# The Anatomy of a Machine Learning Pipeline

## An Introduction

Marco Scutari [1] marco@ppml.dev

Mauro Malvestio [2] mauro@ppml.dev

[1] Dalle Molle Institute for Artificial Intelligence (IDSIA), Lugano, Switzerland
[2] DSCOVR, Milano, Italy

September 16, 2024

→ Course Overview

Industrial Seamless Pipe Piercing: Plug Detection and Localisation with Computer Vision

Machine Learning Pipeline: What Is It?

- Project Scoping and Baseline Implementation
- Data Ingestion and Preparation
- Model Training and Evaluation
- Monitoring, Logging and Reporting

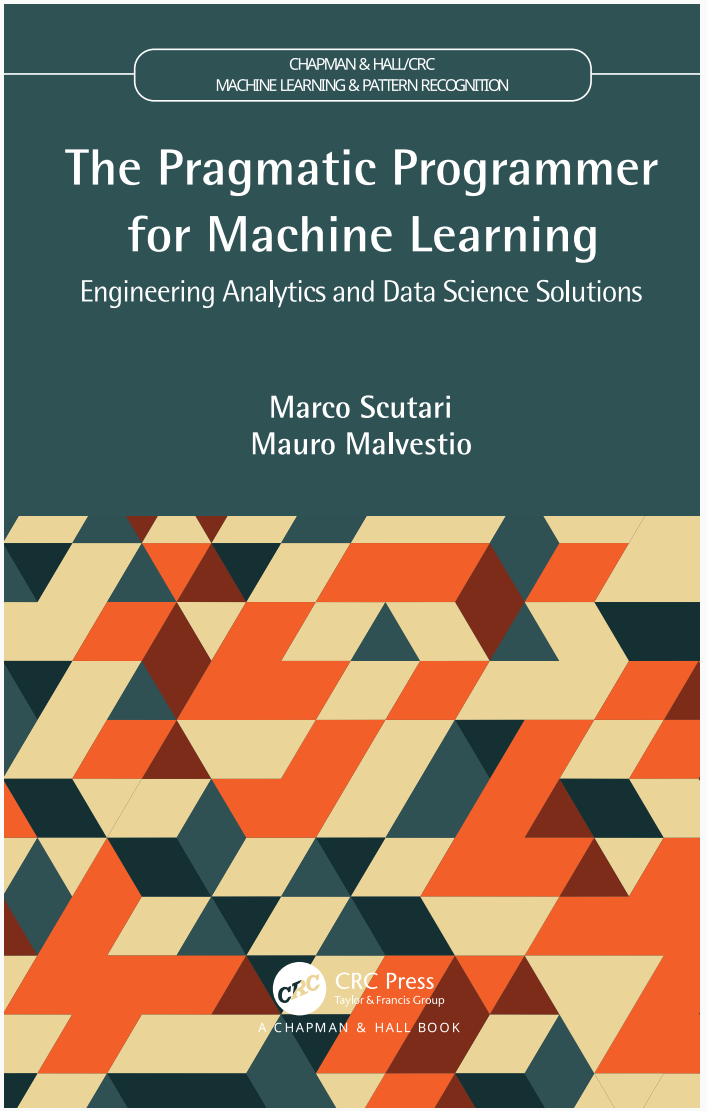
Assorted Trade-Offs and Best Practices

# This Course

## TOPICS:

- A motivating example: Piercing plug detection and localisation in seamless pipe production with computer vision.
- Anatomy of a machine learning pipeline.
- The software and data lifecycles.
- Setting up and working with a pipeline, illustrated with code.
- Trade-offs and best practices specific to machine learning pipelines.

## MORE ABOUT THIS:



# Who We Are


## Marco Scutari

- Senior Researcher in Machine Learning, Consultant.
- Background in statistics and computer science.
- Worked at UCL, University of Oxford; now at Swiss national AI research centre IDSIA.
- Writes popular scientific libraries (bnlearn, fairml).

## Mauro Malvestio

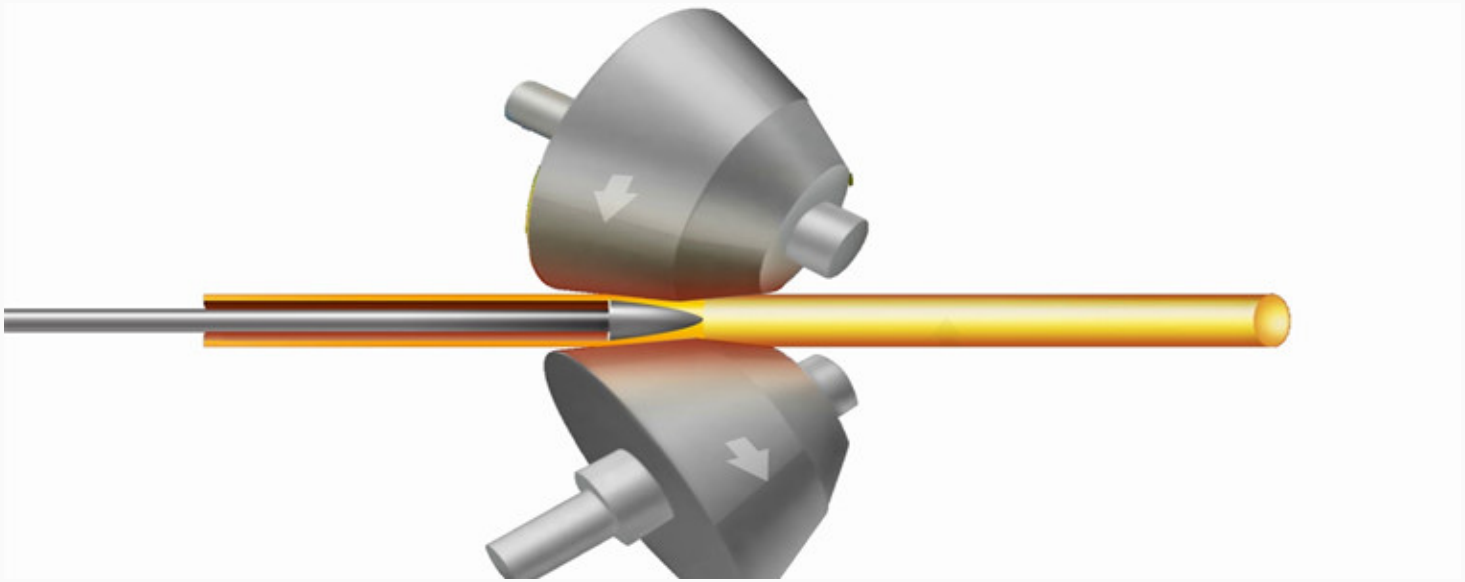
- Senior Technologist, Consultant.
- Background in software engineering, machine learning systems, embedded systems and cloud computing.
- Worked in software engineering, IT operations, and CTO roles.

- ✓ Course Overview
- → **Industrial Seamless Pipe Piercing: Plug Detection and Localisation with Computer Vision**
  - Machine Learning Pipeline: What Is It?
    - Project Scoping and Baseline Implementation
    - Data Ingestion and Preparation
    - Model Training and Evaluation
    - Monitoring, Logging and Reporting
  - Assorted Trade-Offs and Best Practices

# Use Case: Seamless Pipes Production

Seamless pipes for the Oil & Gas and Chemical industries are produced using a rotary piercing method, where circular billets (solid tube blanks) are passed between two rotating rolls and pierced by a fixed plug.

This is what the process looks like:

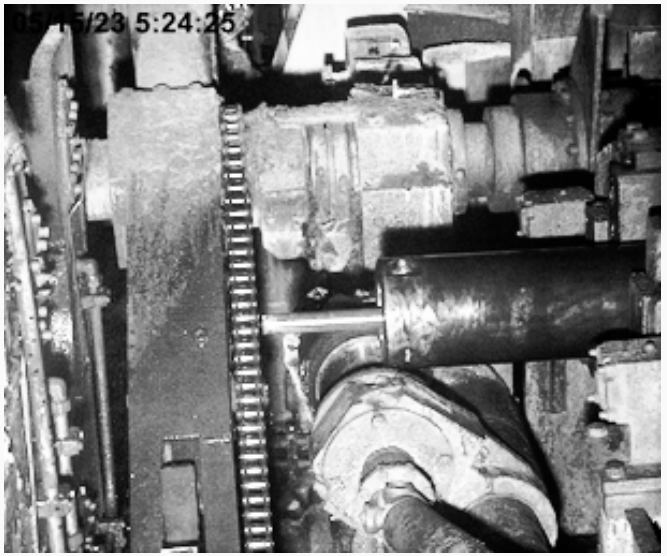


The rotary piercing process is crucial in shaping the round billet into a tubular form and producing seamless pipes.

# Use Case: The Domain Problem

The plug is not just fundamental for shaping and tubular quality. Operating without the plug without proper automated detection increases the risk of severe operational hazards and damage to the manufacturing facility.

with Plug (True positive)



without Plug (True negative)

Occasionally, the mechanical loading fails to attach the plug to the spindle, risking a drill operation without it.

# Use Case: Business Objective

The computer vision model evaluates in real-time if the conditions for starting the piercing process are met, and it delivers feedback to an industrial actuator via a Programmable Logic Controller (PLC).

Technical challenges: the model should

1. work in near real-time, invoked by a remote edge process/machine;
2. be accessible via a remote low-latency inference API;
3. be robust against false positives (when the model detects the plug, but the plug is not actually present).

# Use Case: Vision Approach

We require an approach towards the development of this model to ensure it integrates two key functionalities:

1. Classify each image as “Plug” or “no-Plug”.
2. Locate the plug and highlight it with a bounding box when present.

We use a Convolutional Neural Network (CNN) architecture, specifically the YOLO (You Only Look Once) model, which has been fine-tuned on an industrial data set comprising 2000 images.

- ✓ Course Overview
- ✓ Industrial Seamless Pipe Piercing: Plug Detection and Localisation with Computer Vision
- → Machine Learning Pipeline: What Is It?
  - Project Scoping and Baseline Implementation
  - Data Ingestion and Preparation
  - Model Training and Evaluation
  - Monitoring, Logging and Reporting
- Assorted Trade-Offs and Best Practices

# What is a Machine Learning Pipeline

Depending on whom you ask, you may get different definitions of what a pipeline is:

1. A data scientist will tell you that it is a sequence of operations performed on data from data preparation to model selection, model estimation and inference. In other words, a data analysis pipeline.
2. A software engineer will tell you that a pipeline is the workflow that underlies the process of developing and delivering a piece of software. In other words, a software development pipeline.

Both are true for a machine learning pipeline, which codifies the steps of developing machine learning (ML) software from a simple proof of concept into a production software that uses data to answer some business or academic need.

# What is a Machine Learning Pipeline

A machine learning pipeline is the codification of

- data ingestion and data preparation;
- model training and experiment tracking;
- monitoring, logging and reporting.

into independent, reusable, modular parts that can be pipelined together to orchestrate the flow of data into, and outputs from, machine learning models.

Why should we bother?

- Independent: we can track changes effectively.
- Reusable: we can use the same code in development and production for different tasks.
- Modular: we can swap models and data to explore new problems and adapt to change.

# Believe it or not: how much can we rely on published data on potential drug targets?

*Florian Prinz, Thomas Schlange and Khusru Asadullah*

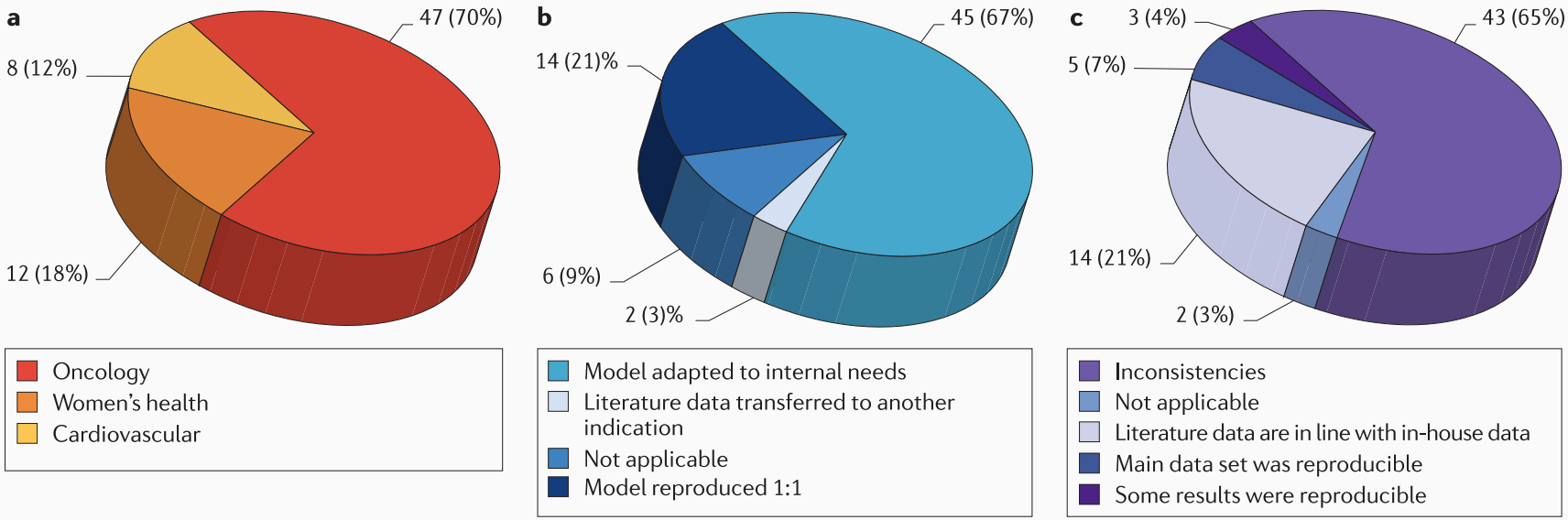



Figure 1 | **Analysis of the reproducibility of published data in 67 in-house projects.**

# Reality check on reproducibility

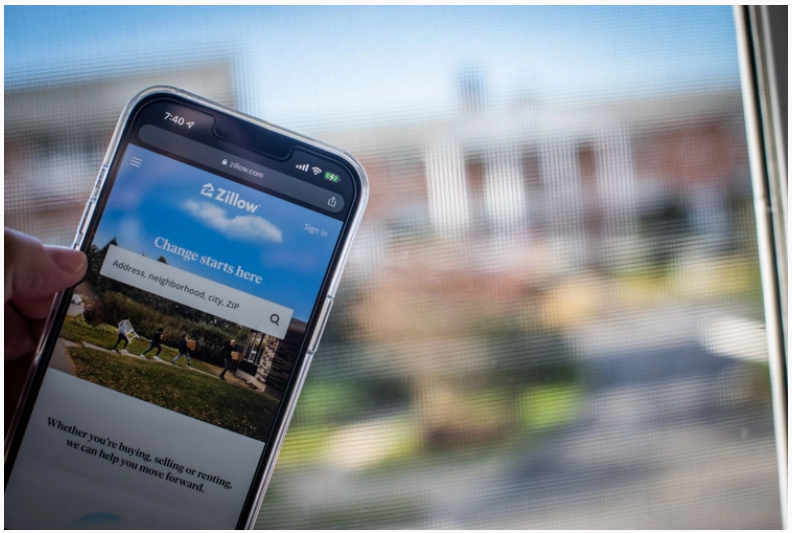
*A survey of* Nature *readers revealed a high level of concern about the problem of irreproducible results. Researchers, funders and journals need to work together to make research more reliable.*

FORTUNE | EDUCATION

# What Zillow's failed algorithm means for the future of data science

BY ERIK SHERMAN

February 01, 2022, 5:19 PM




Big-data analysis told Zillow what to offer and how much to charge on the flip. Easy peasy. Except, come 2021, the wheels came off. Zillow had bought thousands of houses, and the algorithms didn't factor in repairs with the skyrocketing costs of materials and labor.


# Knightmare: A DevOps Cautionary Tale

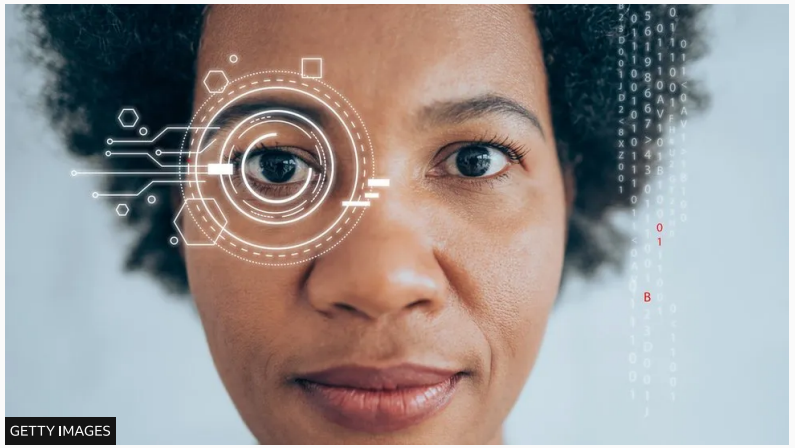
D7 DevOps April 17, 2014

This is the story of how a company with nearly $400 million in assets went bankrupt in 45-minutes because of a failed deployment.

BBC NEWS

# Twitter finds racial bias in image-cropping AI

20 May 2021




**Twitter's automatic cropping of images had underlying issues that favoured white individuals over black people, and women over men, the company said.**

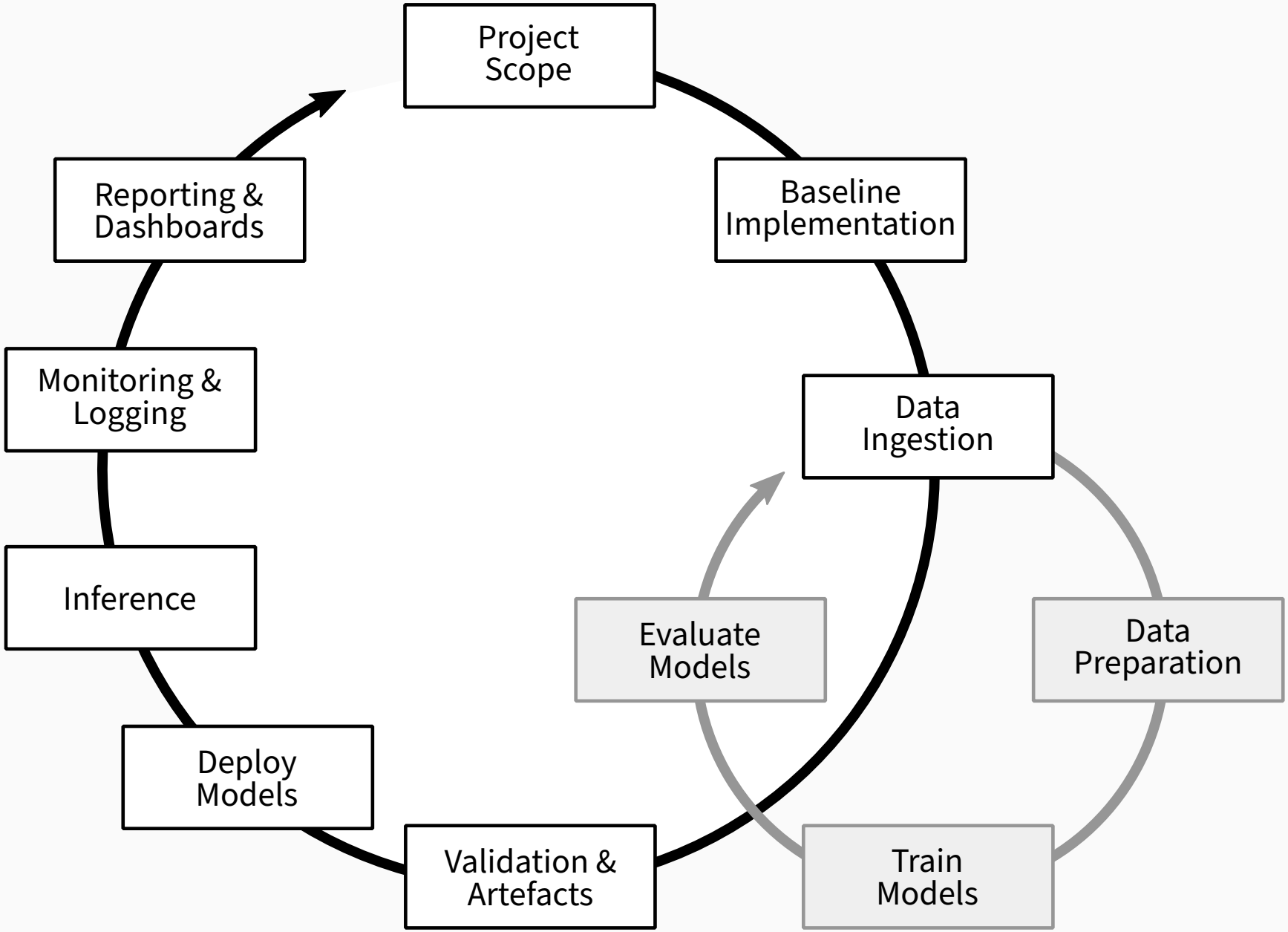
We can make machine learning pipelines robust and reliable with:

- Continuous integration (CI): committing small, frequent changes to a version control repository.
- Continuous Delivery (CD): being able to release a working version of the software at any time.

We do that for models, data and code in an experiment tracking platform to implement MLOps (DevOps for ML) and automate testing, release management and deployment.

Without experiment tracking and MLOps, we quickly accumulate technical debt at the data, model, architecture and code levels because we cannot keep track of their complexity as the pipeline evolves over time.

# The Lifecycle of an ML Pipeline, in Pictures



# Software is never finished, Neither Are Data

Data change.

Software is never finished.

The needs that motivate its existence change over time.

A machine learning pipeline is improved and evolved over time using MLOps: it has a lifecycle.

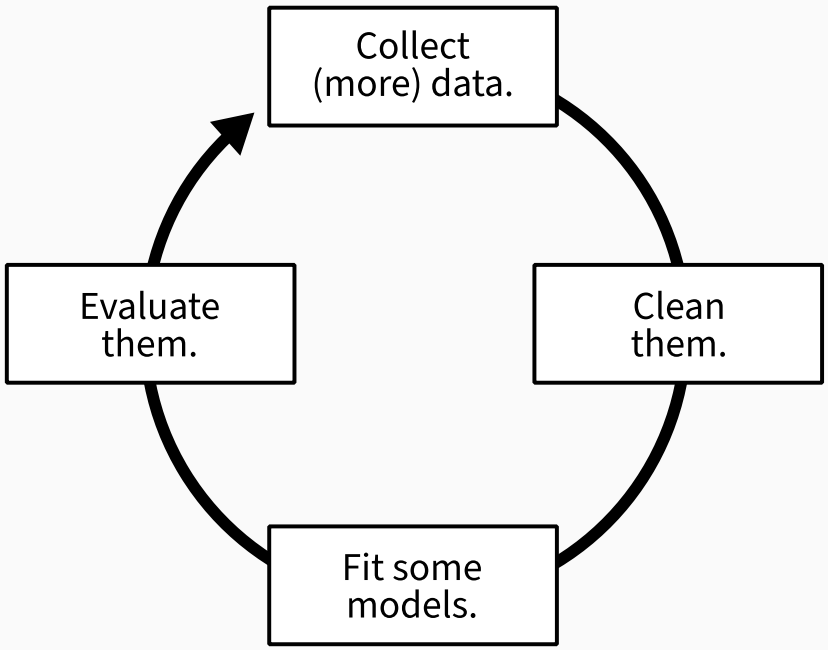
Broadly speaking, we iterate over four phases:

- planning and exploration (Project Scope, Baseline Implementation);
- data and models (Data Ingestion, Data Preparation, Train Models, Evaluate Models);
- deploying to production (Validation & Artefacts, Deploy Models);
- running and using the ML pipeline (Inference, Monitoring & Logging, Reporting & Dashboards).

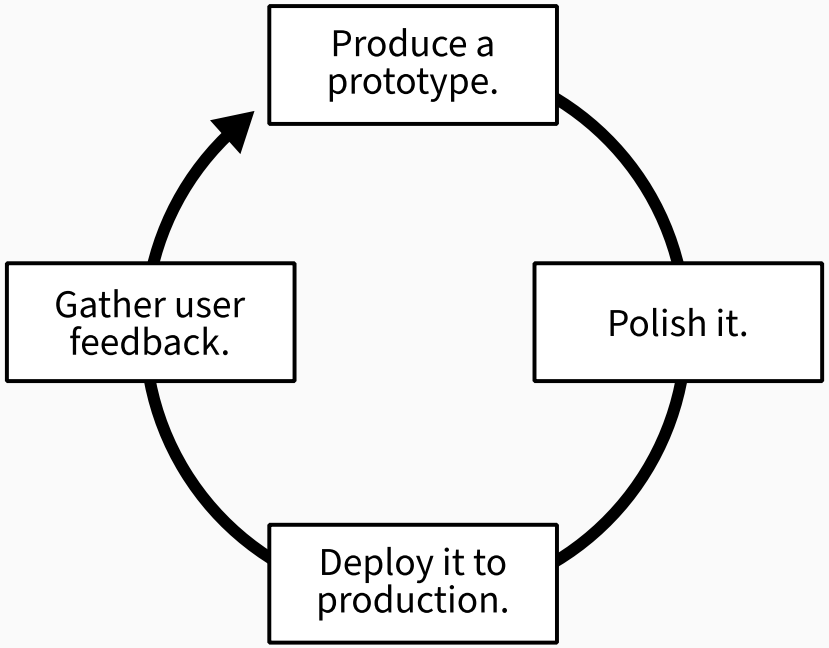
# ML Combines Statistics and Software

Different parts of the ML pipeline lifecycle should sound familiar.

To a data scientist:




To a software engineer:




As a result, it is natural that best practices from both data science and software development apply. And they apply at the same time, along with the best practices that domain experts bring with them in contributing to the design and implementation of the ML pipeline.

# Project Scoping

The first step in building a machine learning pipeline is to understand what it is supposed to do. We should:

- Identify the problem we want to solve: a concrete business or academic need which is worth addressing for enough people.
- Identify the targets we want to optimise for: measurable domain metrics with achievable threshold values that define "success".
- Identify what data we need: all the data sources we want to use, who owns them, and how to access them. Choose them following the best practices from survey sampling and experimental design!
- Perform a preliminary analysis:
  - how much data we can collect, what variable types they will contain;
  - models based on their sample size requirements, their assumptions and the inference types they support (prediction, classification, etc.).

Once we know all this, we can build a minimum viable pipeline.

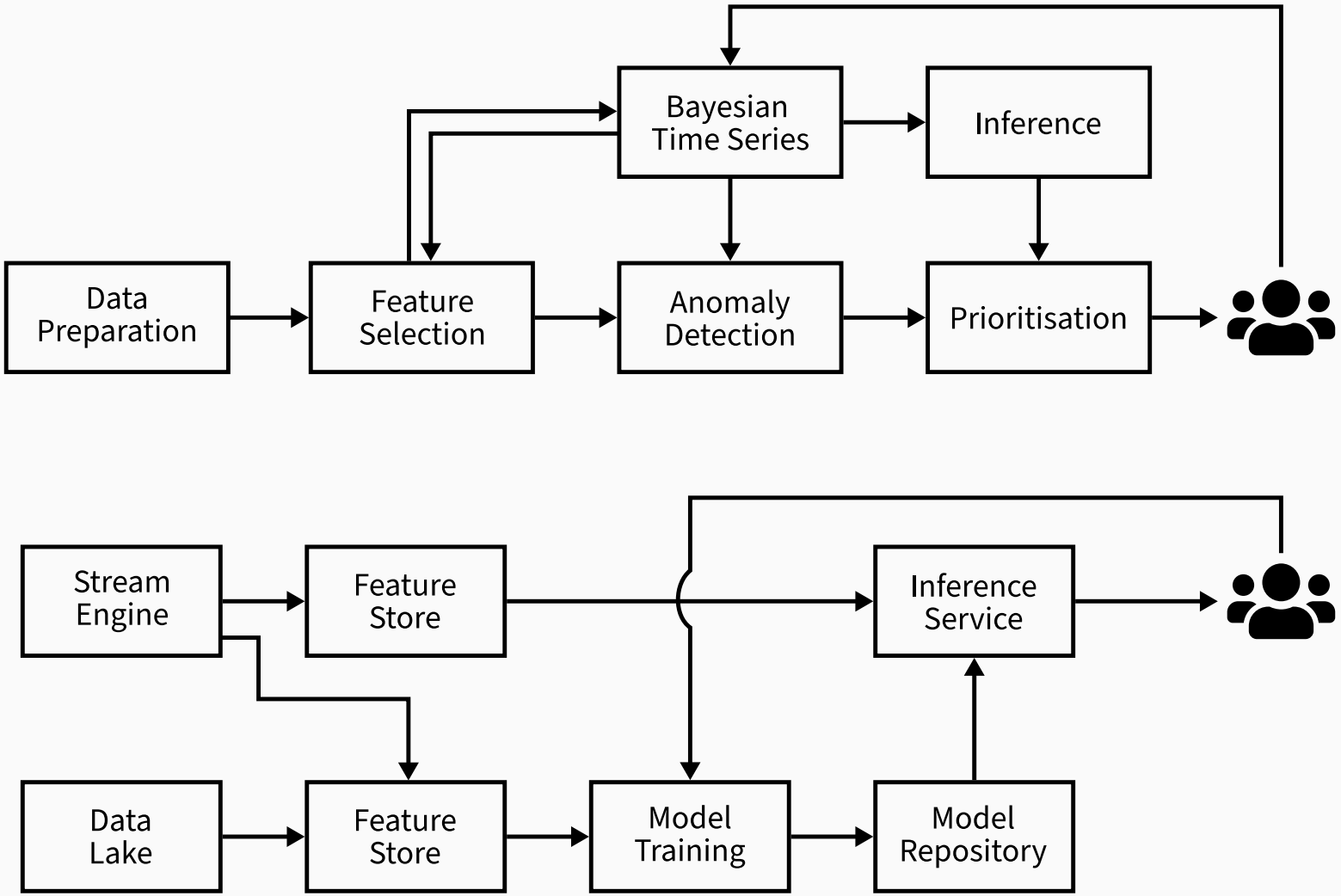
More in detail:

- Robustness: against the noise in the data, model misspecification and adversarial attacks.
- Interpretability and explainability: how well we can understand the behaviour and the outputs of the models either directly or through auxiliary models to provide post hoc explanations.
- Fairness: we should not discriminate against individuals or groups based on sensitive attributes such as gender, race or age. Models can easily incorporate the biases present in the training data.
- Privacy and security: Machine learning models should protect privacy by not disclosing personally identifiable information.

In addition to:

- Hardware requirements: how much (and what types of) compute, how much storage and how we should connect them.
- Software requirements: what MLOps platforms, what programming languages, what libraries, etc.

# The Pipeline as a Directed Acyclic Graph (DAG)



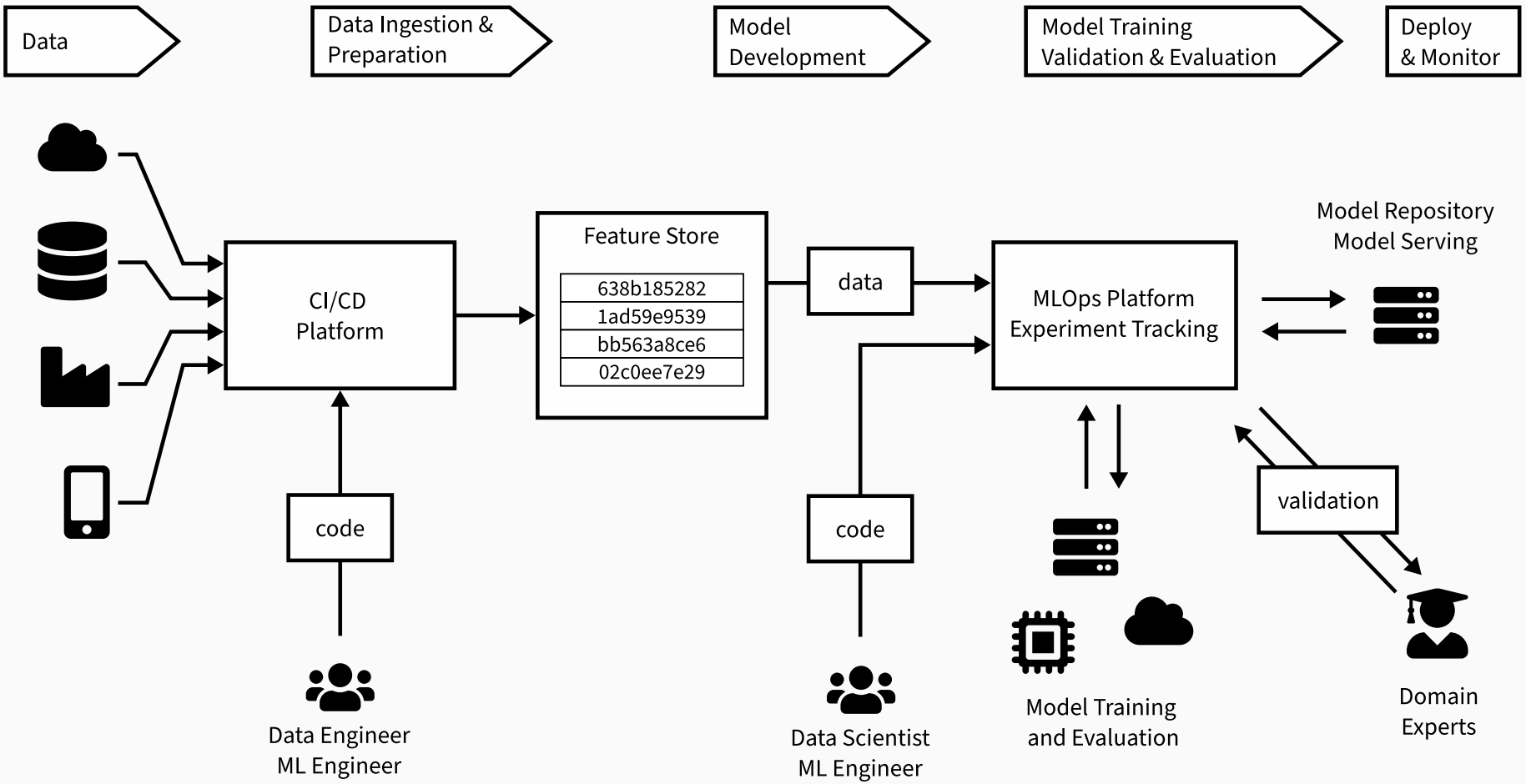

The DAG maps the paths of execution of the pipeline and the flow of data and information. For complex pipelines, we can split it into smaller DAGs.

# Baseline implementation

Sketching a baseline implementation from the DAG is key to validating all the steps of the pipeline and evaluating different approaches. The starting point of producing the real thing by:

- Starting to use experiment tracking: version control all the code, the data and the models to enable continuous integration.
- Constructing a suite of software tests: the starting point to transform the proof of concept into production-quality code by gradually refactoring and documenting it with the help of code review.
- Improving scalability: A proof of concept is typically built using a small fraction of the available data, so we must ensure that its computational complexity is small enough to make learning and inference feasible in production when all data are used.

# The Anatomy of an ML Pipeline, in Pictures




It's an team effort by Software Developers, Data Scientists, Machine Learning Engineers and Domain Experts!

# Data Ingestion

The first part of the pipeline will comprise one or more data ingestion modules which collect data from various sources: relational databases, legacy OLTP/OLAP systems and modern in-house or cloud data lakes.

After collecting/downloading the raw data, we do ETL:

1. Extract the relevant data from the source.
2. Transform the data to allow for further processing.
3. Load the data into a centralised repository.

Issues:

- Tracking data provenance and complying with HIPAA/GDPR/etc.
- Monitoring data sources for availability and data drift.
- Reconciling overlapping data sources.

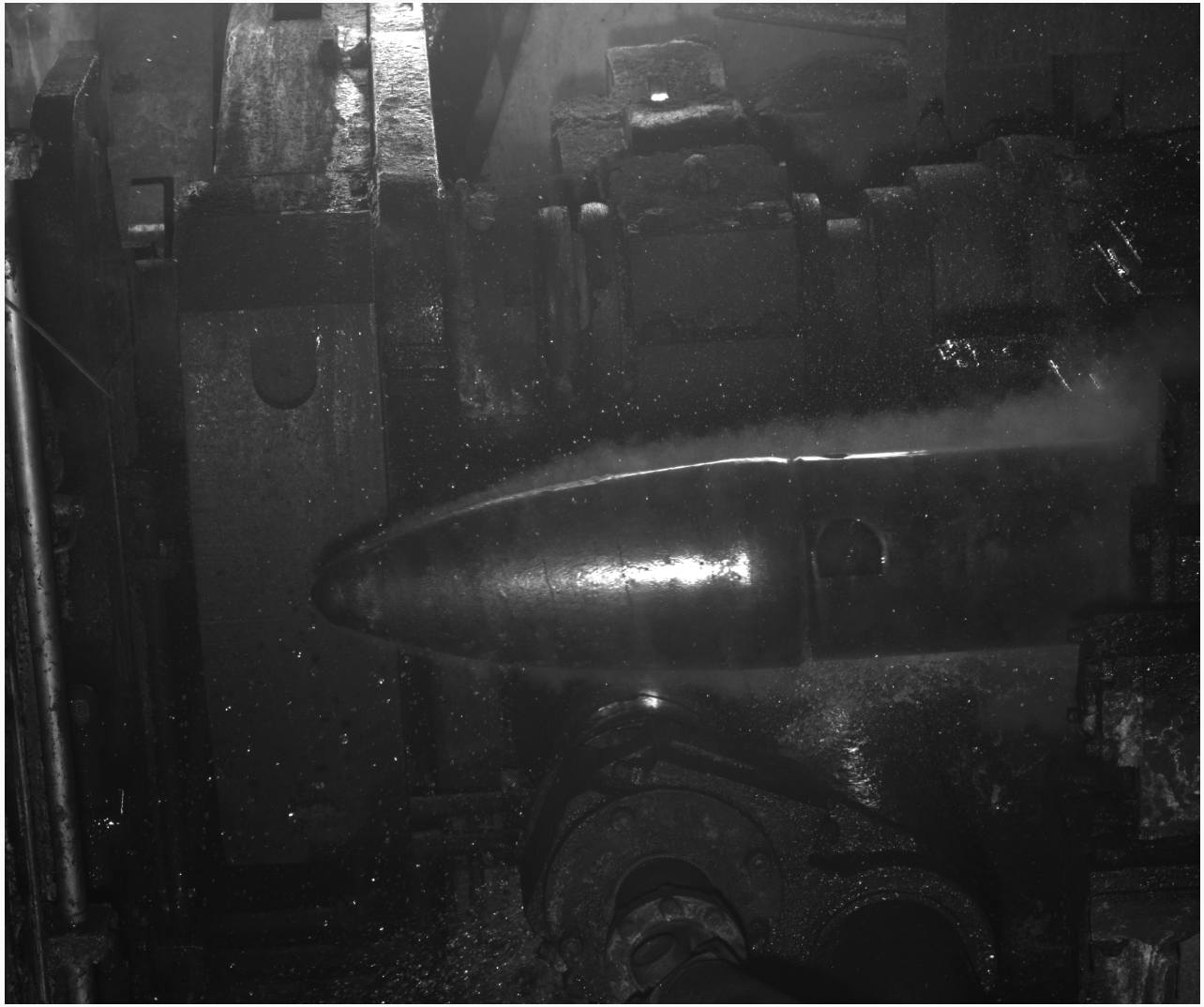
# Data Ingestion: In Our Use Case

The source data are full resolutions images of 2448 by 2050 pixels in 8-bit grayscale taken by an industrial camera, triggered by the PLC.

Each of them is:

1. Stored in the internal flash memory of the camera.
2. Copied via HMI-Panel script to a network share on a local storage.
3. Processed by a long-running task (a Python process) for image preparation and optimization.
4. Copied the original and processed versions on remote object storage (S3) for archiving.

# In Our Use Case: Raw Images



An example of a raw image of 2448 by 2050 pixels in 8-bit grayscale.

# Data Preparation

Improving the quality of the ingested data in an automatic and reproducible way makes later stages of the pipeline more reliable. We do:

- Validate the types, the acceptable values and the statistical distribution of each feature.
- Perform feature selection and feature engineering.
- Split the data into training, validation and test sets for later use.

Issues:

- Labelling data is hard.
- Separating true causal features from epsilon features and marginal features is hard.
- Data will change over time, so setting fixed thresholds is hard.

# In Our Use Case: Image Optimisation

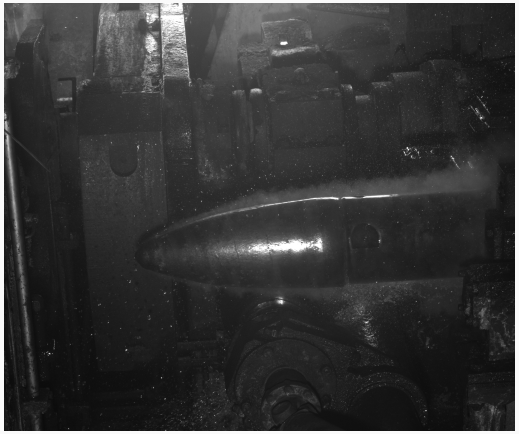
The raw images are processed through several steps to optimize them for effective labelling and training.

## Preprocessing:

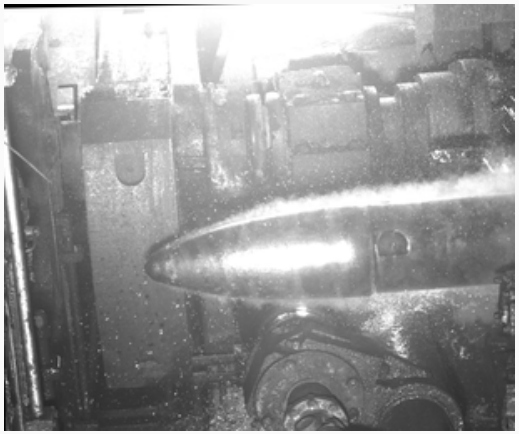
- Resizing to 306 x 256 pixels.

## Augmentations:

- Horizontal shifting.
- Brightness adjustment.
- Rotation.
- Zoom.



Before



After

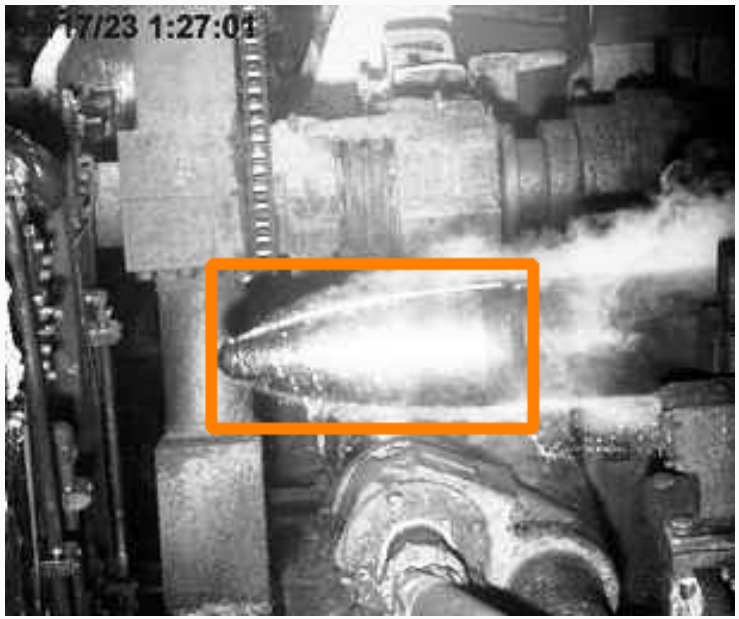
# In Our Use Case: Labelling

Images must be labelled, and the coordinates of the bounding box exported in YOLO format:

| class_id | x_center | y_center | width | height |
|---|---|---|---|---|

Label metadata:

- class_id: 0 (Plug)
- x_center: 0.5031
- y_center: 0.5587
- width: 0.4418
- height: 0.2691



We can use open-source tools such as CVAT or Label Studio for labelling.

# In Our Use Case: Sample Creation and Versioning

The 2075 images are split into 80% training (1660), and 20% validation (415), organised following the YOLO directory structure.

```
└── plugs
    ├── train
    │   ├── images
    │   │   ├── 00232545-2974-4180-a1f3-79764ab1daca_1.png
    │   │   ├── 0028912f-781f-4c0a-a002-d021e7d42eb2_1.png
    │   │   [...]
    │   ├── labels
    │   │   ├── 00232545-2974-4180-a1f3-79764ab1daca_1.txt
    │   │   ├── 0028912f-781f-4c0a-a002-d021e7d42eb2_1.txt
    │   │   [...]
    └── val
        ├── images
        │   ├── 001cb842-c6ad-4ea3-96cc-76ec2609b87c_1.png
        │   ├── 005b5920-b556-44b1-8305-fc47d9b1d492_1.png
        │   [...]
        ├── labels
        │   ├── 001cb842-c6ad-4ea3-96cc-76ec2609b87c_1.txt
        │   ├── 005b5920-b556-44b1-8305-fc47d9b1d492_1.txt
        │   [...]
```

# Model Training and Evaluation

The pipeline schedules training workloads on compute systems with the appropriate hardware and monitors their progress. It should also:

- simplify the parallel training of models with predefined, regular patterns of hyperparameters;
- automate software tests implementing property-based testing of the model's probabilistic properties;
- support the fine-tuning of pre-trained models;
- version models and perform experiment tracking.

We evaluate whether a newly trained model is better in statistical terms than the model we are currently using in terms of predictive accuracy.

# In Our Use Case: Training with a Declarative Approach

Training requires a systematic approach to its configuration and logging:

- Parameters and hyperparameters are managed through a declarative approach with YAML files (`params.yaml`).
- The overall training process is monitored and logged using experiment tracking tools such as Weights & Biases or MLFlow.

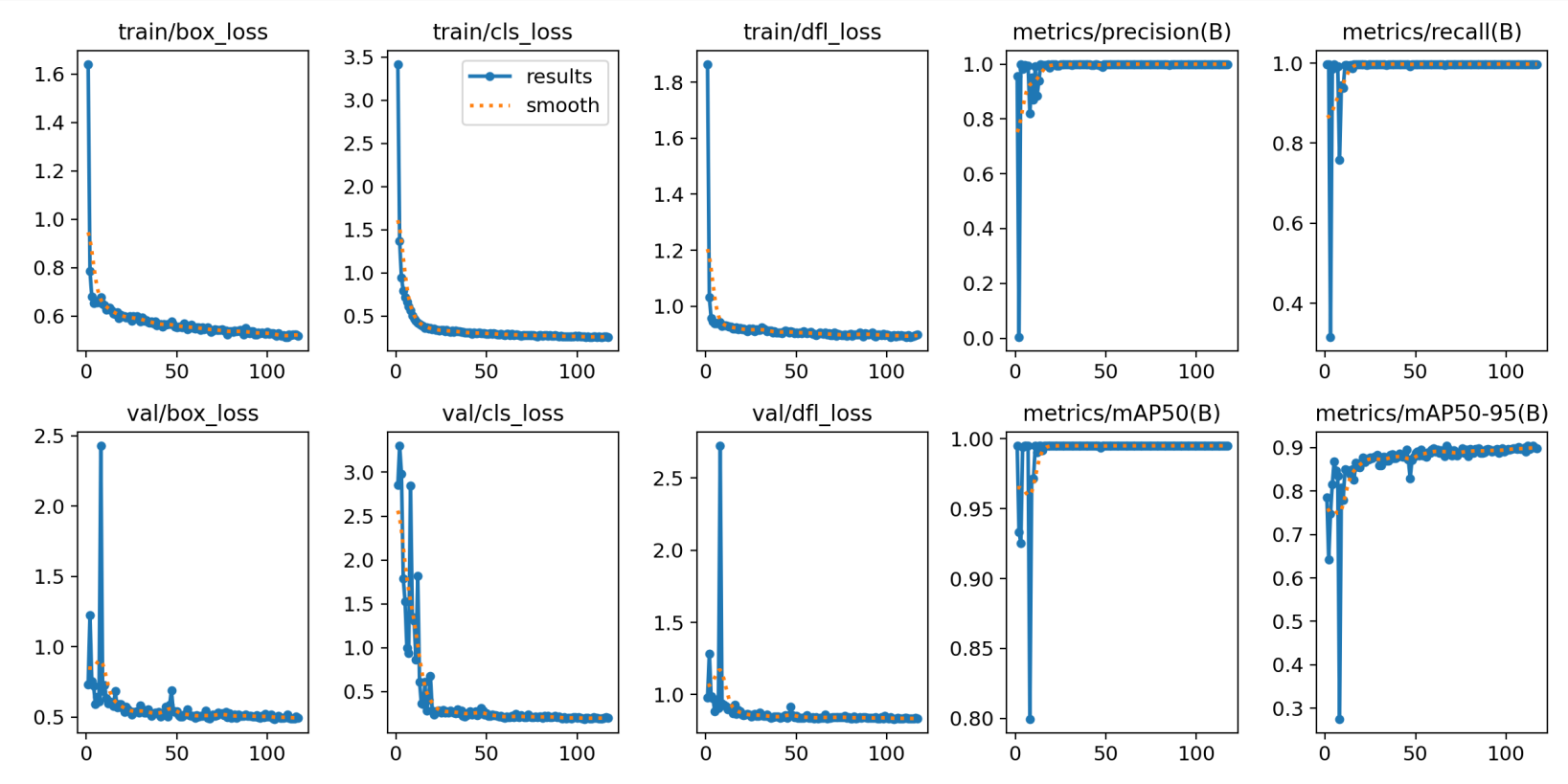
```yaml
pretrained:
  model: "yolov8n.pt"

train:
  data: "data/plugs/dataset.yaml"
  epochs: 100
  seed: 42
  batch: 32
  workers: 4
  best_model: "runs/detect/train/weights/best.pt"
  model_path: "models/best_model.pt"
```

# In Our Use Case: Monitoring Metrics during Training

During the training, we use experiment tracking to monitor:

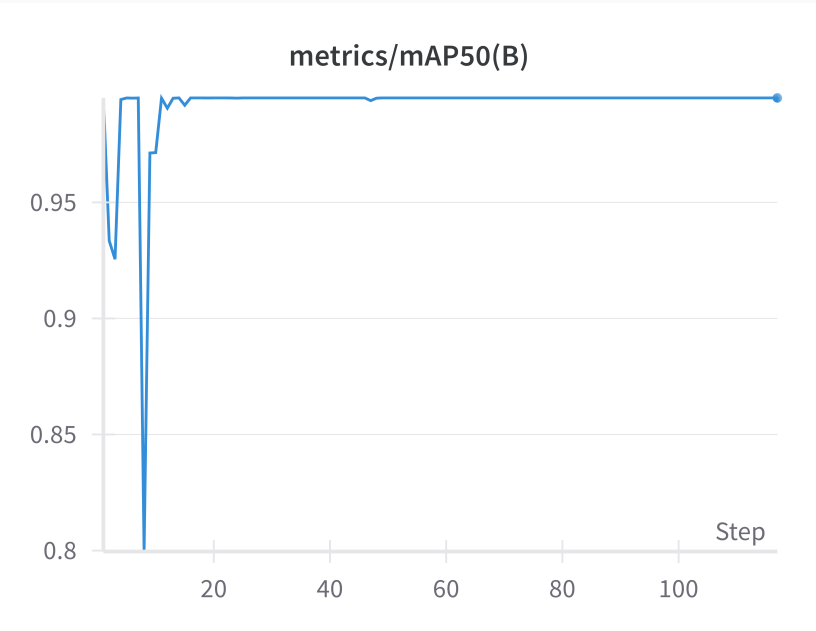
- Loss functions: box loss, distribution focal loss (DFL), class loss.
- System metrics: especially GPU utilisation metrics (RAM).

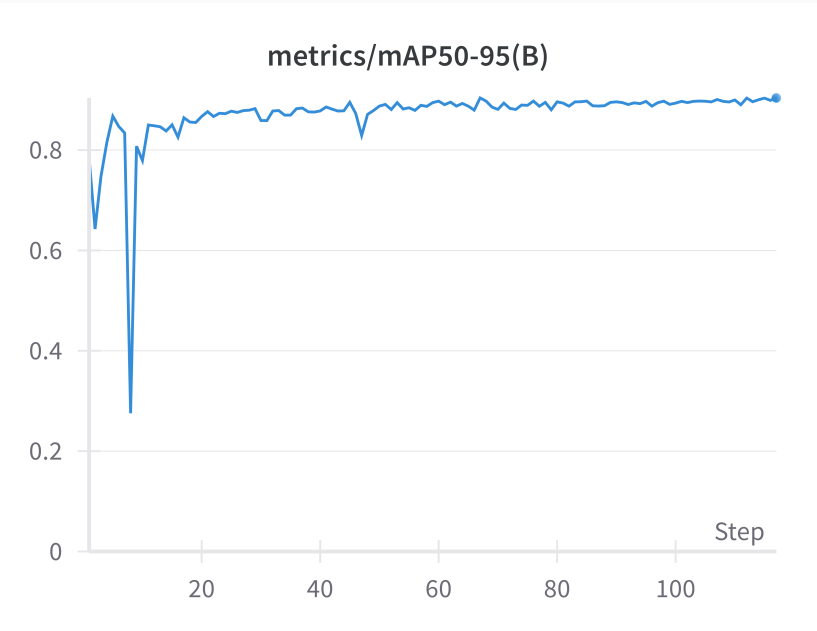


# In Our Use Case: Collecting Performance Metrics

We evaluate the model on the validation data set to assess its performance metrics:

- mAP50: Mean Average Precision at an IoU threshold of 0.50.
- mAP50-95: Average of Mean Average Precision at IoU thresholds from 0.50 to 0.95.



# Model Validation

We also check whether the new model is preferable to the current one in domain terms to validate it using the metrics from the scoping phase.

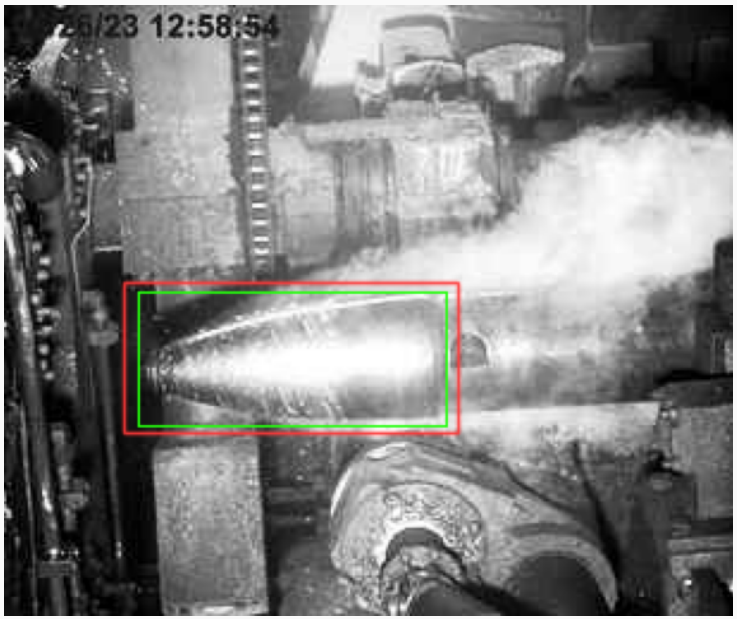
Model evaluation and model validation agree if we choose well-matched domain metrics and statistical accuracy measures. Models with poor statistical properties will typically not encode the domain well enough for practical use, but models with good statistical properties are not necessarily useful either.

Model evaluation can be automated to a certain extent. Model validation requires a domain expert.

# In Our Use Case: Model Validation

Model validation combines two approaches:

- visual inspection of the model predictions by overlaying the predicted bounding box vs the ground-truth bounding box; and
- automatic validation of how much the predicted bounding box matches the actual box (Intersection over Union, “IoU”).

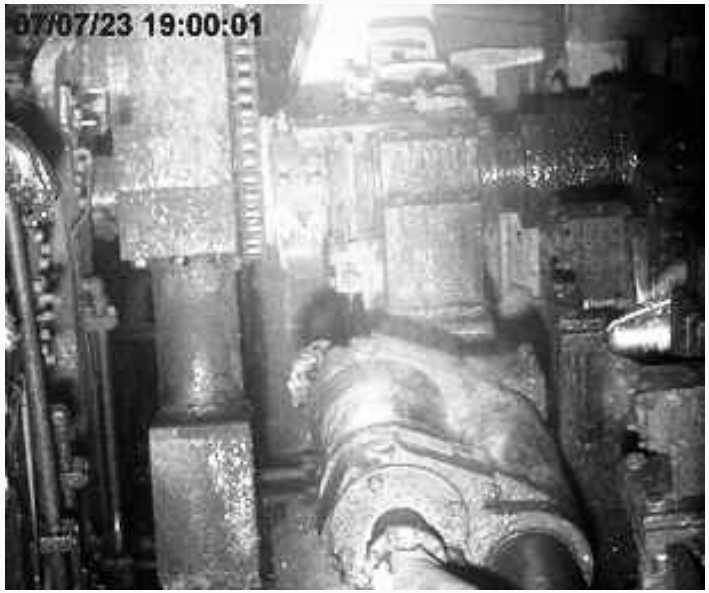


The IoU between the red box and the green ground truth box is 0.82.

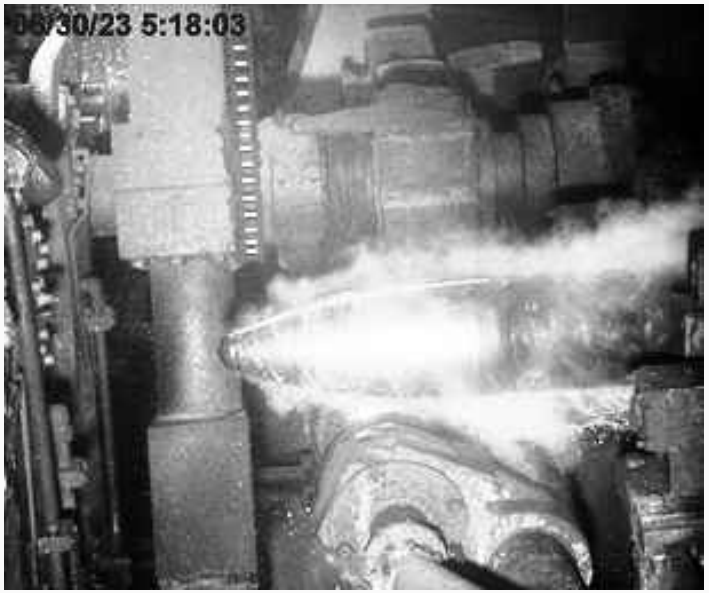
# In Our Use Case: Continuous Validation and Retraining

Evaluating model metrics in production is essential:

- The industrial environment can change due to physical modifications to the production environment.
- Changes can also occur because of technological updates, such as improvements in hardware.
- Therefore, the production images have a different distribution than those on which the model was originally trained and tested.



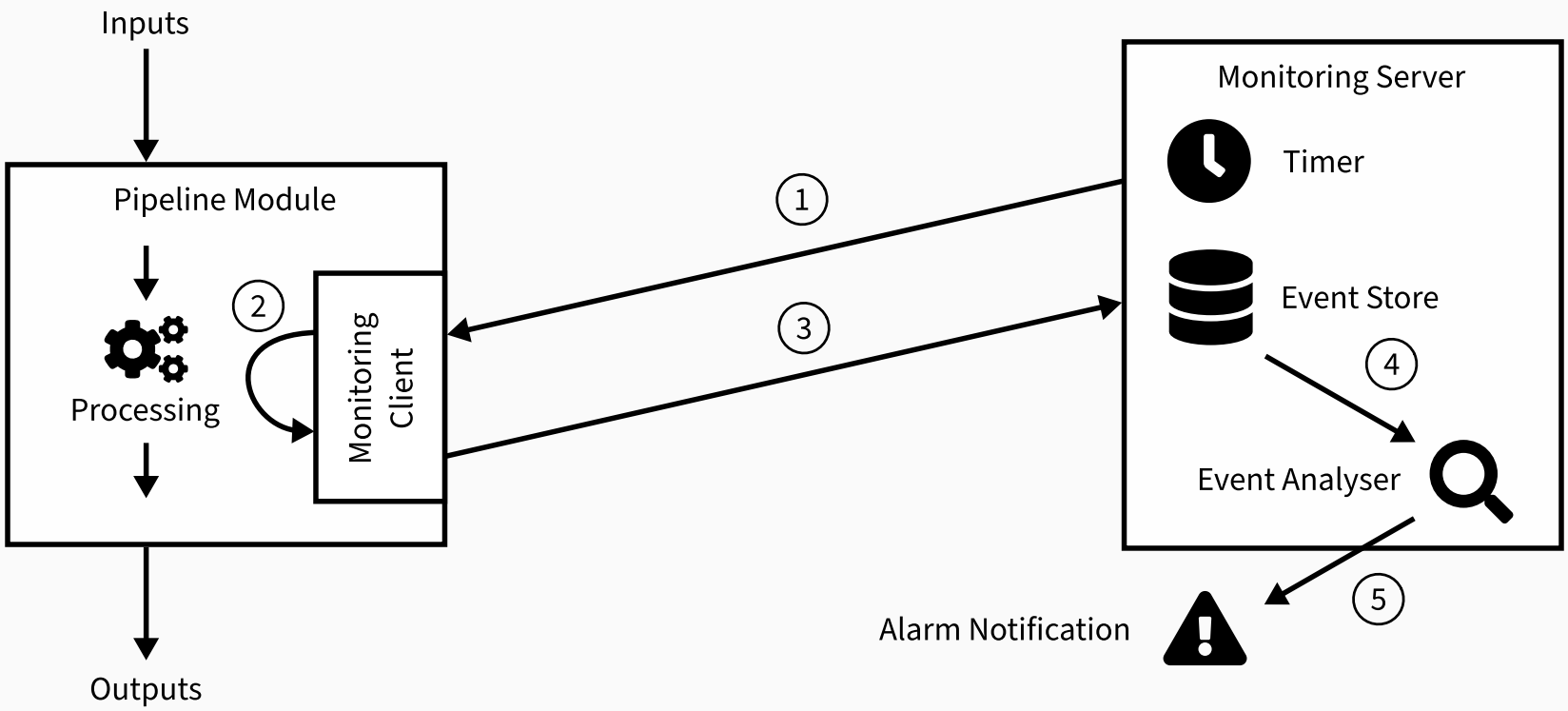

Plug barely visible on the right.




Lots of smoke clouding the plug.

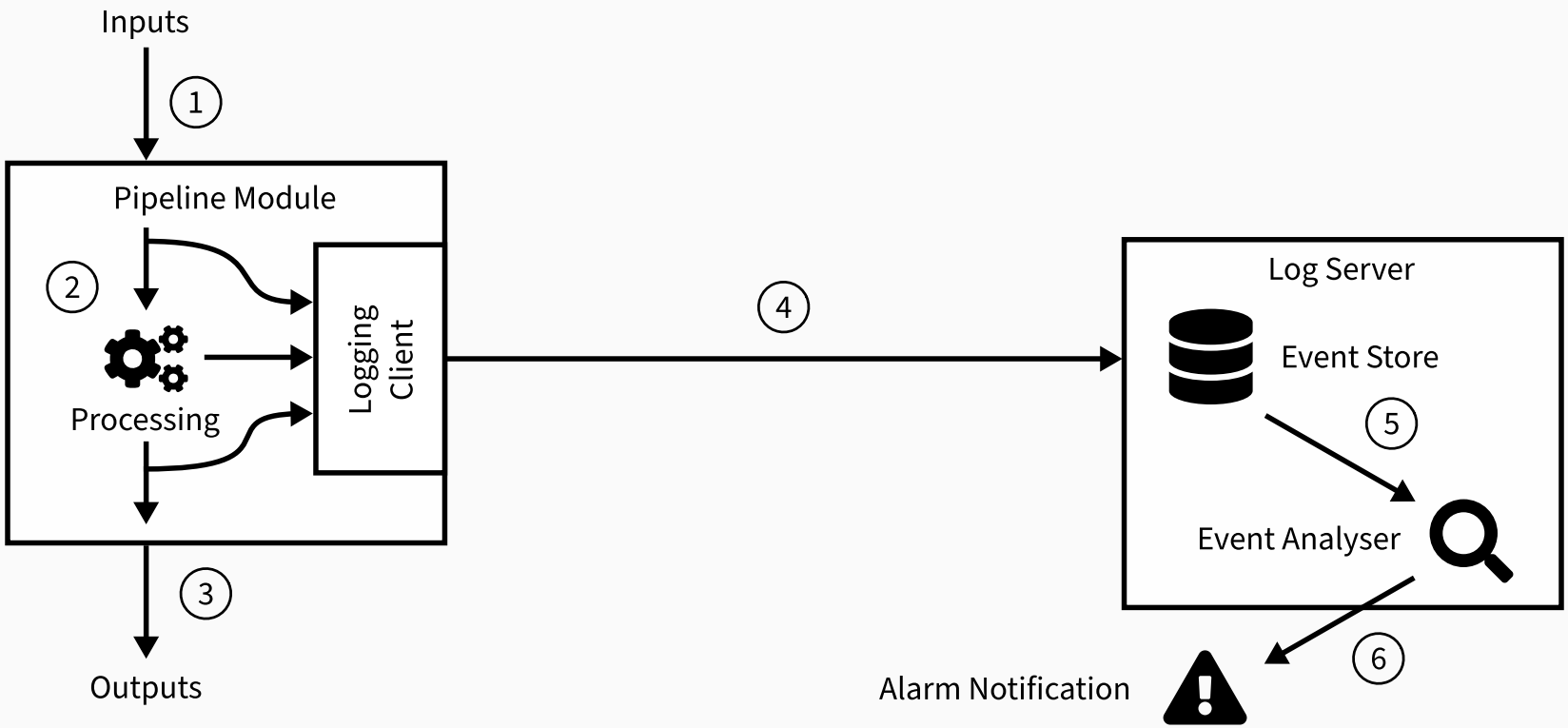
# Monitoring

Monitoring collects the metrics to track whether the pipeline and the individual modules achieve the required statistical and domain performance levels at all times and to allow us to pinpoint the source of any issue we may have to troubleshoot.



# Logging

Logging complements monitoring by recording relevant information about events that occur inside individual modules (or the pipeline orchestration) in a sequence of timestamped log messages. This improves our ability to debug and troubleshoot issues on remote systems we cannot access directly.



# Reporting

Reporting implements graphical interfaces that display monitoring and logging information with intuitive, interactive dashboards. For instance:

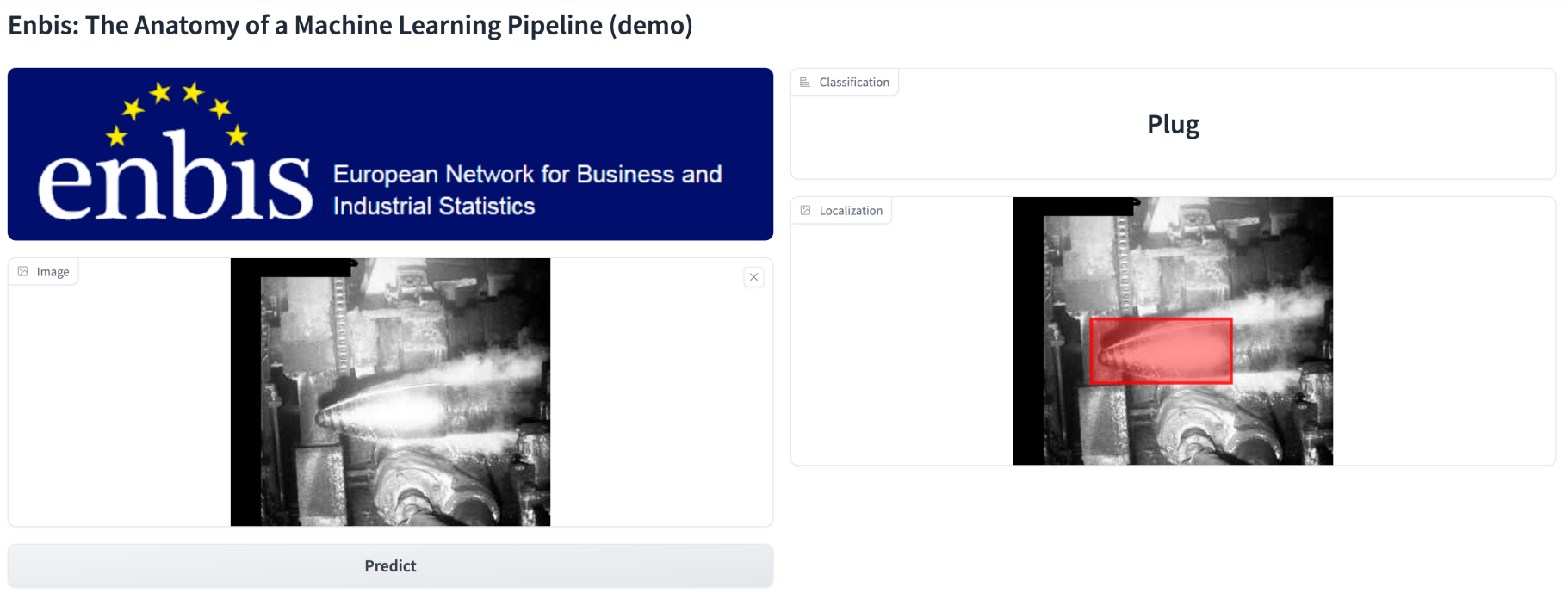
- Data ingestion and preparation: plots of the empirical distribution features and key summaries from minimal statistical models.
- Training: plots of model performance and behaviour (explainability), interactive dashboards with sliders to pick hyperparameters on the fly.
- Serving and inference: time series plots to detect data drift and loss of accuracy.

All plots should also include confidence intervals!

# Reporting: Demo Dashboard

A well-designed dashboard is essential for:

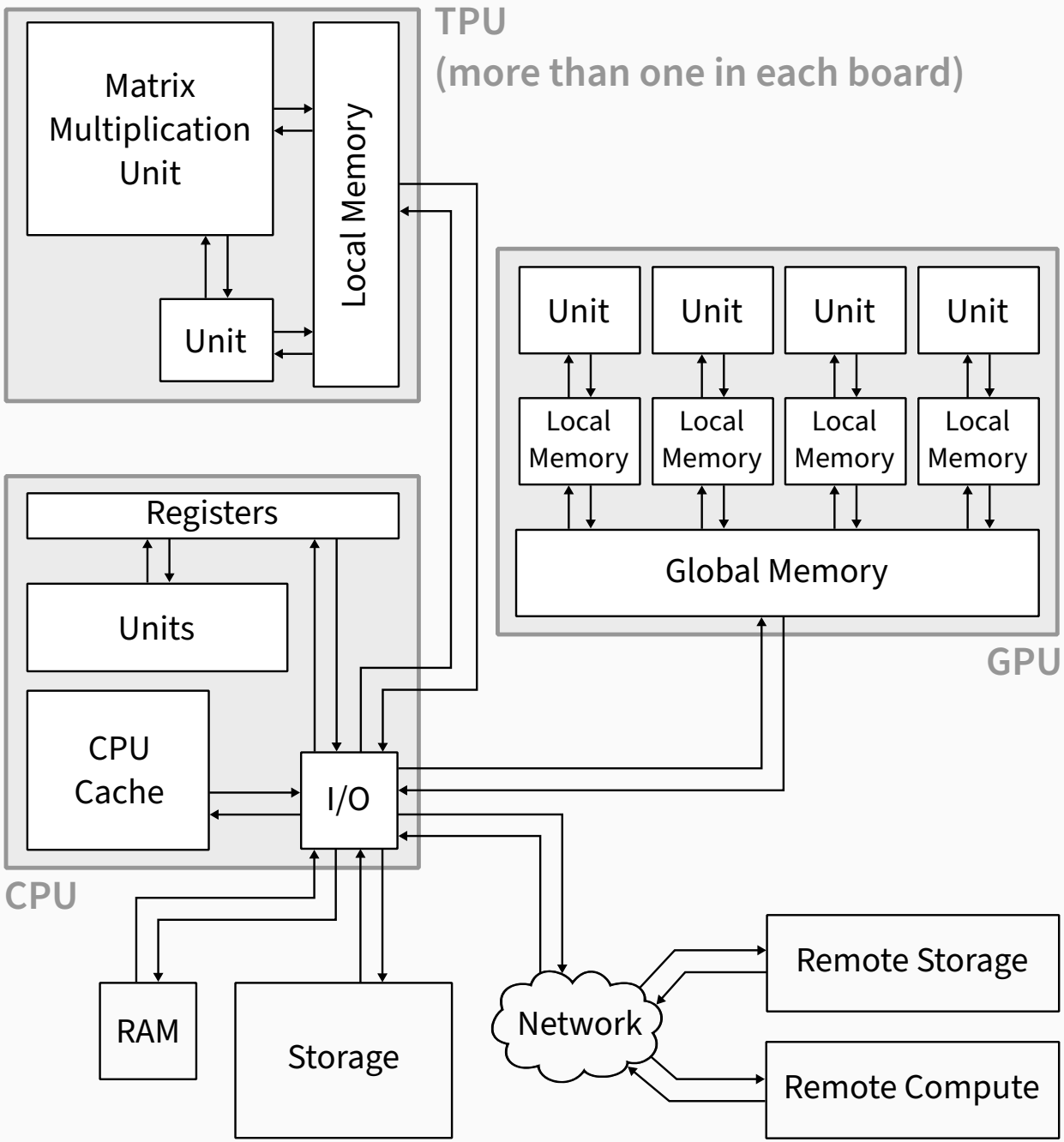
- Interactive visualization and demonstration;
- Feedback and testing.




(link)

- ✓ Course Overview
- ✓ Industrial Seamless Pipe Piercing: Plug Detection and Localisation with Computer Vision
- ✓ Machine Learning Pipeline: What Is It?
  - Project Scoping and Baseline Implementation
  - Data Ingestion and Preparation
  - Model Training and Evaluation
  - Monitoring, Logging and Reporting
- → **Assorted Trade-Offs and Best Practices**

# Hardware


TPU
(more than one in each board)
Matrix Multiplication Unit
Local Memory
Unit
Unit
Unit
Unit
Unit
Local Memory
Local Memory
Local Memory
Local Memory
Global Memory
GPU
Registers
Units
CPU Cache
I/O
CPU
RAM
Storage
Network
Remote Storage
Remote Compute

# Hardware

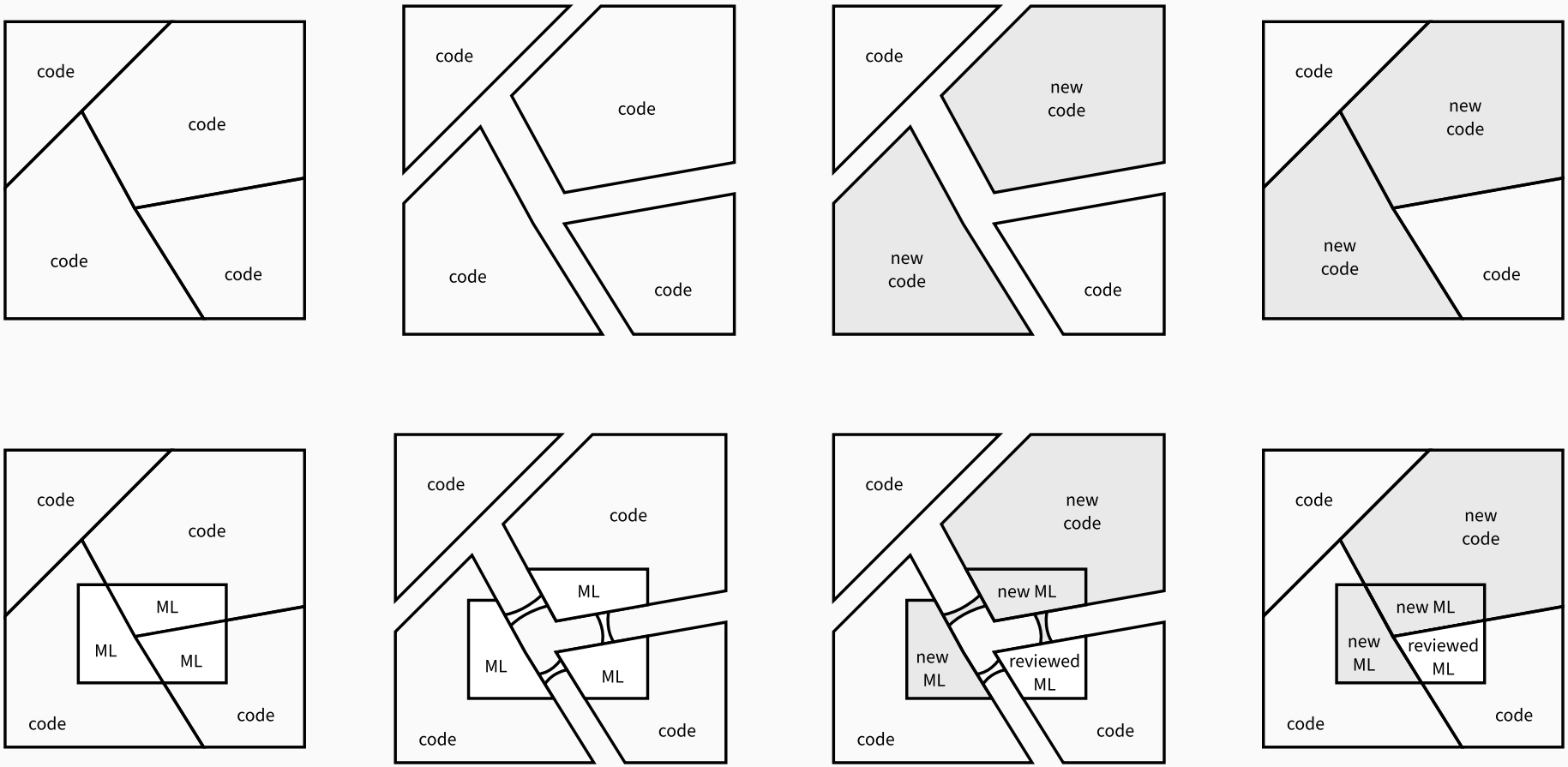
- Each model and task is better suited to particular types of hardware.
- Operational intensity: we want to keep all those processors busy as much as possible by doing things in parallel (at the instruction, data and thread level).
- Local vs remote:
  - Colocating the data and all the compute systems that will work on it to avoid large, repeated data transfers across different locations.
  - It is desirable to keep geographical spread for disaster recovery.
- Cloud computing is not a universal solution to capacity planning:
  - It may be cheaper to buy the hardware outright if we use it a lot.
  - Good at horizontal scalability but struggles with vertical scalability.
  - Cloud instances are more difficult to profile and trace.

# Programming and Code Styles

- Variable types: consider size and hardware support.
- Data structures: consider memory efficiency, what ML libraries use, and the ML algorithms access patterns.
- Algorithms: consider their complexity and how they should scale, keeping in mind that hardware and data structures can matter as much as big-$O$ notation.
- Programming language(s): consider observability and library availability.
- Coding styles and standards: consider readability and prevent code smells, including those specific to machine learning!
- Filesystem structure: follow a standard one that cleanly separates different modules, is easy to navigate and to version.

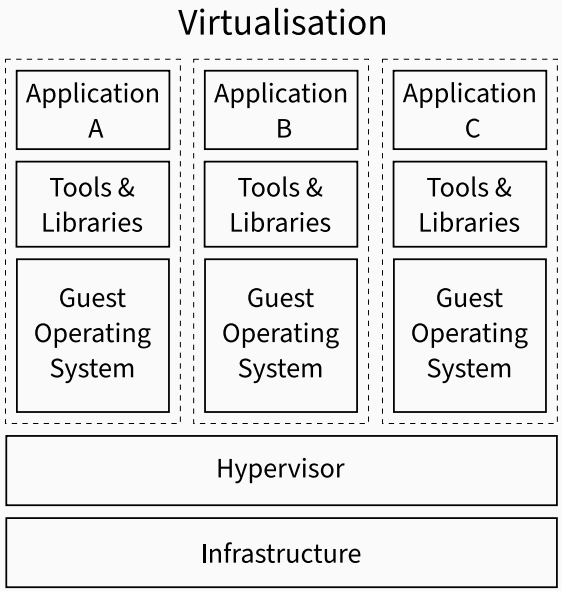
# Refactoring

We can refactor modular code and evolve it piece-wise...

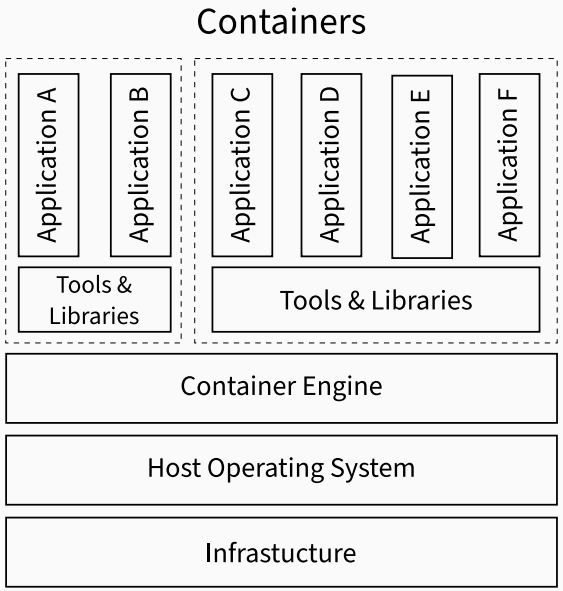



... but we have to refactor the ML models across modules to keep their mathematical and probabilistic properties consistent across the pipeline.

# How To Package




- Hardware abstraction.
- Horizontal and vertical scalability.
- Portability.




- Smaller, start faster.
- Better orchestration.
- Better MLOps integration.

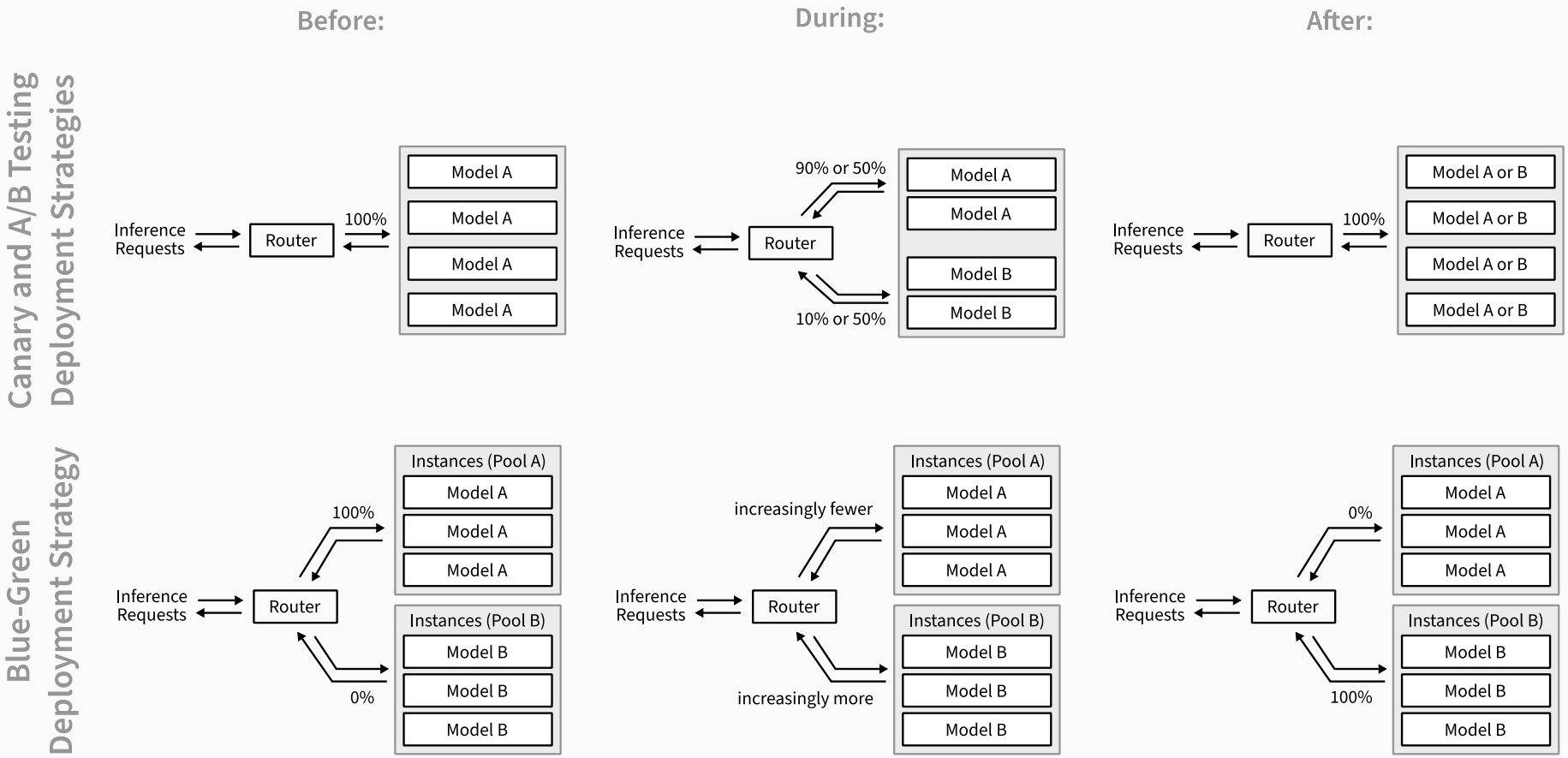
For models: we can either embed them in the software or ship them as standalone (say, ONNX) packages from a model registry.

# In Our Use Case: How To Package

To guarantee maximum portability over time:

- We export the model in ONNX and PyTorch format. ONNX facilitates framework interoperability, serving as an intermediary format for deploying trained models across diverse ML platforms.
- We wrap up the model and all software dependencies and we store them in a container environment for consistency and reproducibility.
- We store both the model and its dependencies in version control, ensuring traceability and facilitating rollback to previous versions if needed.
- Additional automated testing procedures can be established to validate the model's performance, integrating automatic metric validation within the CI/CD pipeline before container creation.

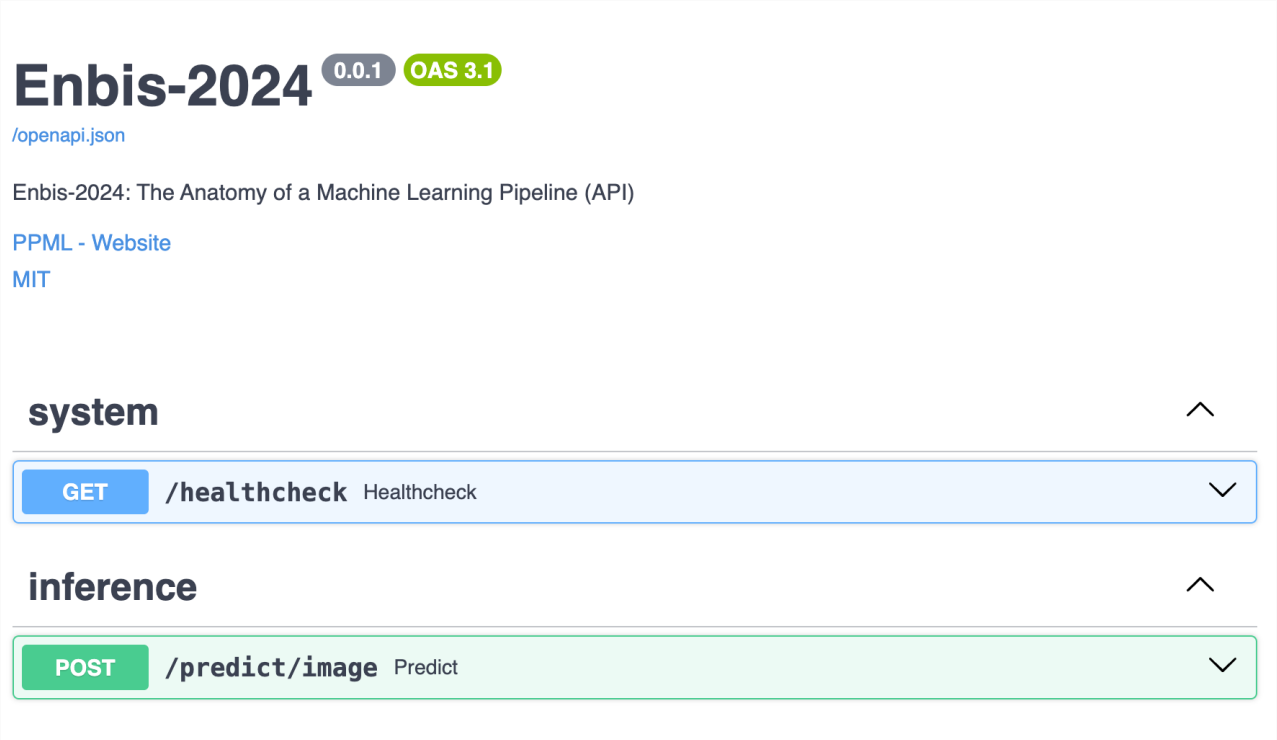
# How To Deploy and Serve




Consider availability, throughput and latency requirements, and remember to allow for rollbacks!

# In Our Use Case: How To Deploy and Serve

- Container deployment is feasible on public cloud and container orchestration platforms like Kubernetes.
- The inference endpoint is accessible through a RESTful API to ensure interoperability and flexibility.




OpenAPI specs

That's all!

Happy to discuss in more detail.