engineer asks a support question in Slack, she replies with the best known answer, pulling from internal sources and past expert responses.

As usage grew, the prototype started to creak. Answers were slow, coverage was patchy, and senior engineers were still being pulled into repeat questions. The challenge was to productionise NORA and level up her skills on the pass.

The Final Dish

We worked with the broadcaster's team to whip NORA into a production-ready AI support assistant that learns from how the team actually solves problems.

She was rebuilt to train on real expert replies from Slack, alongside documentation in Confluence and GitHub, so her answers reflected the team's lived knowledge rather than generic best practice. We added evaluation and monitoring so new versions could be compared safely, and performance could be tracked over time.

By the end of the project, average response time dropped from 64 minutes to 37 seconds, time to resolution improved by 24.8 percent, escalations decreased, and new hires could onboard faster with NORA as a first-stop source of help.

All in all, a bit like hotdogs at a ball game. Not fancy, absolutely relied upon, and expected to deliver quickly when the crowd is loud.

Case Study Four:

South Yorkshire Police

Building an AI assistant to halve case file prep time

URL:

fuzzylabs.ai/casestudy/c4

Tasting Notes:

RAG Prototype

Area:

AI Prototype

MLOps:

- Experiment Tracking
- Model Serving
- Model Evaluation

Ingredients

- MLflow
- RAGAS

The Client

South Yorkshire Police is one of 43 territorial police forces in England and Wales, serving around 1.4 million people.

With support from the Police Science and Technology Fund, the force wanted to explore whether an AI assistant could help reduce the administrative effort involved in preparing case files for prosecution. We were brought in to see what was possible in practice.

The Order

Preparing a case file is structured, methodical work. Evidence has to be handled in exactly the right way, every time.

As volumes increased and evidence became more digital, that preparation took longer. Much of the work requires training and care, and more importantly time, which was pulling officers away from frontline duties.

The challenge was to see whether an AI assistant could take on some of that preparation while keeping the recipe exactly the same. Any solution had to work only with existing case material, run entirely inside police systems, and leave a clear, reviewable audit trail. No mystery ingredients.

The Final Dish

We worked closely with frontline officers to build a prototype AI assistant that runs entirely within the police IT infrastructure. Rather than generating answers from scratch, it works with the material already in the case, helping officers prepare files more efficiently.

The assistant can summarise cases, build timelines, highlight inconsistencies across documents, and support statement review. Officer judgement stays central throughout.

Every interaction is logged, recording what was asked, what material was used, and which model version responded. This provides the audit trail needed for governance and review.

Early evidence suggests the prototype could halve the time officers spend on case file preparation. It is still a prototype, but the structure is there. Careful, layered, it's a bit like (dare we say it) your classic Cornetto.

fuzzy labs

Let's Cook Together

Whilst we always aim to satisfy, we hope you're still hungry for more.

This book is a first edition – a small selection of recipes that show how we think about building AI systems that last, and the kind of kitchen they need to thrive in. But there's more depth behind each one, more variations to explore, and more work we're continuing to do as tools evolve and requirements change.

If something here sparked your appetite – a recipe you'd like to try, adapt, or take further – then feel free to have a go. Better yet, give us a shout and we'd be glad to cook together.

For now, thanks for taking the time to dine with us. And if you're still peckish, you know where to find us.

Contact

fuzzylabs.ai

talk@fuzzylabs.ai

0161 533 0337

Chefs Wanted

We're always looking for people with a refined palate to join the team so go to our careers page to view the open roles.

fuzzylabs.ai/careers

Appendix

Our Core Offering

Practical AI, Properly Engineered

In 2011, Marc Andreessen declared that "software is eating the world."

He was right – but software didn't do it alone. It needed developers who could write, test and execute code quickly, and it needed DevOps to make deployment fast, reliable and mostly free from human error. Today, nobody builds production software without those DevOps principles.

Building AI systems looks a lot like building software, but it comes with extra moving parts. Models aren't just code – they're experiments, data, training runs and the unpredictable behaviour that comes with all three.

You still need DevOps. But you also need something that understands this extra layer. That something is MLOps.

Conventional MLOps thinking centres around a figure-of-eight model: gather data → experiment → train → deploy → monitor → loop back.

But we believe ***you don't need the full MLOps lifecycle unless you're actually putting AI into production.***

In an ideal world, everyone would be productionising their AI and thinking about the full application of MLOps from the very start. In reality, most organisations are still earlier on the curve – exploring use cases, trying new modelling approaches, or building prototypes to see if an idea has any value in the first place.

At those stages, you need some MLOps, but not all of it.

Appendix

Our Three Areas of Delivery

Each of our three areas of delivery solves a different type of problem. It all starts by exploring what's possible, moves to build and, if all goes well, ends with something new and exciting running reliably in the real world.

Successful projects rarely stay within one 'area' and will move between research, prototype and production as the value and opportunity becomes clearer.

What connects them all is MLOps. Not as a rigid framework, but as a set of practices that make progress possible: running experiments that can be trusted, building prototypes that won't need rewriting, collaborating without confusion, and productionising AI when the time is right.

AI Research Engineering

We call this the "is this even possible?" stage: tackling complex problems at scale by combining research with solid engineering. The work can include analysing models for security risks, rethinking data interpretation, or building predictive models such as energy demand forecasting. This isn't research for its own sake. It always leads to something tangible – a working tool, a proven model, or a production-ready MLOps pipeline.

AI Prototypes

This is where ideas become tangible. A prototype shows the potential of an approach without the weight of a full production system, giving teams something real to explore, test assumptions, and see what the idea could become. The focus is on the art of the possible: applying AI to a specific challenge in a clear, compelling way. Prototypes vary by what needs validating – from lightweight demonstrations of core behaviour to richer workflows that lay early foundations for production.

AI Production

AI Production is where models become real systems, supported by a full MLOps toolkit. We focus on traceability and repeatability through data versioning, experiment tracking, and training pipelines, alongside scalable serving and monitoring. The work ranges from taking proofs of concept into production to improving existing systems with the foundations needed to run AI with confidence.

Appendix

‘The Ramp’

What changes across the Areas of Delivery is not whether you need MLOps, but how much of it you need – and how mature those capabilities must be for the work to hold together.

As projects move from research to prototype to production, the same MLOps capabilities apply – but with increasing depth. Early work can use lighter foundations; prototypes need more structure; and production requires mature, automated versions of those same phases.

Appendix

Building the Ramp

Five Core MLOps Capabilities

1. Data Version Control

Data is everything in machine learning. It grows, changes, and is shaped by many hands – all of which affect how a model performs. Data Version Control keeps that evolution orderly. It tracks changes, records lineage, and makes large datasets shareable. Much like Git does for code, Data Version Control lets people collaborate confidently, compare states over time, and understand exactly which data produced which model.

2. Experiment Tracking

Machine learning involves a lot of experimenting. As we work to improve our models, we try different techniques, tune parameters, augment data, and examine the metrics that matter. Experiment Tracking is what allows us to compare models over time. It logs every run, every setting, every outcome, and enables us to collaborate more easily with colleagues.

3. Training Pipelines

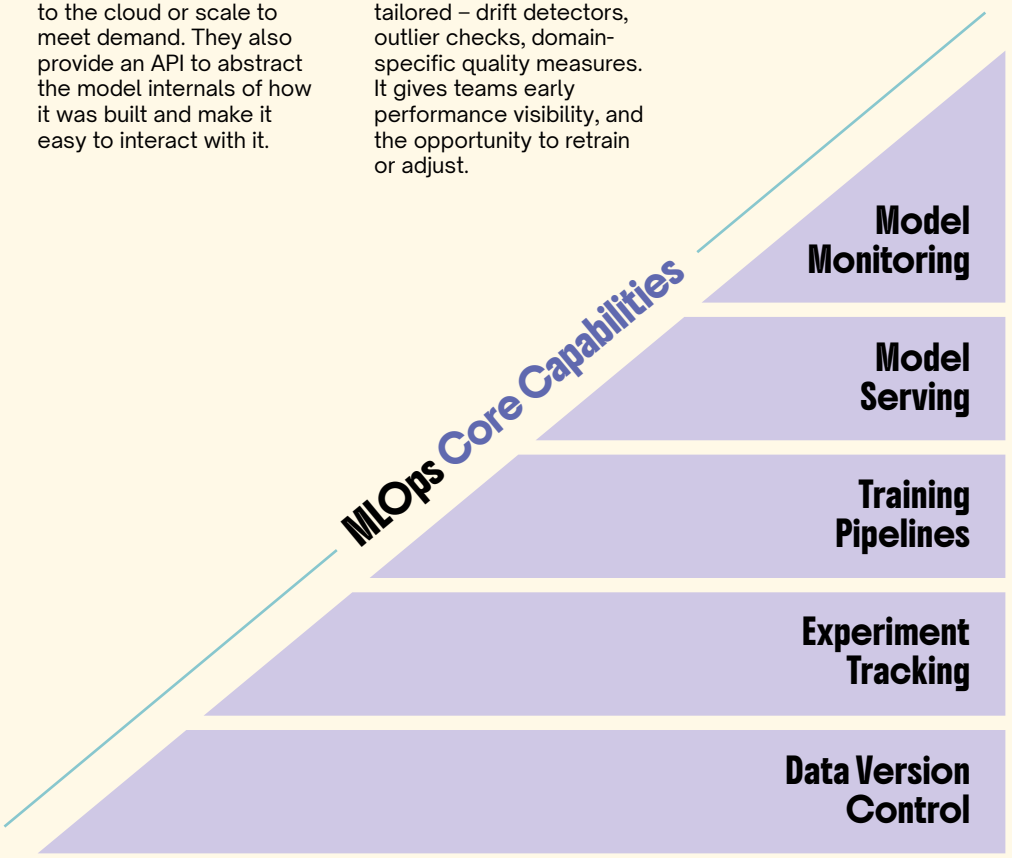
Training Pipelines capture all the steps required to turn raw data into a trained model in a consistent and repeatable way. They define each step – preparing data, training, evaluation, and producing the final model – and ensure those steps run the same way every time. Pipelines make automation straightforward, support distributed or large-scale training when needed, and provide a clean route from experimentation to production.

4. Model Serving

Models alone don't have much value – it's all in how you use them. Model Serving frameworks bring models to life so that they become active components in a wider application. Model serving frameworks allow you to easily deploy your model to the cloud or scale to meet demand. They also provide an API to abstract the model internals of how it was built and make it easy to interact with it.

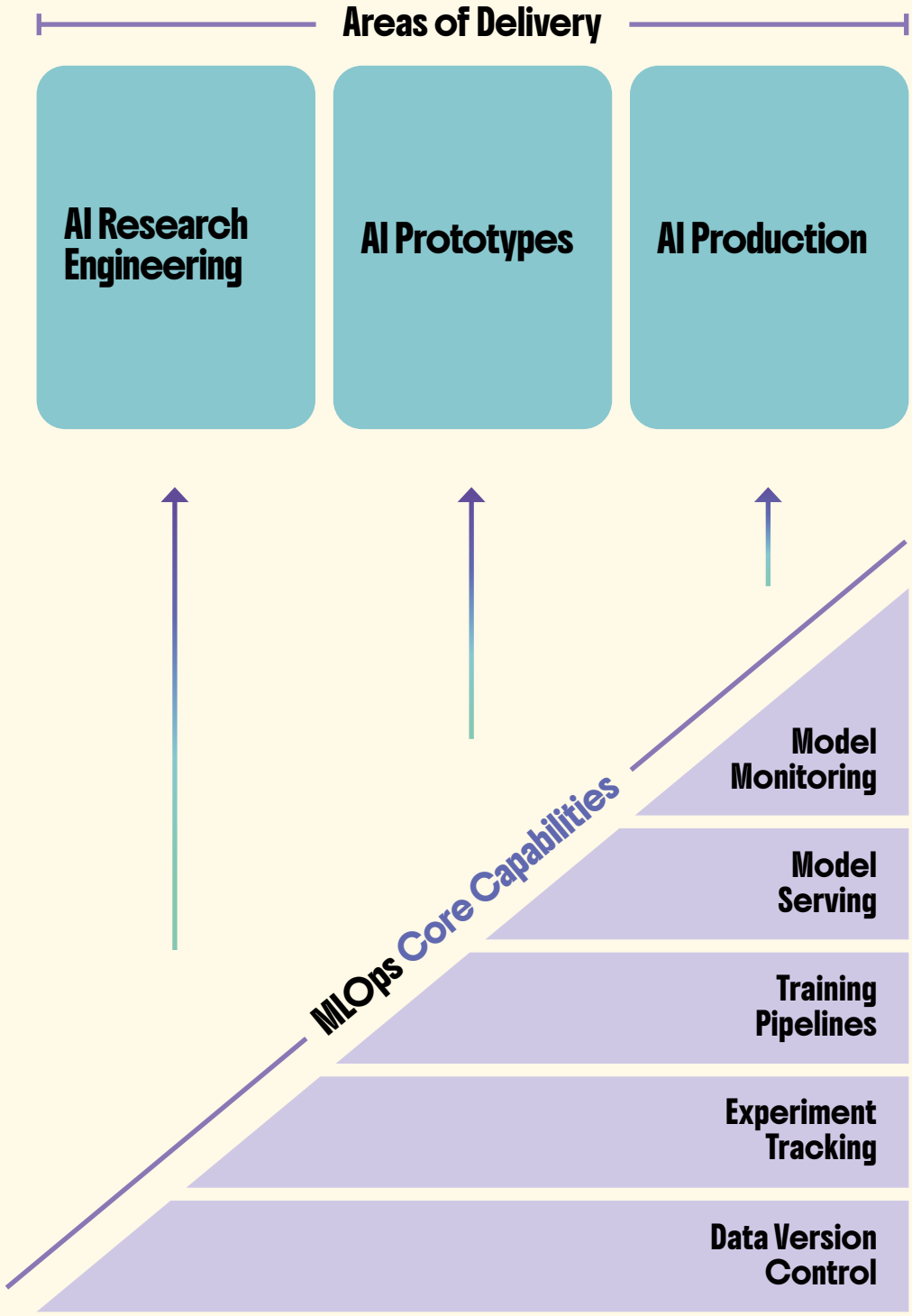
5. Model Monitoring

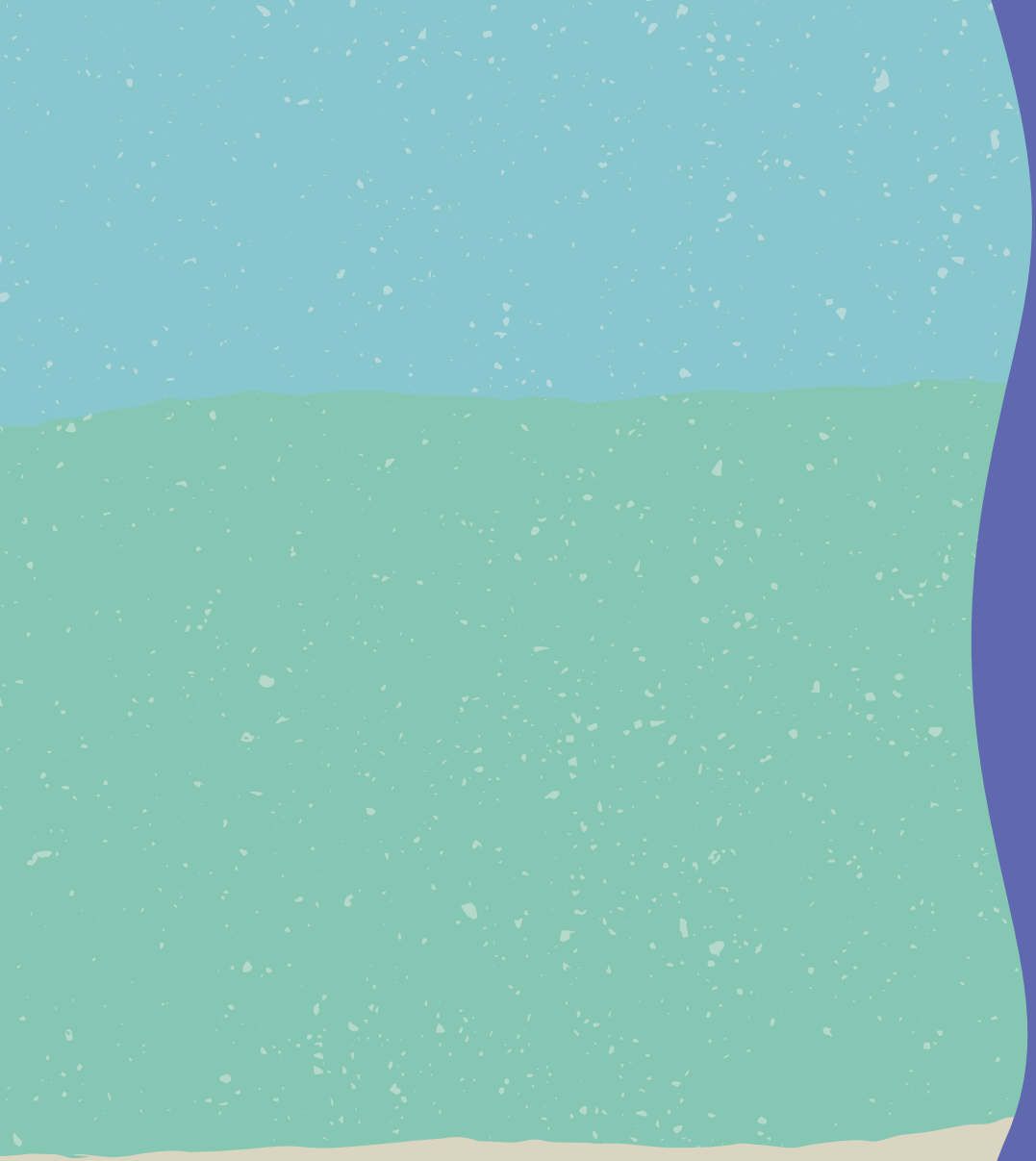
Model Monitoring tracks how a model behaves once it is deployed, capturing signals such as performance, data drift, latency, errors and other indicators of unexpected behaviour. Real-world data changes, monitoring needs to be tailored – drift detectors, outlier checks, domain-specific quality measures. It gives teams early performance visibility, and the opportunity to retrain or adjust.



Appendix

A visual overview of the Fuzzy Labs Approach





fuzzy labs