



# ML System Design

MLOps | Requirements, offline vs online, batch vs real-time inference, latency/cost/quality trade-offs, capacity with Little's law, and a full design-interview walk-through (card fraud, feed ranking)

#MLOps #SystemDesign #Inference #Interview

## Start from the problem



**WHERE THE ANALOGY BREAKS** A kitchen sees immediately when a plate is wrong; an ML system can serve wrong answers for weeks because its labels arrive late or never.

Designing an ML system is like designing a restaurant kitchen before choosing the chef

### Prerequisites

Train/validation/test splits and metrics such as precision, recall, AUC (machine-learning folder); HTTP APIs and containers; percentiles. Serving runtimes are in sheet 04, feature stores in sheet 06: here they are only boxes on the diagram.

- 1 Business goal and metric** what decision does the prediction drive, and which number moves if it is right (loss avoided, clicks, minutes saved)?
- 2 Is ML needed at all?** a rule, a SQL threshold or a lookup may reach 80 % of the value; it is also the baseline and the fallback
- 3 Prediction timing** when is the answer needed: in 50 ms during a request, within a minute of an event, or by tomorrow 08:00?
- 4 Scale** requests per second at peak, number of entities (users, items), data volume per day
- 5 Labels** where do they come from, how late do they arrive, are they biased by the current system?
- 6 Cost of errors** false positive vs false negative cost; they set the decision threshold, not the model
- 7 Constraints** privacy (personal data, GDPR), explainability (credit decisions), regulation (EU AI Act high-risk), on-device

The requirement questions to ask first, in an interview or a kick-off meeting

### YOU MAY HEAR

"Start by choosing the model architecture"

BUSTED

### ACTUALLY

**Start with the decision, the metric and the latency budget**

The architecture follows from timing, data and cost of errors; a GBDT with good features beats a transformer with poor ones on most tabular problems.

### YOU MAY HEAR

"Optimise accuracy, then worry about latency"

BUSTED

### ACTUALLY

**Latency and cost are design inputs**

A model that misses its p99 budget is unusable whatever its AUC; size the model to the budget from day one.

### YOU MAY HEAR

"Real-time is always better"

BUSTED

### ACTUALLY

**Batch is cheaper, simpler and often enough**

If inputs change daily (churn, credit limits, recommendations email), nightly scores in a table serve millions of lookups for almost nothing.

Design beliefs interviewers and production both punish

## Where and when predictions are made

| Mode                       | How it works                                                  | Latency | Good for                                     | Watch out                                                         |
|----------------------------|---------------------------------------------------------------|---------|----------------------------------------------|-------------------------------------------------------------------|
| Batch                      | scheduled job scores all entities, writes to a table or cache | hours   | churn, lead scoring, nightly recommendations | stale for new users (cold start), wasted scores for inactive ones |
| Streaming (near real-time) | consumer reads events, scores, emits to a topic               | seconds | anomaly alerts, content moderation queues    | ordering, exactly-once, backpressure                              |
| Online (synchronous)       | service scores each request while the caller waits            | ms to s | fraud at authorisation, search ranking, chat | tail latency, availability, cost per request                      |

Three inference modes. Most real systems combine them: batch for heavy features, online for the final score

### Offline (batch) features

aggregates over days, computed by a warehouse job

### Online (real-time) features

computed at request time or from a stream in seconds

vs

Tell: How fresh must the value be when scored?

### Offline evaluation

metrics on held-out historical data

### Online evaluation

metrics on live traffic: canary, A/B

vs

Tell: Is real traffic involved?

### Precomputed prediction

looked up by key, O(1) at serve time

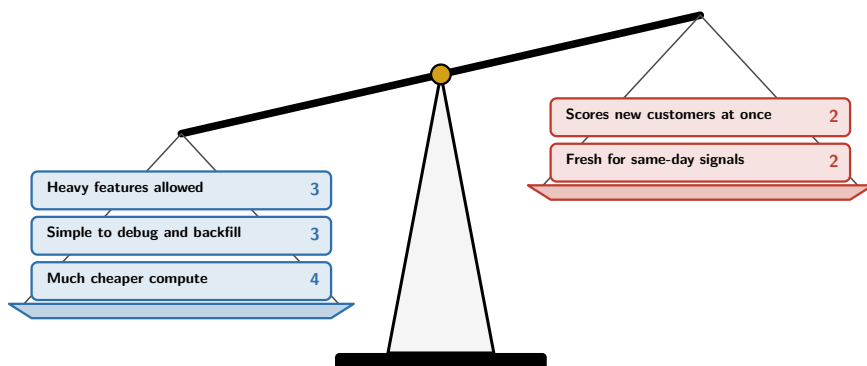
### On-demand prediction

model runs per request

vs

Tell: Is the input known in advance?

Offline vs online is about two different things: where features are computed and when the prediction is made



Batch nightly

weight 10

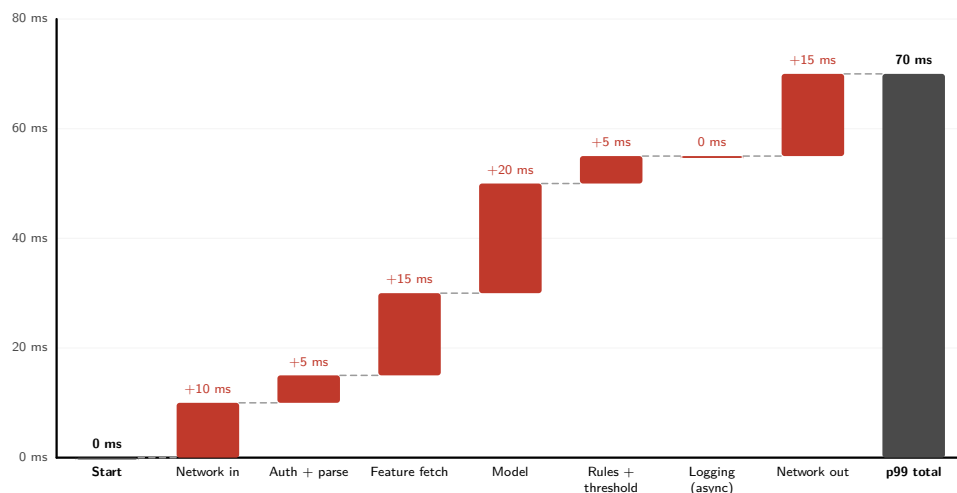
Online per request

weight 4

Batch nightly outweighs (10 vs 4)

Batch vs online for a 'customers likely to churn' list used by the sales team each morning: batch wins clearly (weights = importance for this use case, 1-5)

## ⚡ Latency, cost and quality



Latency budget of an online fraud check (illustrative p99 values in ms). The sum must fit the budget the payment network gives you; here 70 ms of a 100 ms budget, leaving 30 ms headroom

## Little's law (concurrency)

FORMULA  $L = \lambda \times W$

WHERE

- $L$  = average number of requests in the system at once (concurrency) (requests)
- $\lambda$  = arrival rate (requests per second)
- $W$  = average time a request spends in the system (latency) (seconds)

WORKED

$\lambda = 2000$   
 $W = 0.05$   
 $L = \lambda \times 0.05$   
 $L = 100$  requests in flight

RESULT  $L = 100$  requests in flight

READ IT At 2,000 requests/s and 50 ms mean latency, 100 requests are in flight; with 8 concurrent requests per replica you need at least 13 replicas, more for peaks and headroom (aim for 50-70 % utilisation).

MISTAKES

- Use the mean latency for  $W$ , then size for the p99 separately
- Latency grows sharply as utilisation nears 100 % (queueing); never size to 100 %

Capacity sizing with Little's law: how many requests are in flight, hence how many workers or GPU slots you need

| Lever                         | Gains            | Costs                                 |
|-------------------------------|------------------|---------------------------------------|
| Bigger model                  | quality          | latency, cost per request             |
| Precompute in batch           | latency, cost    | freshness, wasted scores              |
| Cache predictions or features | latency, cost    | staleness, memory, invalidation work  |
| Two-stage ranking             | latency at scale | recall lost in the first stage        |
| Quantise / distil (sheet 08)  | latency, cost    | a little quality; needs re-evaluation |
| Request batching on GPU       | throughput, cost | added queueing latency                |
| Rule fallback                 | availability     | quality during the fallback           |

Levers and what they cost. Every lever buys one property with another

## ✓ Design interview walk-through: card fraud

### PROBLEM

Design a system that decides, during card authorisation, whether to approve, decline or challenge (3-D Secure) each payment.

#### 1. Clarify and set metrics

⇒business: fraud loss + false-decline rate; model: recall at a fixed false-positive rate, PR-AUC  
why: with 0.1 % positives, accuracy is useless (99.9 % by approving everything)

#### 2. Choose the mode

⇒online synchronous scoring; heavy aggregates precomputed  
why: the decision is needed before the authorisation response

#### 3. Data and labels

⇒transactions, card, merchant, device; label = chargeback within N days  
why: fix the label window, accept delay; blocked payments get no label (feedback loop)

#### 4. Features

⇒velocity: count and sum per card in 1 h / 24 h; amount vs card average; new device; distance from last location; merchant risk  
why: streaming aggregates in an online store, same definitions offline (sheet 06)

#### 5. Model

⇒rules baseline → gradient-boosted trees; threshold per segment  
why: tabular data, fast inference (< 5 ms), explainable reasons for analysts

#### 6. Offline evaluation

⇒time-based split; replay past month; cost-weighted metric  
why: a random split leaks future patterns into training

#### 7. Serving

⇒stateless scorer behind the gateway, features from Redis-like store, rule fallback on timeout  
why: availability matters more than the last point of AUC

#### 8. Release

⇒shadow 2 weeks, then canary by merchant segment  
why: compare decisions without customer impact first (sheet 04)

#### 9. Monitoring and retraining

⇒PSI on key features, decline rate, analyst-confirmed precision; retrain weekly or on drift  
why: fraudsters adapt; labels arrive late (sheet 05)

#### 10. Exploration for the feedback loop

⇒approve a tiny random share of borderline cases, weight them in training  
why: otherwise the model never learns about what it blocks

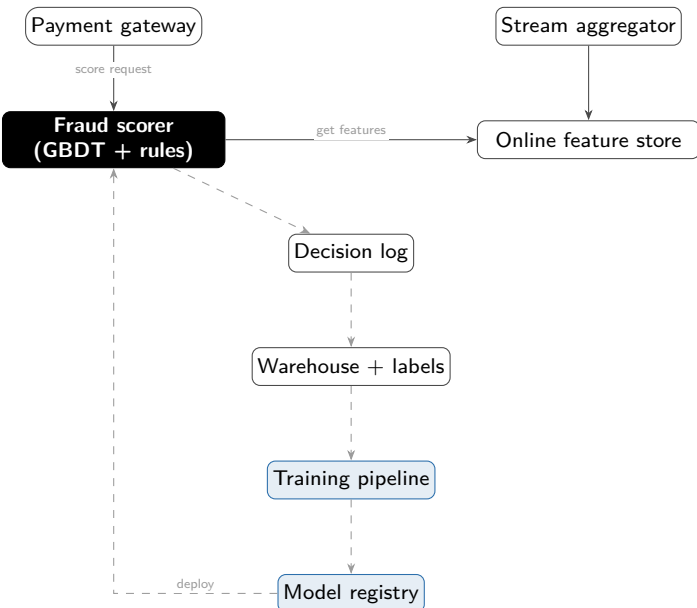
### GIVEN

- Peak 2,000 transactions/s; p99 budget 100 ms for our part
- Fraud rate about 0.1 %: 1 fraud per 1,000 payments
- Labels: chargebacks and analyst reviews, arriving days to months later
- Cost: a missed fraud loses its amount; a false decline loses the sale and annoys the customer

### ANSWER

Online GBDT scorer with streaming velocity features, rule fallback, shadow-then-canary release, drift and delayed-label monitoring, weekly retraining

Check: Every box on the diagram traces back to a requirement: latency → precomputed features and small model; 0.1 % positives → PR-AUC; delayed labels → proxy monitoring



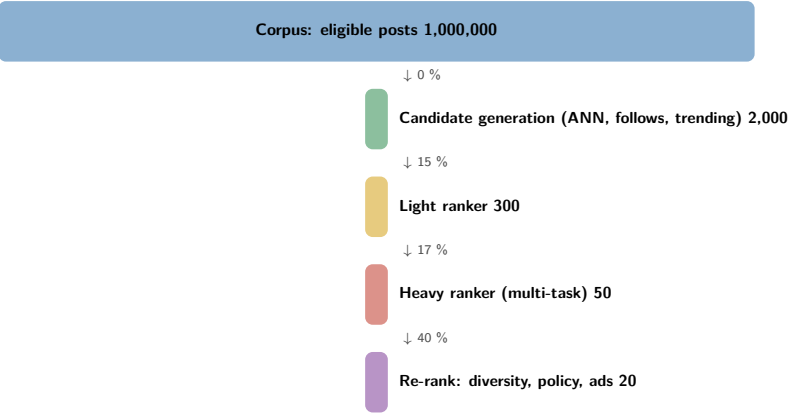
The resulting architecture: the online path (solid) must fit the latency budget, the learning path (dashed) runs offline

### Ⓜ Follow-up: 'Your fraud rate doubled overnight. What do you check?'

First, whether it is real: a data pipeline bug (a feature stuck at zero, a schema change) is more likely than a fraud wave. Compare feature distributions with yesterday (PSI), check the share of fallback decisions and null features, then look at segments (one merchant, one country, one BIN range). Only then retrain or adjust thresholds.

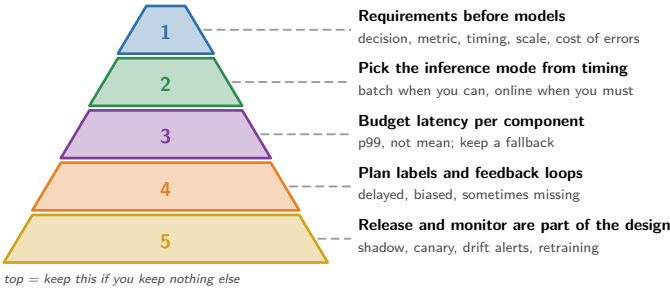
A 45-minute design answer, step by step. Figures are illustrative assumptions you state aloud, not industry facts

★ Second walk-through: feed ranking



Multi-stage ranking funnel for a social feed (illustrative counts per request): each stage uses a costlier model on fewer items

|                      |                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Objective            | a weighted sum of predicted engagement probabilities (click, like, share, dwell > 10 s) minus negative signals (hide, report); the weights are a product decision |
| Labels               | implicit feedback from impressions; position bias means top slots get clicks regardless, so log position and correct for it                                       |
| Candidate generation | two-tower embeddings with an approximate nearest neighbour index (algorithms folder) plus social graph and fresh content                                          |
| Freshness            | user embedding updated in batch daily, session features online; new posts need a cold-start path                                                                  |
| Evaluation           | offline NDCG / recall@k on logged data, then A/B test on session time and retention; interleaving for faster ranker comparisons                                   |
| Guardrail metrics    | reports, hides, creator diversity, latency p99; a ranker that raises clicks but raises reports does not ship                                                      |



What to remember from this sheet, most important at the top

✓ Self-test

- Q1 Why is accuracy a bad metric for a fraud model with 0.1 % positives?

A Approving everything scores 99.9 % accuracy; use recall at a fixed false-positive rate, PR-AUC or a cost-weighted metric.
- Q2 Batch, streaming, online: which for a nightly churn list?

A Batch: inputs change daily and nobody waits for the score.
- Q3 Apply Little’s law: 500 req/s at 200 ms mean latency?

A  $L = 500 \times 0.2 = 100$  requests in flight.
- Q4 Why size latency with p99 and not the mean?

A Users and upstream timeouts feel the tail; a good mean can hide 1 % of requests timing out.
- Q5 Why a time-based split for fraud evaluation?

A Fraud patterns evolve; a random split lets the model see the future and overstates its score.
- Q6 What is the feedback-loop problem in fraud?

A Declined payments never get a fraud label, so the model cannot learn whether it was right; keep a small exploration sample.
- Q7 Why two-stage ranking?

A The heavy model cannot score a million items per request; cheap retrieval cuts candidates to hundreds first.
- Q8 Name three fallbacks for an online model.

A Rules, last cached score, a simpler model (or a safe default decision) when the model times out.