Day8 - Deployment

02476 Machine Learning Operations

Nicki Skafte Detlefsen, Associate Professor, DTU Compute

January 2026

Remember This Figure

We are now in the end part of the figure...

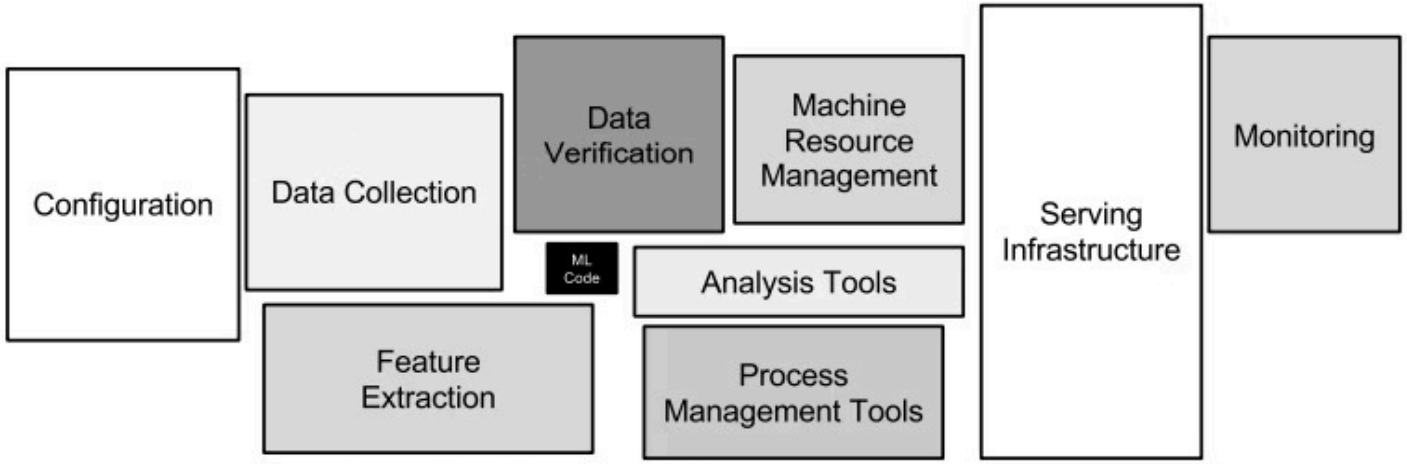


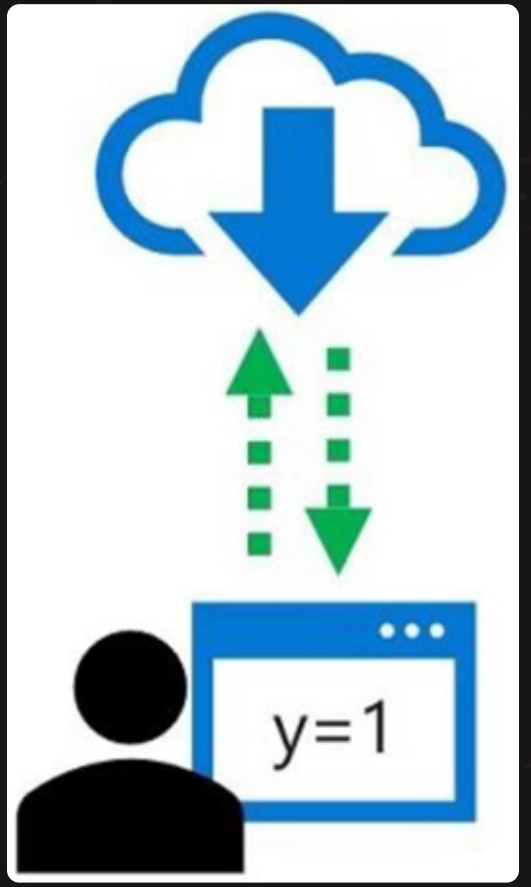
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

[1] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., & Dennison, D. (2015). Hidden Technical Debt in Machine Learning Systems. *NIPS*, 2494-2502.

Operations is Model Deployment

- 💡 In ML, **inference** refers to the use of a trained model to predict labels/generate for new data on which the model has not been trained.

- 💡 Around **80% of compute spend** in the cloud on machine learning is spent on inference.



Production Requirements

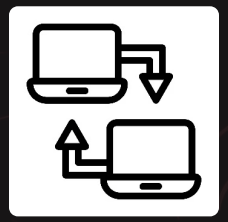
Deploying ML models to production requires careful consideration of multiple critical factors. Each requirement plays a vital role in ensuring your models perform reliably at scale.

Infrastructure

Scalable compute resources, storage solutions, and networking capabilities to handle production workloads efficiently.

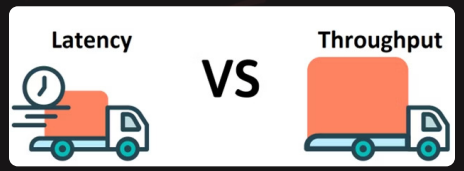
Portability

Models must run consistently across different environments—from development to staging to production.



Performance

Low latency, high throughput, and efficient resource utilization are essential for production workloads.



Stability

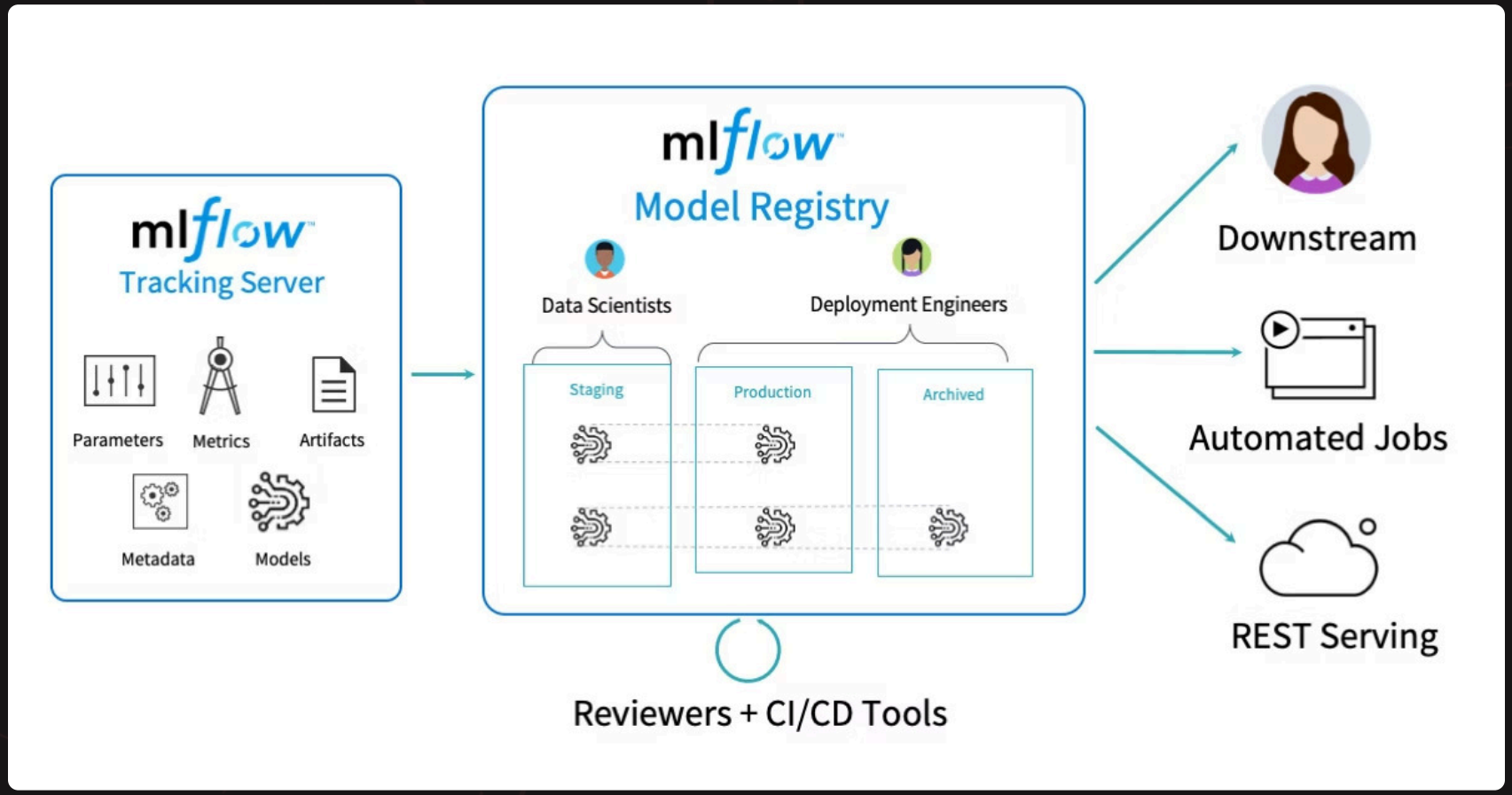
Models must maintain consistent performance over time, handling edge cases and unexpected inputs gracefully.

Security

Protect models from adversarial attacks, ensure data privacy, and maintain secure access controls.

We Start With a Model

The deployment journey begins in the staging area, where your trained model has been validated and tested.



Before Deployment: Optimize

Start by optimizing the model

01

Pruning

Remove unnecessary weights and connections to reduce model size without sacrificing accuracy.

02

Quantization

Convert high-precision weights to lower precision (e.g., FP32 to INT8) for faster inference.

03

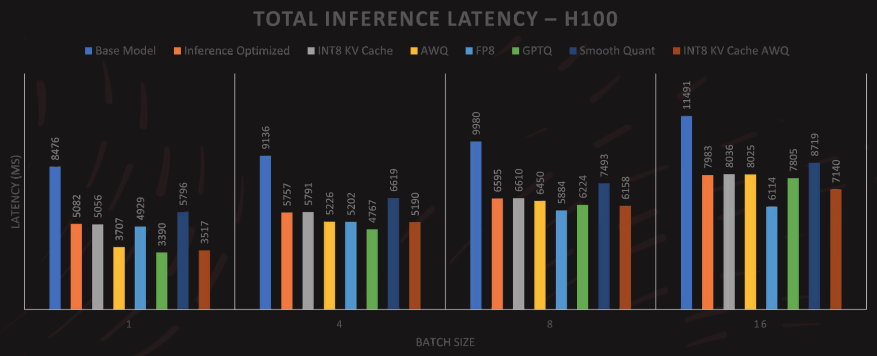
Compiling / Layer Fusion

Combine multiple operations into optimized kernels for improved execution speed.

04

Device Optimizations

Apply hardware-specific optimizations for CPUs, GPUs, or specialized accelerators.



Goal: Increase throughput, reduce memory footprint, and minimize energy consumption.

[1] **https://infohub.delltechnologies.com/en-us/p/unlocking-llm-performance-advanced-quantization-techniques-on-dell-server-configurations/**

Next Step: Model Packaging

Convert your trained model into a format optimized specifically for inference workloads. Different frameworks offer various advantages for production deployment.

TensorRT

NVIDIA's high-performance deep learning inference optimizer and runtime for GPU deployment.

ONNX

Open Neural Network Exchange—an open standard for representing machine learning models across frameworks.

GLOW

Graph-Lowering compiler designed to accelerate deep learning frameworks on various hardware.

Next Step: Inference Engine

Choosing the right inference backend can dramatically impact your model's performance in production. Different engines are optimized for specific hardware, frameworks, and deployment scenarios.

- **TorchServe** – Optimized for serving PyTorch models, with built-in support for batching, model versioning, and metrics.
- **TensorFlow Serving** – Designed for efficiently deploying and scaling TensorFlow models with support for gRPC and REST APIs.
- **Triton Inference Server** – A highly optimized multi-framework serving engine (supports PyTorch, TensorFlow, ONNX, XGBoost, etc.) with advanced features like dynamic batching, GPU acceleration, and model ensembles.
- **BentoML** – A flexible framework for packaging and deploying models from any framework into production-ready services with strong integration into MLOps pipelines.
- **Ray Serve** – A scalable model serving library built on Ray, ideal for distributed inference, model composition, and integrating with reinforcement learning or online training systems.
- **vLLM** – Specialized for serving large language models (LLMs) efficiently, using optimized attention kernels and paged memory management to maximize throughput and minimize latency.

Type	Response Time	Total Requests (Request Per Second)
Vanilla Python gRPC		17340 (43.5)
PyTorch Serve		17493 (44.9)
NVIDIA Triton Inference Server		28133 (71.9)

Next Step: API Time

We need to create an application that defines:

1. **Model Loading**

   How the model should be loaded into memory and initialized

2. **Input Specification**

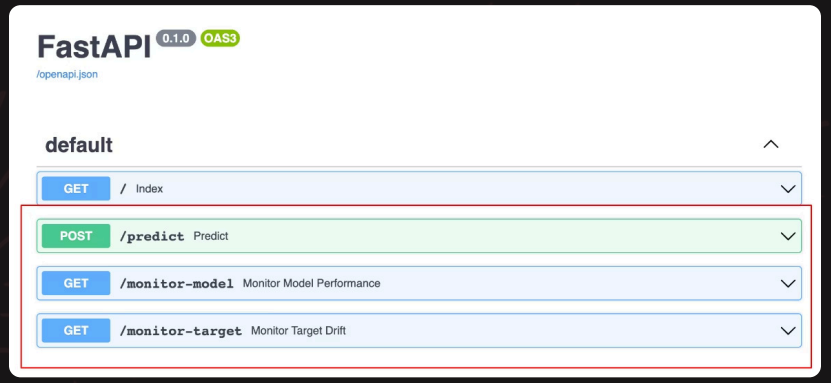
   What input format and data types the model expects

3. **Inference Logic**

   How to execute predictions with the model efficiently

4. **Output Format**

   How model outputs should be structured and returned to clients



Essentially, a machine learning model expects data (tensors) in one format but the way we send data is normally in another format (json), and the way the user expects it is a third format (visualization)

APIs are in charge of this conversion.

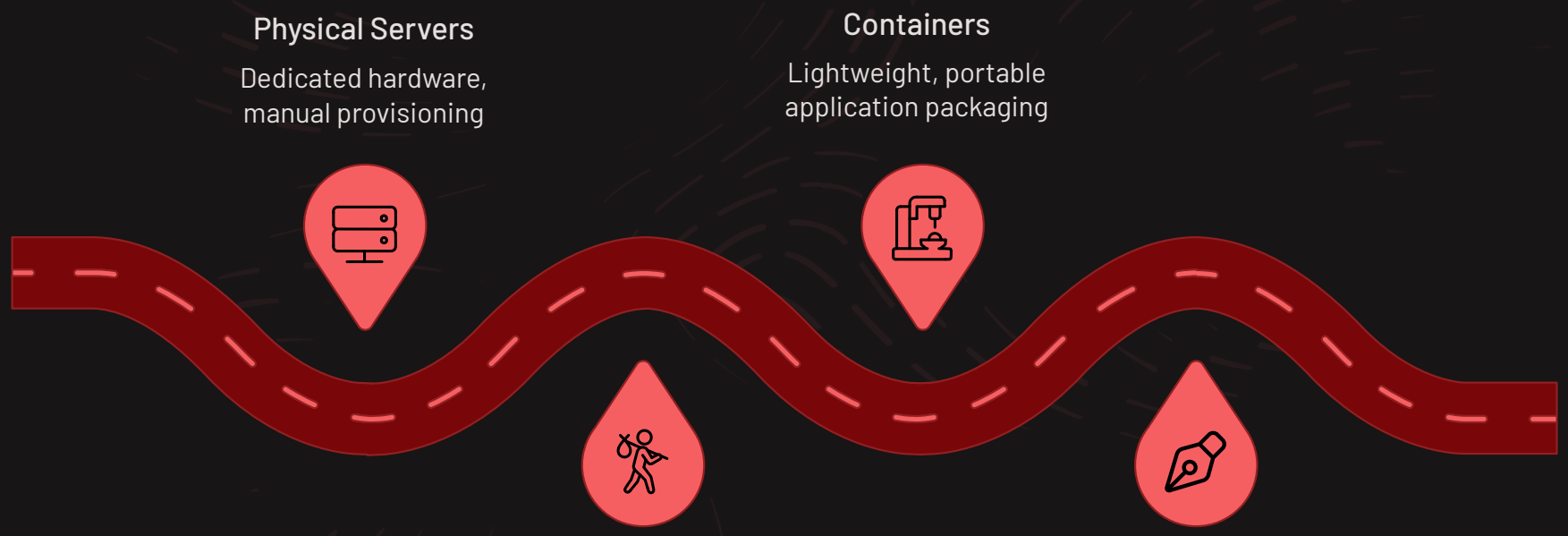
Next Step: Containerize

Containerization solves the "it works on my machine" problem by packaging your entire application environment into a portable, reproducible unit.

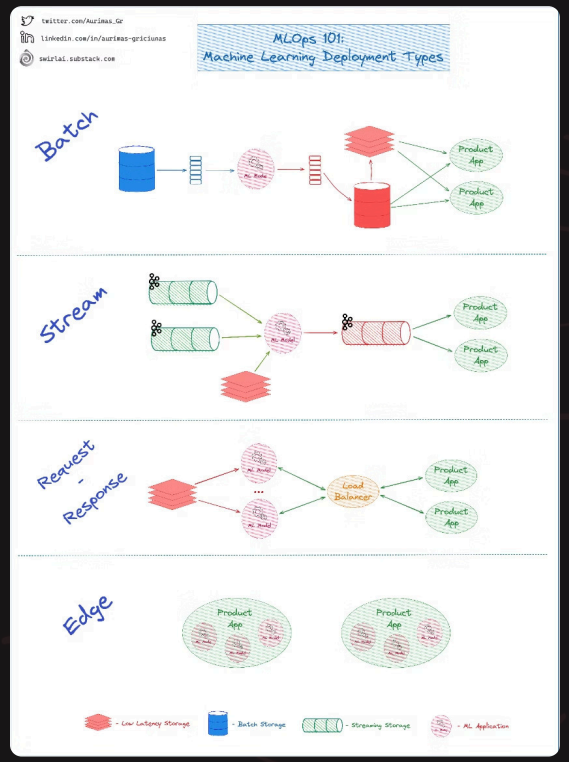

```
# system information
FROM python:3.11-slim
RUN apt update && \
    apt install --no-install-recommends -y build-essential gcc git && \
    apt clean && rm -rf /var/lib/apt/lists/*

RUN mkdir /app
WORKDIR /app

# Include relevant files
COPY requirements.txt /app/requirements.txt
COPY pyproject.toml /app/pyproject.toml
COPY src/ /app/src

# Install dependencies
RUN pip install .

# Execution statement
EXPOSE $PORT
CMD exec uvicorn src.example_mlops.app:app --host 0.0.0.0 --port $PORT
```

Side step: The Road to Modern Deployment

Deployment technology has undergone a significant evolution, driven by the need for greater efficiency, flexibility, and scalability. This journey highlights key innovations that transformed how we manage and run applications.



Deployment Types



🔥 Batch Inference

Process large volumes of data at scheduled intervals. Ideal for high throughput, high latency applications like recommendation systems or data pipelines.

🔥 Stream Processing

Continuous processing of data streams in real-time. Perfect for LLM applications, chat interfaces, and scenarios requiring progressive output generation.

🔥 Request-Response

Synchronous predictions for individual requests. The standard pattern for web services, APIs, and most real-time applications requiring immediate results.

🔥 Edge Deployment

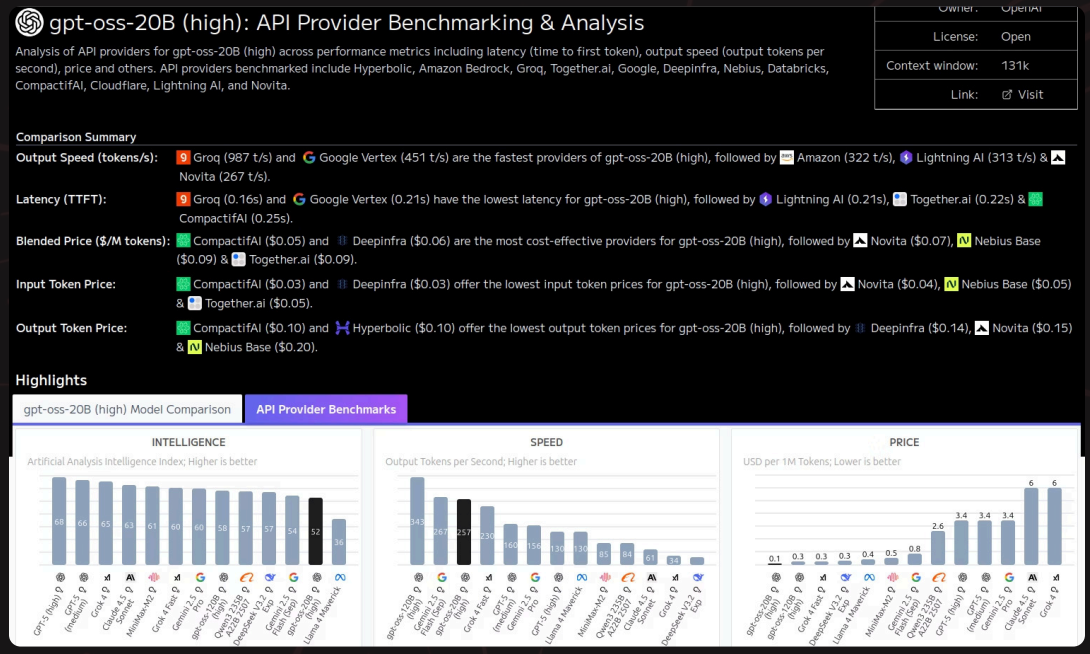
Models running directly on end-user devices. Essential for applications requiring low latency, offline capability, or data privacy (IoT, mobile apps).

Cloud Deployments

AWS SageMaker

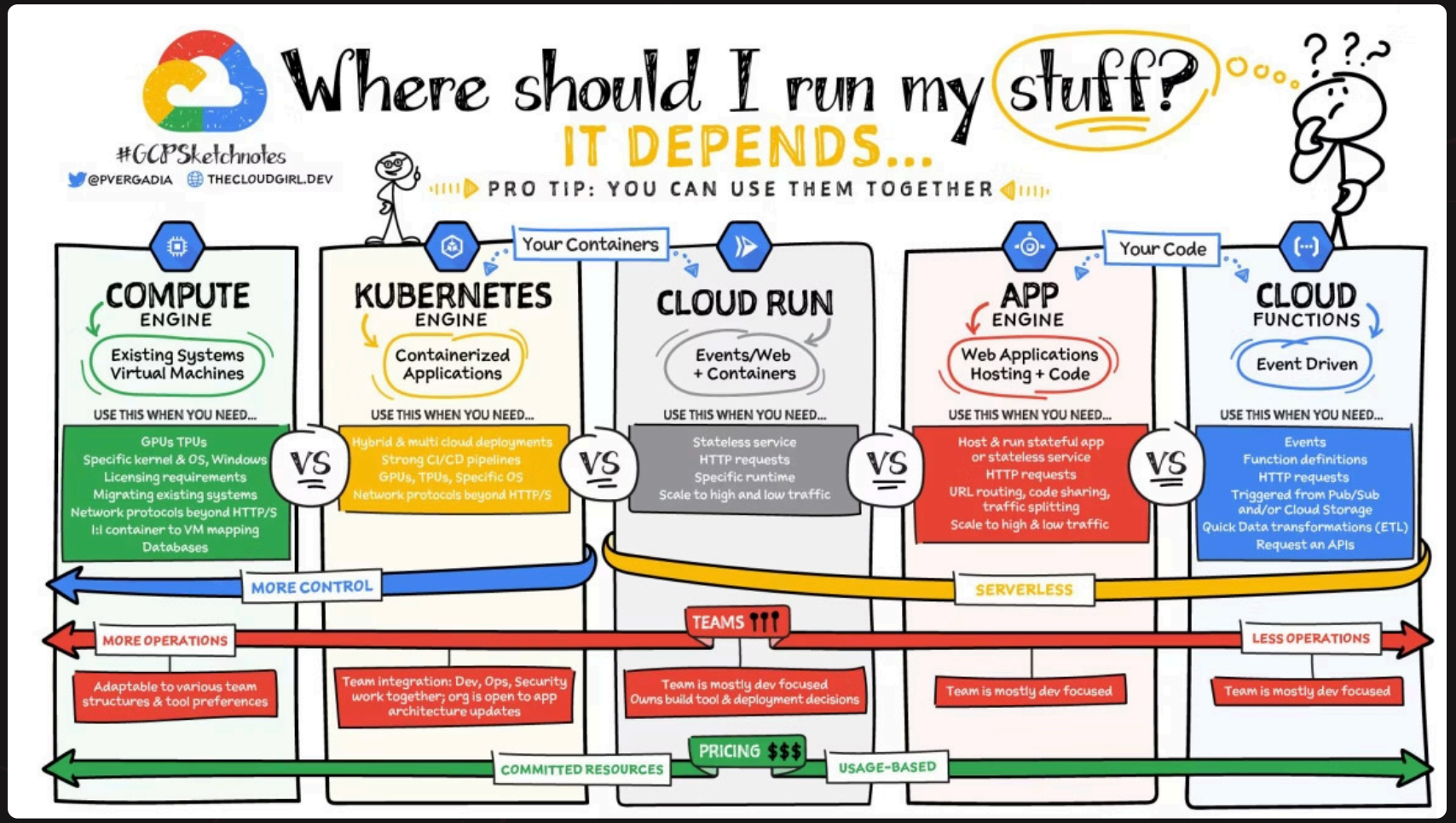
Google Vertex AI

Azure ML



[1] https://artificialanalysis.ai/

Where should I run my stuff?



Final step: Deployment Strategies

Shadow Mode

Deploy the new model alongside the old one, logging its predictions for comparison without serving them to users. Ideal for risk-averse validation.

Canary Release

Roll out the new model to a small percentage of users (e.g., 5%) and monitor closely before gradually increasing traffic. Perfect for controlled, incremental deployment.

A/B Testing

Serve different model versions to randomized user groups to directly compare their impact on business metrics. Best for quantifying performance differences.

Monolith vs. Microservices: Architectural Choices

Monolithic Architecture

Pros:

- Simpler initial development and deployment
- Easier debugging in one codebase

Cons:

- Hard to scale components independently
- Slower development with growth
- Higher risk of system-wide failures

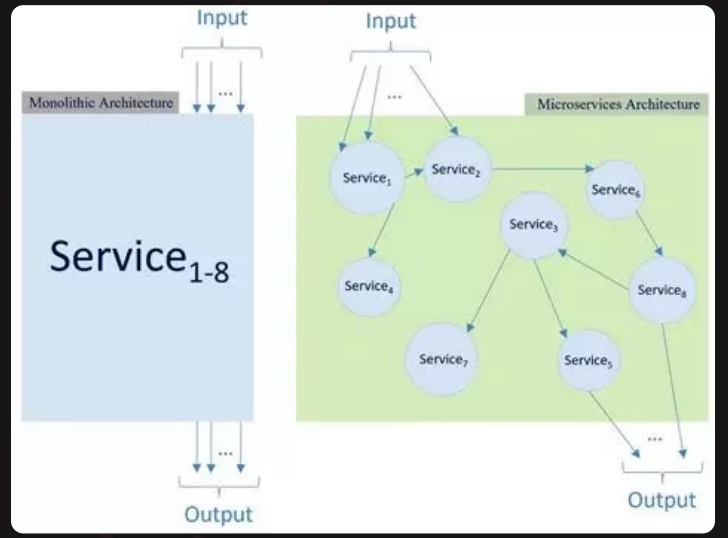
Microservices Architecture

Pros:

- Independent deployment and scaling
- Increased resilience and fault isolation
- Enables diverse tech stacks

Cons:

- Increased operational complexity
- Distributed data management challenges
- Robust inter-service communication needed



Beyond the Model: Essential Deployment Components

API Gateway

Acts as a single entry point for all API calls, handling routing, security, and traffic management before requests reach your model endpoints.

Authentication

Secures access to your model, verifying user or service identities to prevent unauthorized use and protect sensitive data.

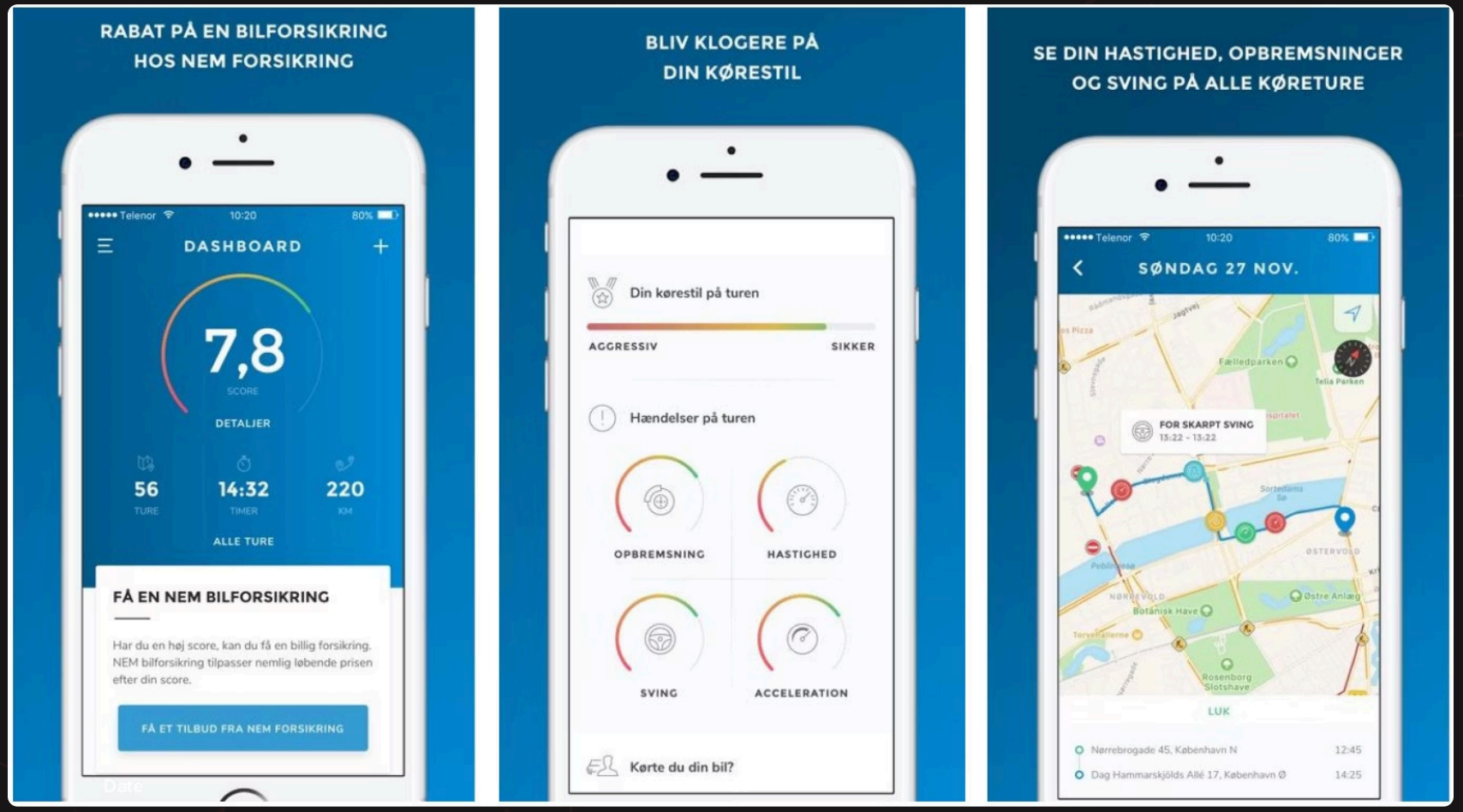
Autoscaler

Dynamically adjusts computing resources (e.g., number of model instances) based on real-time demand, ensuring performance and cost efficiency.

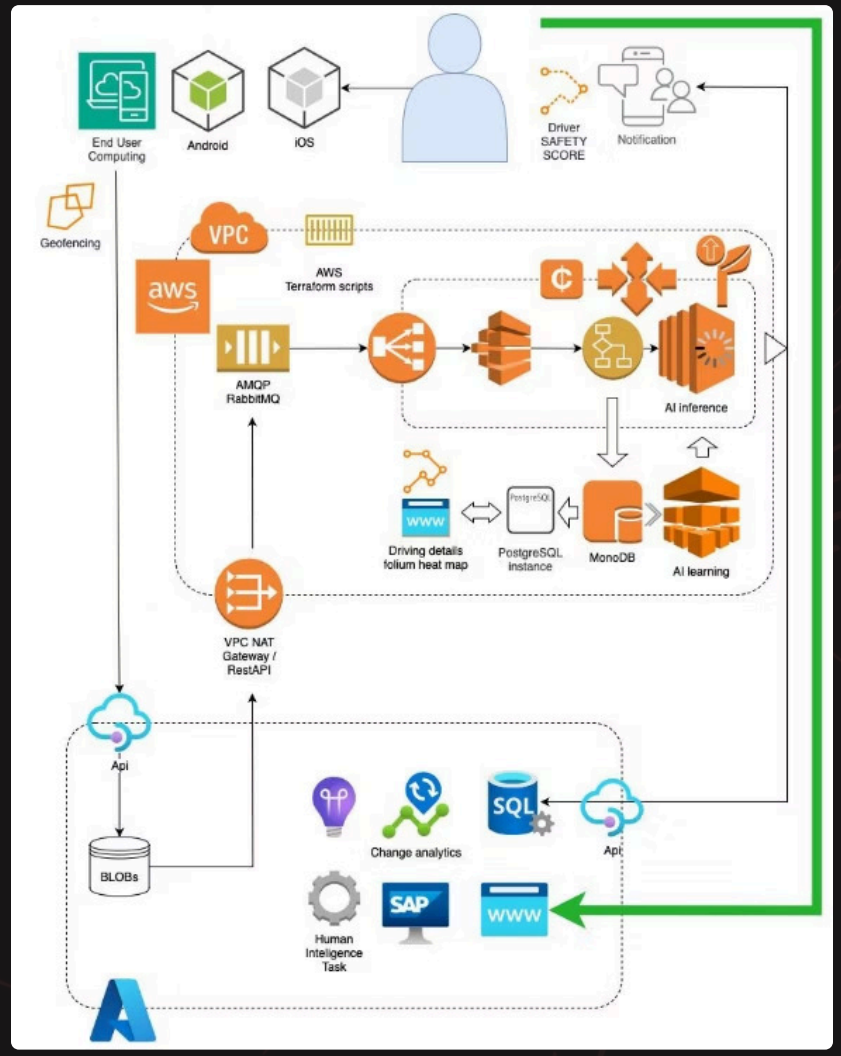
Database

Stores model inputs, outputs, monitoring data, and feature stores, crucial for logging, auditing, and continuous model improvement.

Real life example by Petr Taborsky



Real life example by Petr Taborsky



- **AMQP RabbitMQ:** Message broker for incoming data.
- **Processing Servers:** Perform initial data processing.
- **Data Processing Logic:** Formats data for the AI model.
- **AI inference:** Generates predictions using the trained AI model.
- **AI learning:** Trains the AI model using data from PostgreSQL and MongoDB.

Exercise: Choosing the Right Infrastructure

Your team has built an **AI-powered image captioning service**. It currently runs on a single GPU virtual machine, which performs well for limited traffic. After a successful marketing campaign, usage spikes to thousands of image requests per minute. Response times increase, and costs stay high because the GPU VM runs continuously, even when idle.

Group Task

Open the Excalidraw link (backup and press "*Replace my content*" if needed) and invite your group members to a collaborative session. In your group, discuss and map out a new improved infrastructure that handles the above scenario.

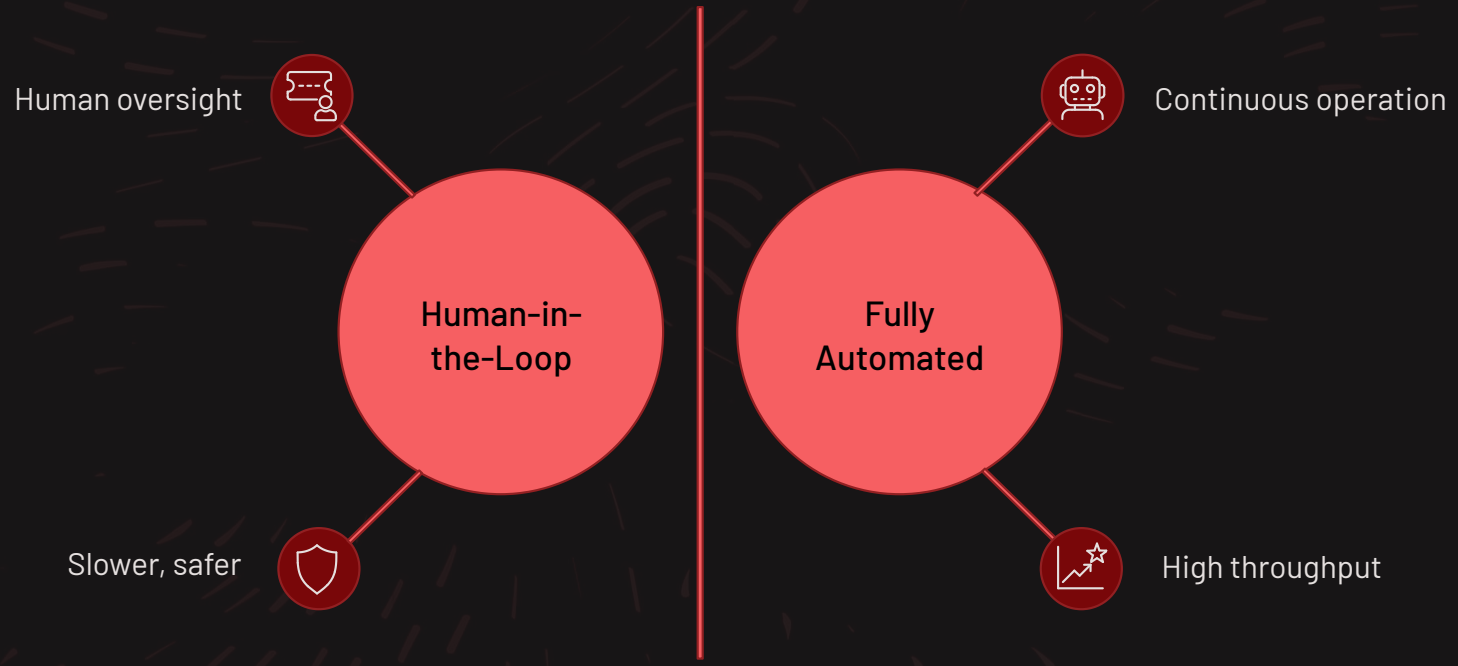
Excalidraw

Excalidraw — Collaborative whiteboarding made easy

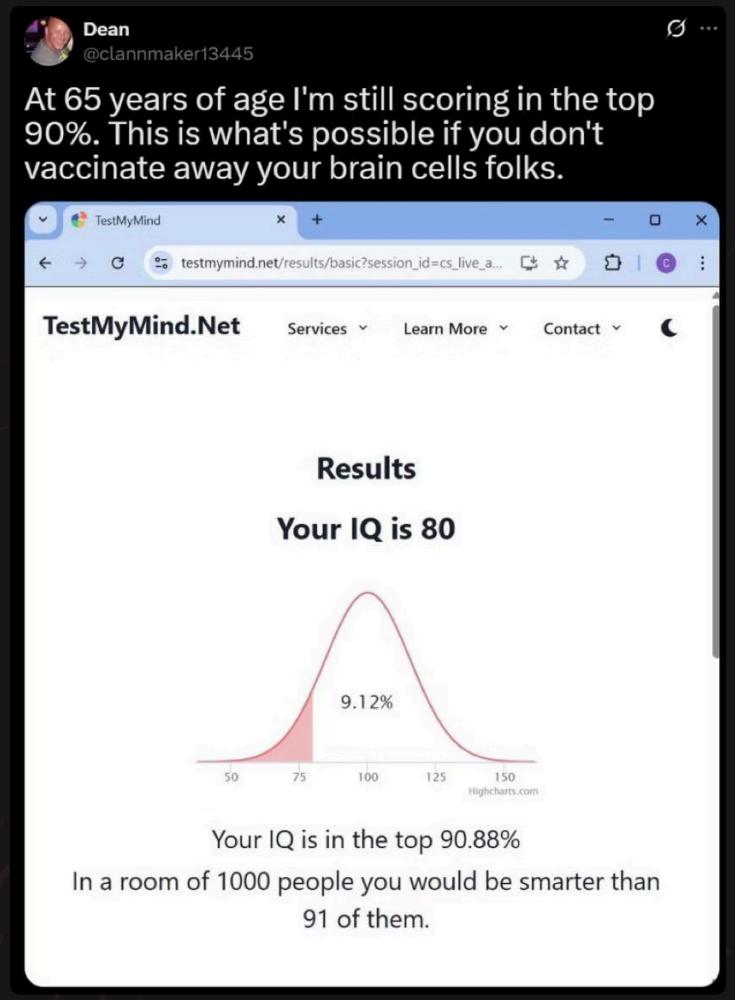
Excalidraw is a virtual collaborative whiteboard tool that lets you easily sketch diagrams that have a hand-drawn feel to them.

Taking a step back

We often think prediction == decision in automated systems. However, in the real world, human judgment often plays a critical role, leading to different deployment paradigms.



Conveying information is hard....



Same model. Different decisions.

A single machine learning model can produce various types of output, each influencing how decision-makers perceive and act upon the information.

Regression

Predicted demand: **87 units (±12)**

Focuses on numerical prediction with a confidence interval.

Forecast

Sales curve over 6 months with **80% confidence band**

Highlights future trends and associated uncertainty.

Classification

Fraud risk: **72%**

Provides a probability score for a specific category or outcome.

Generation

Text summary of medical note

Creates new content, often synthesizing complex information.

> What should we show to the decision maker? The number, the uncertainty, or the narrative?

The form of model output changes how people and systems act.

Different Models, Different Communication Strategies

Model Type	Typical Output	Presentation Options	Decision Implications
Classification	Discrete label / probabilities	Top class, top-k, probability dist.	Hides or exposes uncertainty
Regression	Continuous value	Point estimate, interval, distribution	Affects perceived precision
Forecasting	Time series of future values	Mean, confidence band, scenarios	Influences planning & buffers
Ranking / Recommendation	Ordered list	Top-N items, scores, rationale	Changes exposure or fairness
Clustering / Embedding	Group assignments, latent vectors	Cluster labels, visual map	Changes interpretability
Generative	Text, image, plan	1 sample vs multiple candidates	Affects creativity vs. control
RL / Policy models	Action suggestions	Action, policy heatmap, reward estimate	Determines autonomy of the system

Summaries vs. Signals vs. Stories

The way model outputs are presented significantly influences how they are interpreted and acted upon by decision-makers. Choosing the right representation is crucial for effective communication.

Representation	Example	Strength	Risk
Summary	“Demand = 87 units”	Simple, fast decisions	Ignores uncertainty
Signal	“Demand = 87 ± 12, confidence 80%”	Transparent	Slower, needs literacy
Story	“Demand likely stable, but risk of shortage in Q4”	Actionable context	Requires human interpretation

Who Consumes the Output?

Information must fit the decision-maker.

Consumer	Needs	Example
Automated System	Thresholded, numeric, fast	“If P>0.8 → alert”
Operator / Analyst	Confidence, visual trends	“Show top 3 forecasts + uncertainty”
Executive / Policy Maker	Scenarios, narratives	“3 demand futures: optimistic, base, conservative”

Discussion Exercise: "What Was the Model Really Saying?"

Scenario: A logistics company uses an AI model to predict the likelihood of delivery delays. The system outputs a probability score (e.g. 0.82) for each shipment, but the dashboard only displays "Low / Medium / High Risk" labels based on fixed thresholds. One week, a large shipment labeled "Medium Risk" arrives five days late, causing major financial penalties. Later analysis shows the model had actually estimated an 82% probability of delay — but the "Medium" category covered all predictions between 40% and 85%.

Think

- Who made the critical error — the model, or the way it was presented?
- What assumptions did users make about the label "Medium"?

Pair

- How could this have been prevented through better presentation or interface design?
- What information should the logistics manager have seen to make a sound decision?
- When should we simplify model outputs, and when does that become dangerous?

Options to Debate: Who bears responsibility?

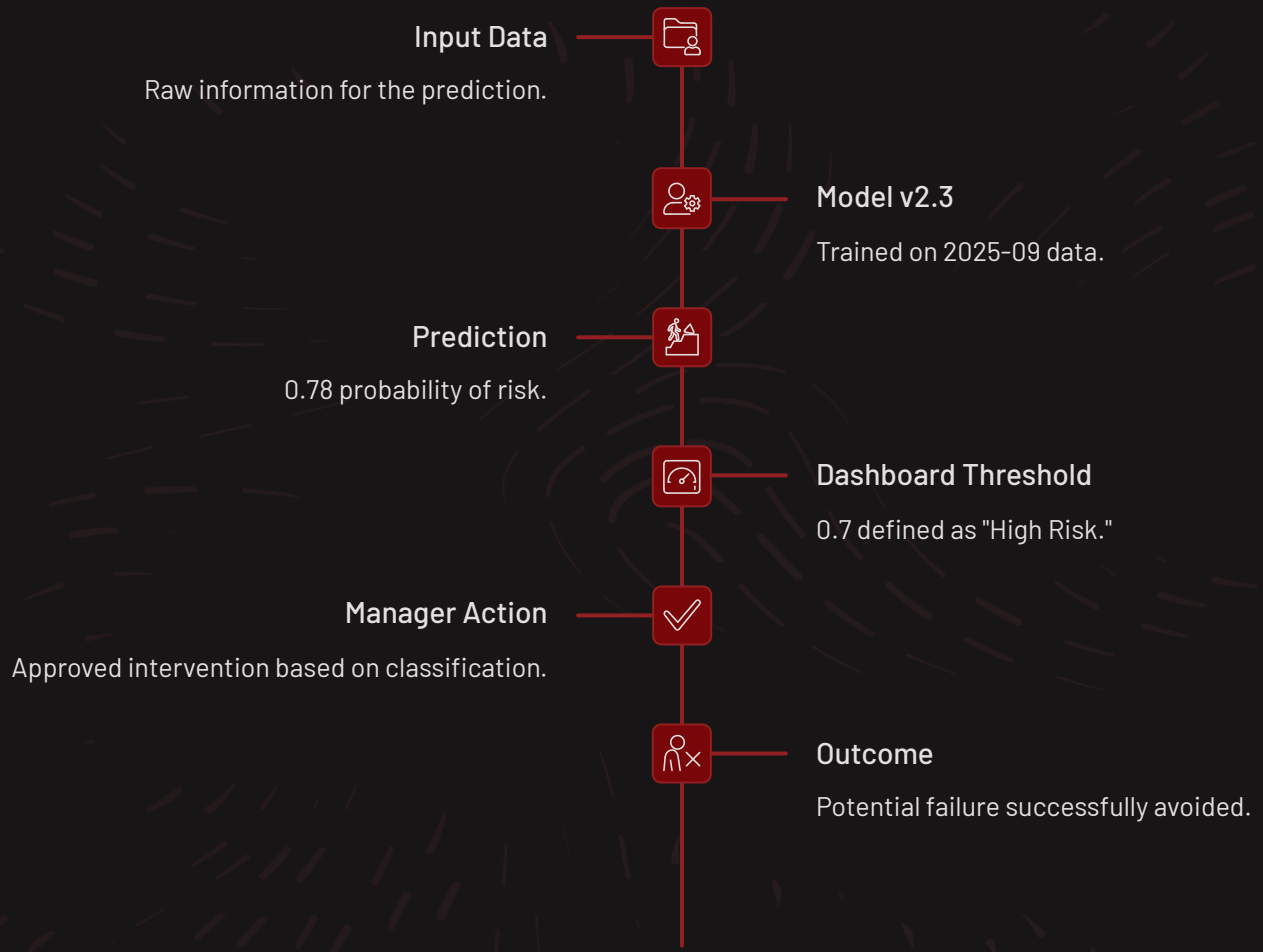
Data Scientist For not calibrating thresholds or explaining probabilities clearly?	**Product Designer** For oversimplifying model outputs for the dashboard?
Operations Manager For acting on vague categories without question?	**Company Leadership** For prioritizing simplicity over accuracy?

"A decision you can’t explain today becomes a liability tomorrow."

Machine learning decisions don’t end when predictions are made — they live on in audits, reviews, and accountability.

What	Why it matters	How to enable it
Decision traceability	You need to reconstruct *what was predicted, by which version, with what data.*	Keep versioned model logs, data lineage, and config history.
Interpretability record	You must justify *why* a model made a given prediction.	Store feature attributions or SHAP values per inference.
Human–AI decision logs	You must understand *who acted on what, and how.*	Record user actions, thresholds, and overrides linked to model outputs.

Deployment trace



Before deploying, ask:

01

Who is the output for — system, analyst, or executive?

02

Is uncertainty visible and interpretable?

03

Should we present multiple scenarios, not one number?

04

What's the latency vs. interpretability trade-off?

05

How will decisions be audited or explained later?

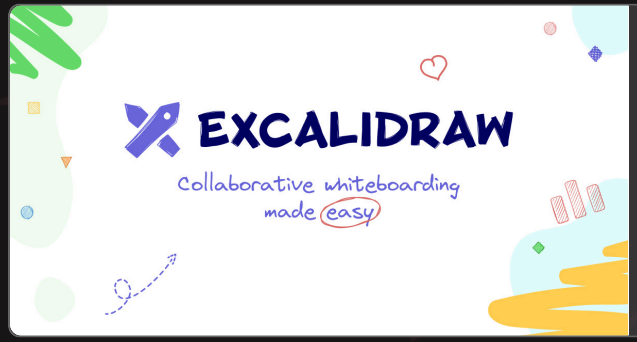
> [Model Computation] → [Information Representation] → [Decision Context] → [Outcome]

Exercise: Automation vs Human Oversight in AI Operations

Your company has deployed an **AI model that detects skin cancer from medical images**. The system automatically ingests new data, retrains weekly, and redeploys models whenever performance metrics improve. After several months, doctors report that the model is missing rare cancer types - but the monitoring dashboard still shows “healthy” performance. You suspect that automation worked *as designed*, but without **human auditing**, key issues went unnoticed.

Group Task

Open the Excalidraw link (backup and press "*Replace my content*" if needed) and invite your group members to a collaborative session. In your group, complete a monitoring and auditing loop by placing actors - either automated systems or humans - at key checkpoints in the AI lifecycle. They’ll decide who does what and where human oversight adds value.



Excalidraw

Excalidraw — Collaborative whiteboarding made easy

Excalidraw is a virtual collaborative whiteboard tool that lets you easily sketch diagrams that have a hand-drawn feel to them.

Meme of the day