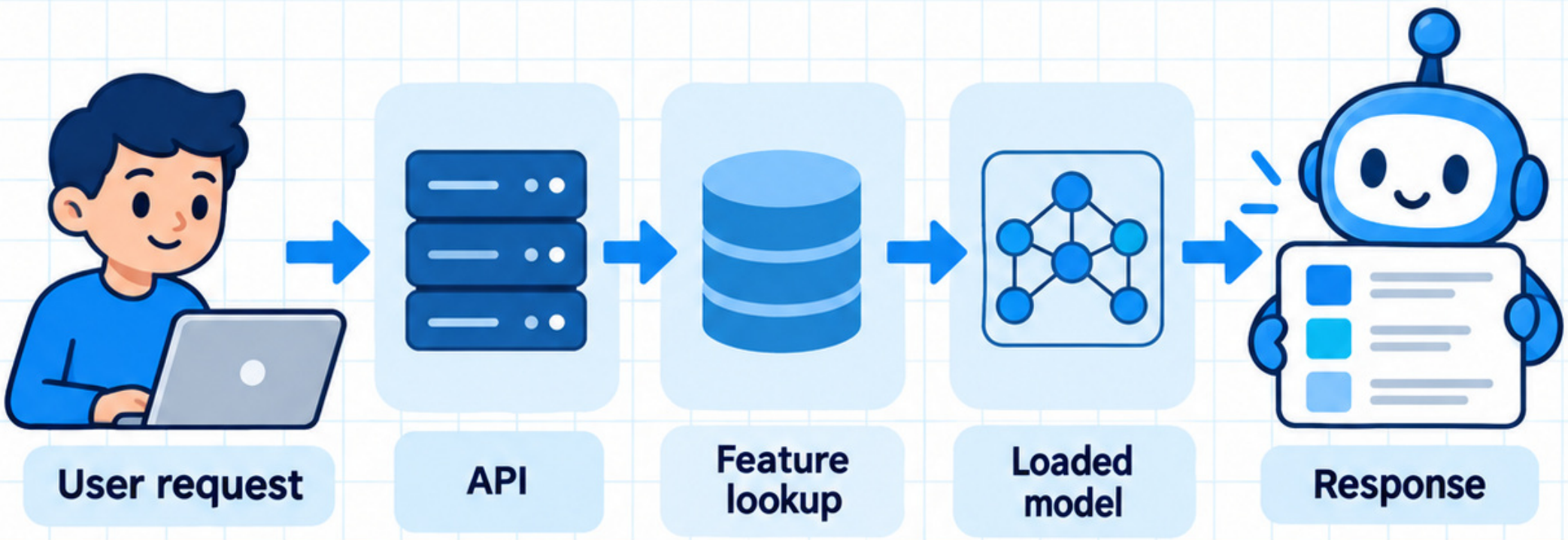
THE USER IS WAITING FOR THE PREDICTION

Online inference

A live prediction service must return a useful result within a deadline.

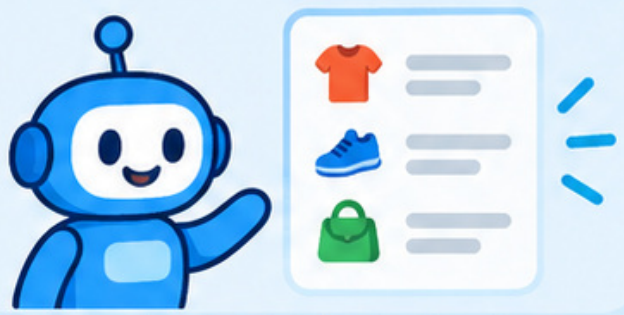
Today: design the request path, feature checks, fallback behavior and measurements for a small recommendation service.



ONLINE MEANS REQUEST-TIME PREDICTION

A request triggers prediction using the accepted release and appropriate request-time features.

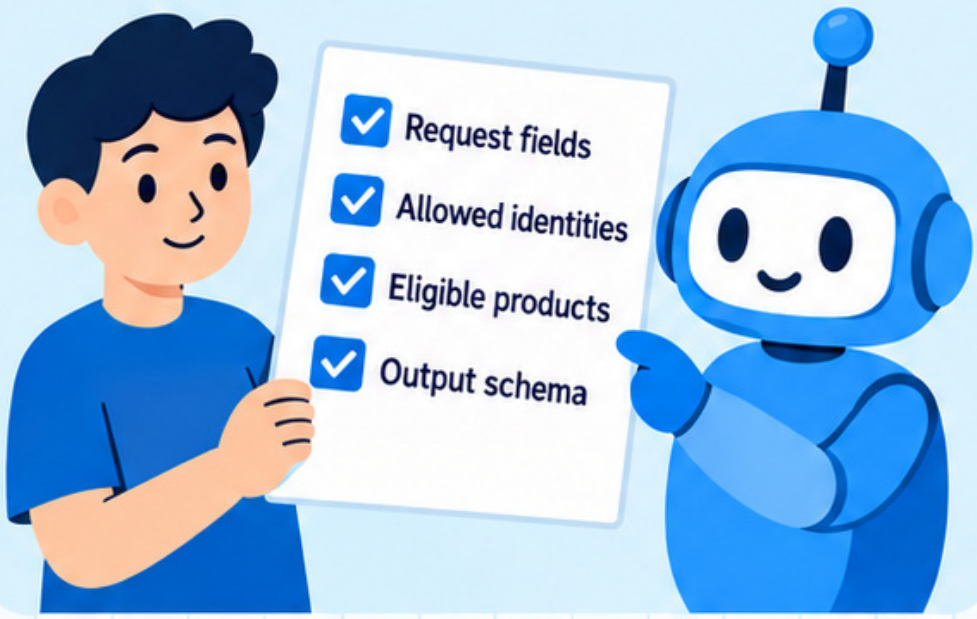
Our fictional store wants to rank a small eligible product set for the current visit.



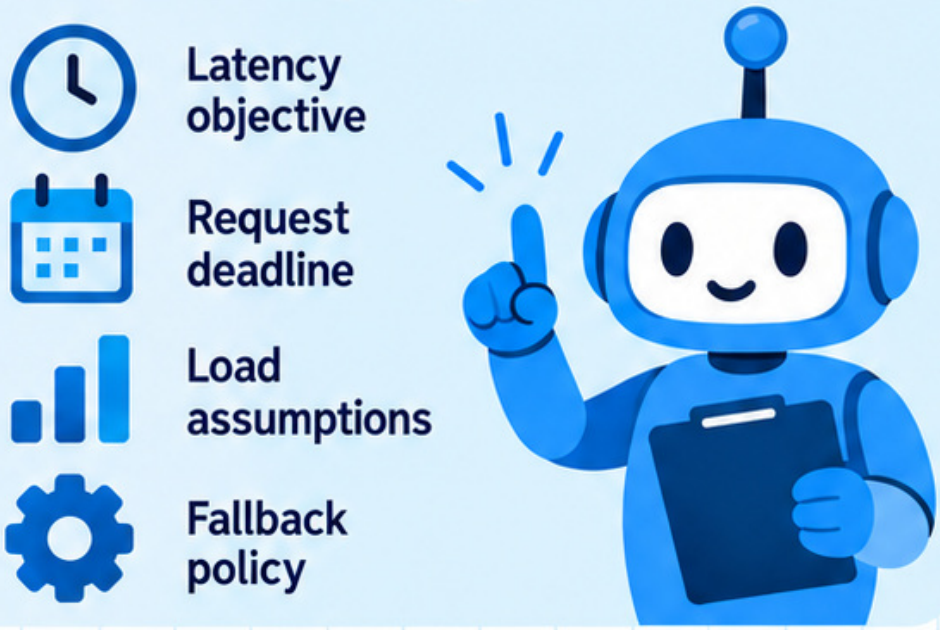
This path serves predictions. Training and release acceptance happen separately.

DEFINE THE RESPONSE BEFORE CHOOSING A SERVER

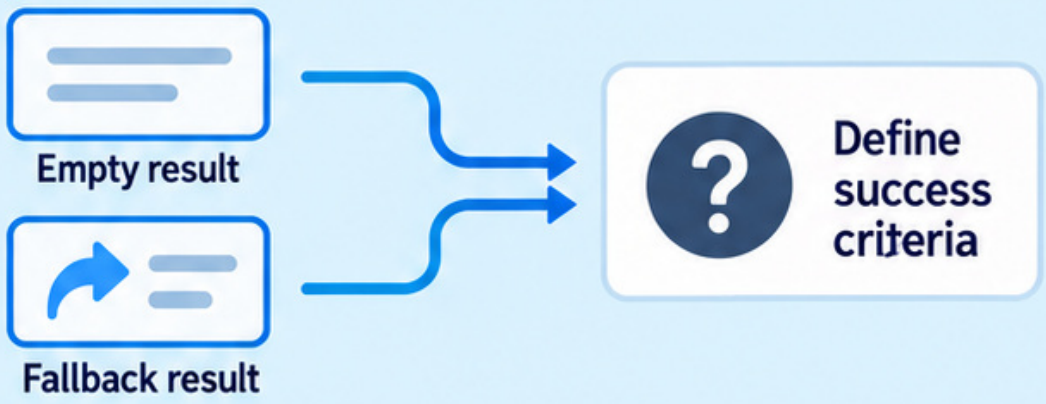
Specify request fields, allowed identities, eligible products and output schema.



Set the latency objective, request deadline, load assumptions and fallback policy.

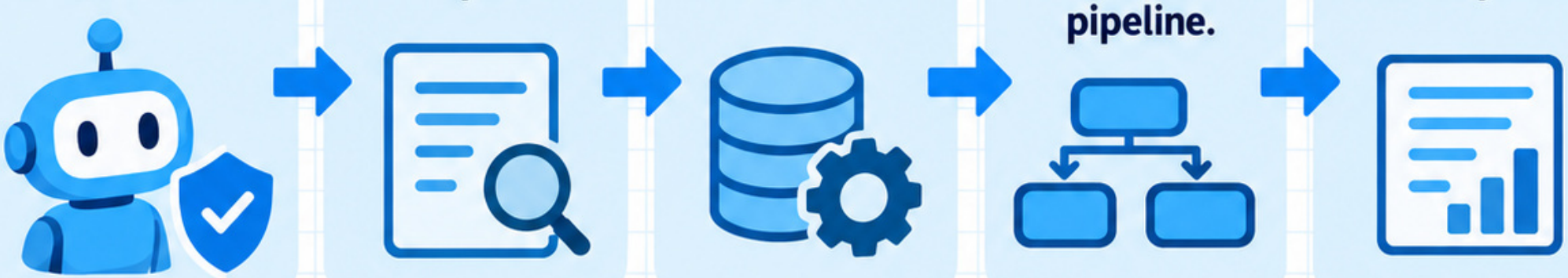


State whether an empty or fallback result counts as a successful product experience.



MAKE EVERY REQUEST STAGE VISIBLE

1. Authenticate and authorize.
2. Validate input.
3. Resolve features.
4. Apply the loaded pipeline.
5. Filter and shape the response.



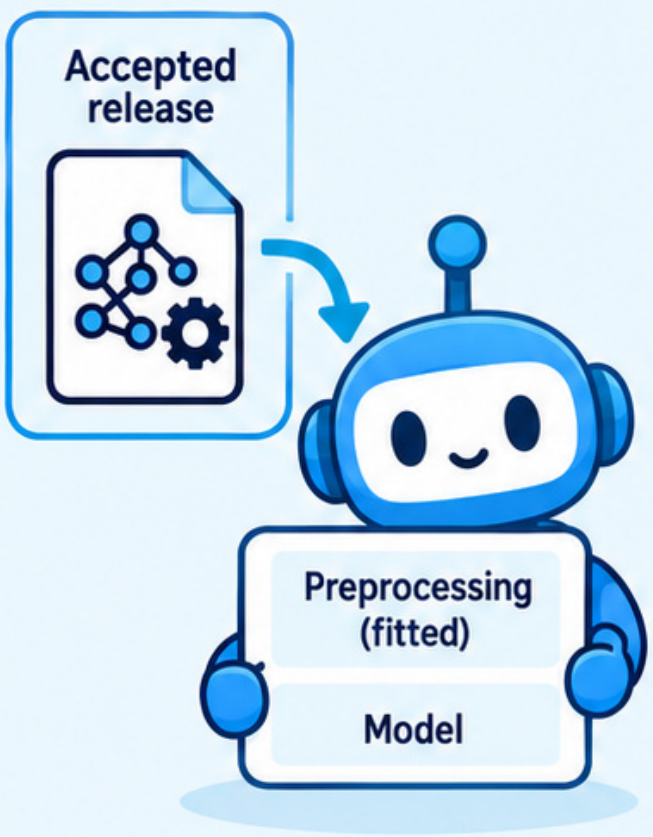
Carry a request ID across stages.

request ID

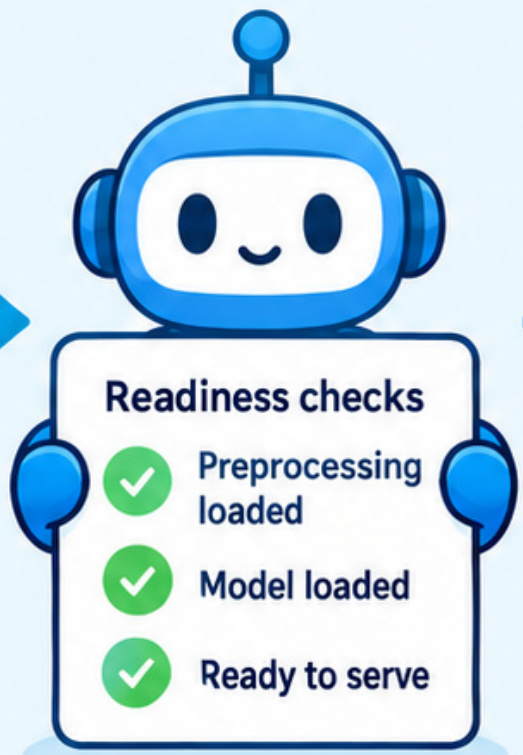
Do not let the client choose an unauthorized customer identity or model artifact.

LOAD THE ACCEPTED PIPELINE BEFORE SERVING

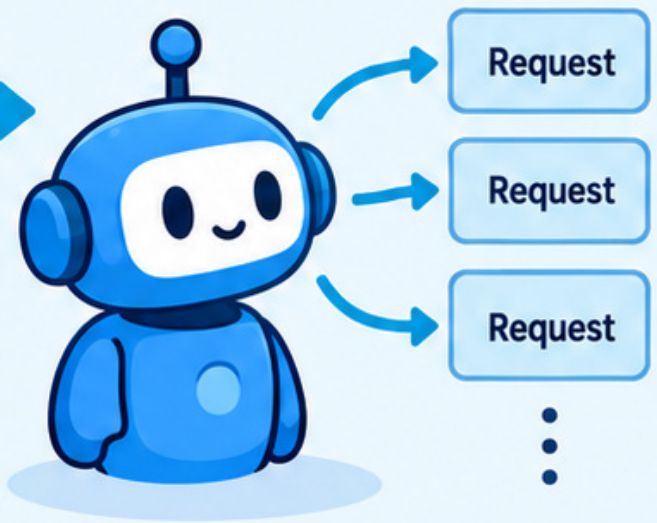
Load and check the model with its fitted preprocessing when a worker starts.



Run readiness checks before sending traffic to that worker.



Do not download and deserialize the model anew for every request. A release change needs a controlled loading and activation path.



FAST RETRIEVAL STILL NEEDS VALID FEATURES

Check entity key, feature definition, units, missing values and acceptable age.

Feature lookup

2 ms
(illustrative)

- entity key
- feature definition
- units
- missing values
- acceptable age

A feature returned in 2 ms can still be too old or belong to the wrong customer.

Lookup duration:
2 ms

Feature age:
check timestamp

Treat missing and zero as different values unless the feature contract explicitly equates them.

missing value	zero value
(missing)	0

ALLOCATE TIME ACROSS THE WHOLE PATH

Illustrative 200 ms server-side budget:

Queue:	20 ms
Identity and input checks:	20 ms
Feature retrieval:	60 ms
Model scoring:	70 ms
Response shaping:	10 ms
Reserve:	20 ms

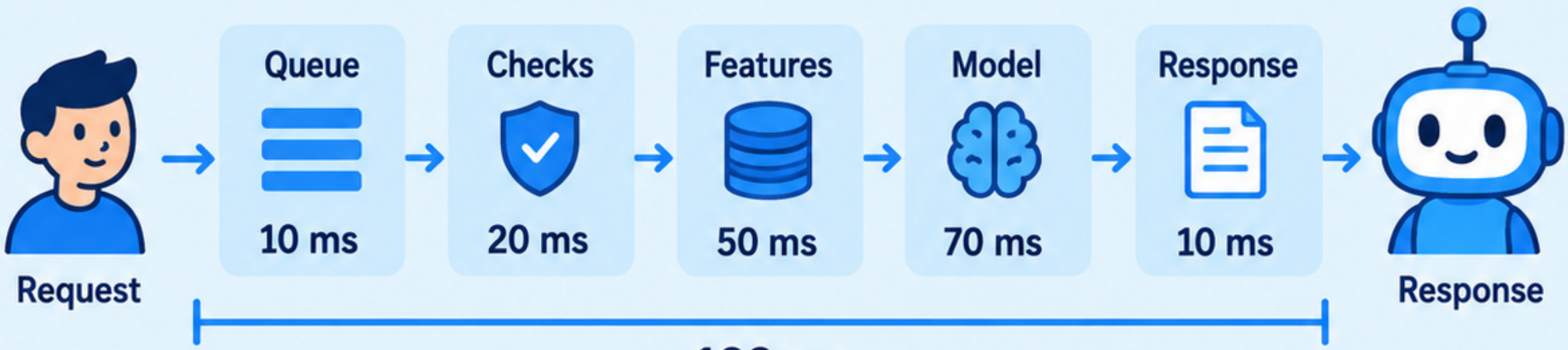
200 ms

These are planning allocations, not measured percentiles.

MEASURE ONE COMPLETE REQUEST TRACE

Toy sequential trace:

Queue 10 + checks 20 + features 50 + model 70 + response 10 = 160 ms.



These are a single toy trace, not percentile measurements.

Measure the full request boundary, including queue time.

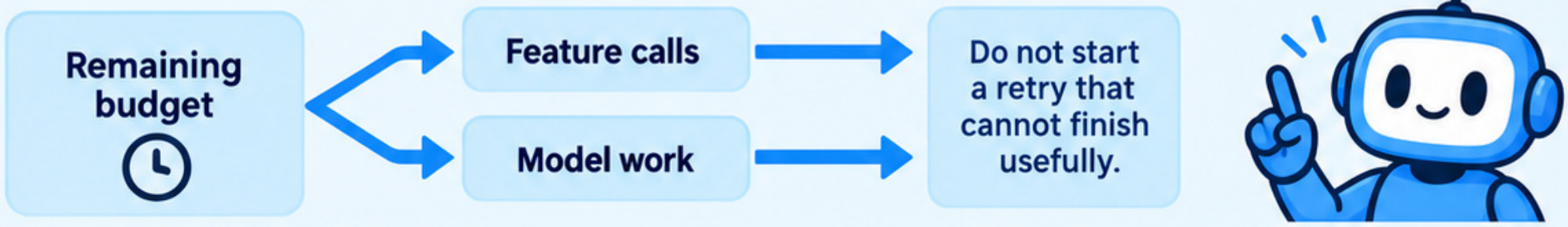
Client-visible latency can also include network and frontend work.

Adding component p95 values does not calculate end-to-end p95.

PROPAGATE THE REMAINING DEADLINE

If earlier stages consume time, later stages have less time left.

Bound feature calls and model work by the remaining budget. Avoid starting a retry that cannot finish usefully.

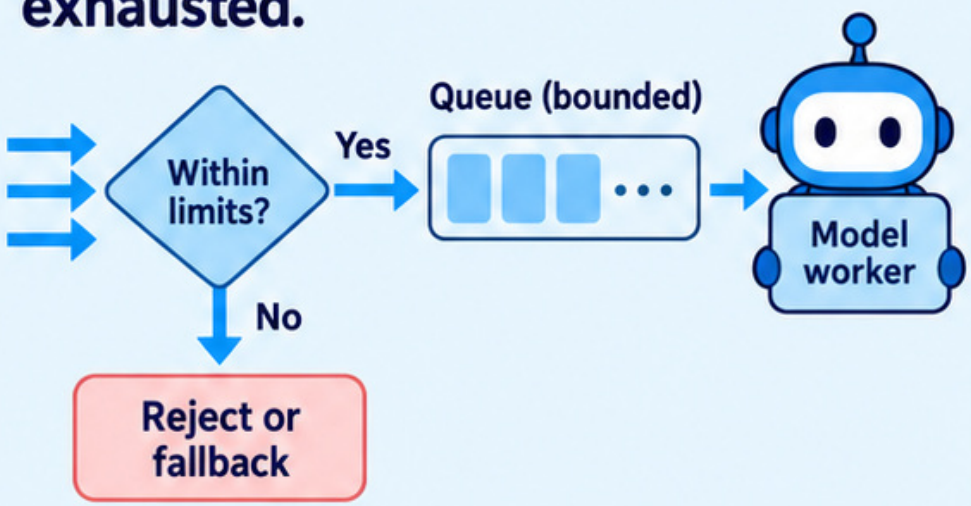


A client timeout does not automatically prove that server or accelerator work has stopped.

BOUND CONCURRENCY AND QUEUE GROWTH

More incoming requests can create queueing before model work starts.

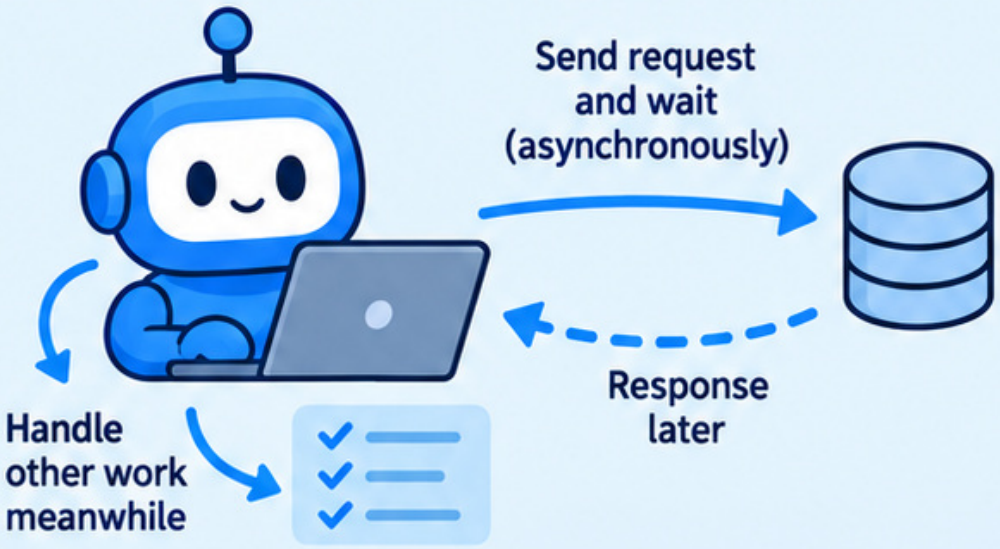
Limit admitted work and queue size. Define rejection or fallback behavior when capacity is exhausted.



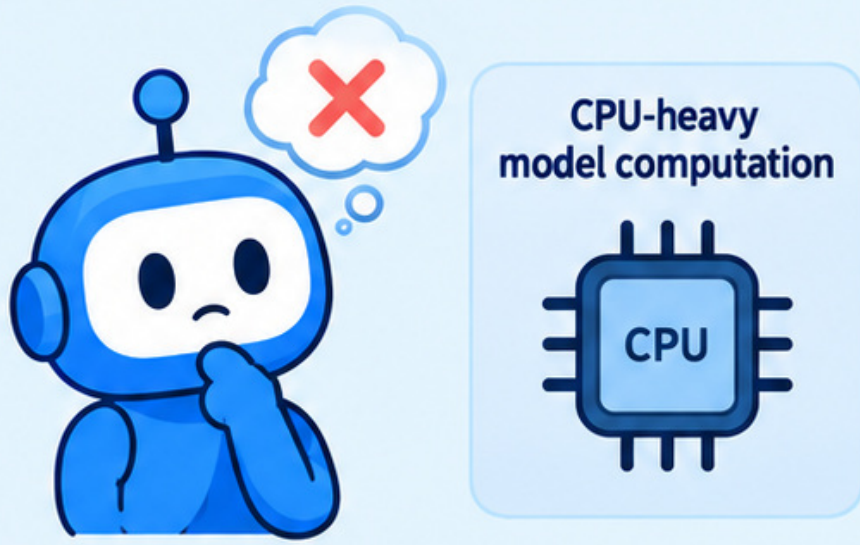
A longer queue can increase latency even if each model call takes the same time.

WAITING AND COMPUTING NEED DIFFERENT CAPACITY PLANS

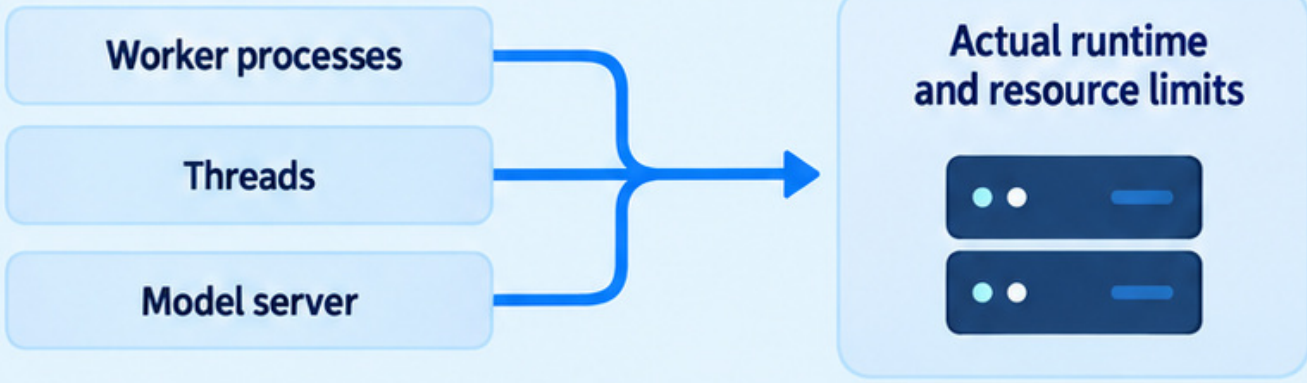
Async I/O can let a server handle other work while a supported database or network call waits.



It does not make CPU-heavy model computation parallel by itself.



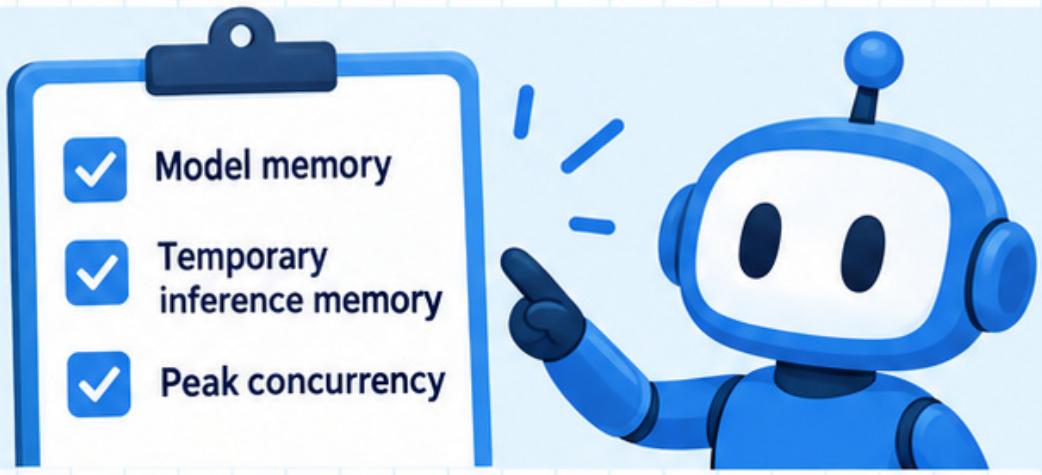
Plan worker processes, threads or a model server around the actual runtime and resource limits.



EACH WORKER CAN ADD MEMORY COST

Multiple server processes often load separate copies of the model.

Measure model memory, temporary inference memory and peak concurrency before increasing worker count.



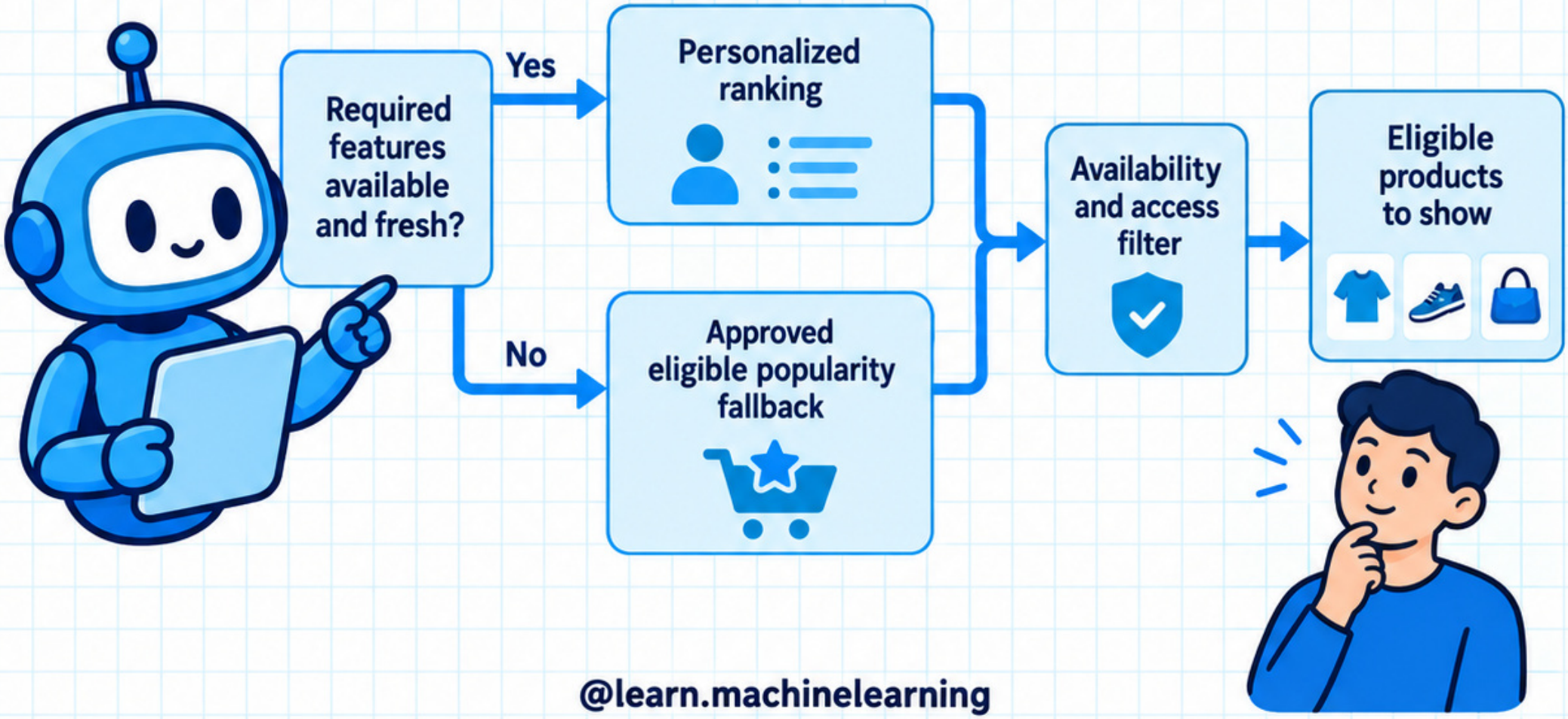
Worker count is a capacity decision, not a free speed setting.

DESIGN A USEFUL DEGRADED RESPONSE

If required features are missing, stale or late, use only an approved fallback.

For this store, a fallback could return currently eligible popular products. It still obeys availability and access rules.

Record the fallback reason. Do not describe it as a fresh personalized prediction.

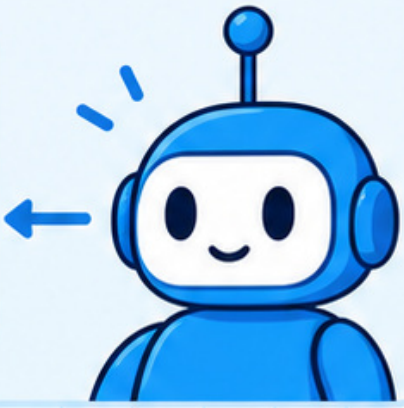


RETURN ENOUGH METADATA TO EXPLAIN BEHAVIOR

The application response can include result IDs and an outcome such as personalized, fallback or unavailable.

Application response

result IDs:	[101, 202, 303]
outcome:	personalized

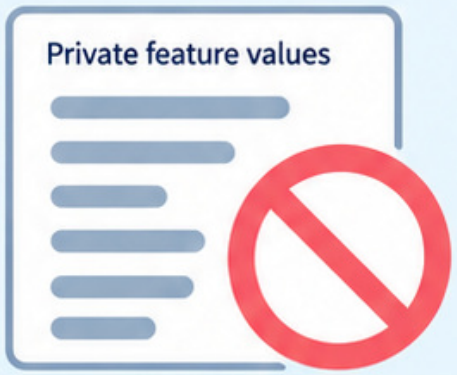


Internal traces record release version, feature timestamps, stage timings and fallback reason.

Internal trace metadata

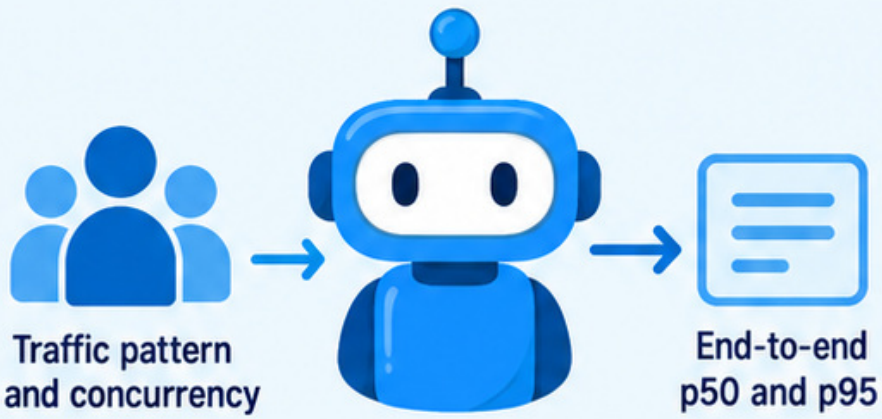
release version:	1.4.0
feature timestamps:	included
stage timings:	included
fallback reason:	included

Avoid returning private feature values just for debugging.



EVALUATE LATENCY TOGETHER WITH OUTCOME QUALITY

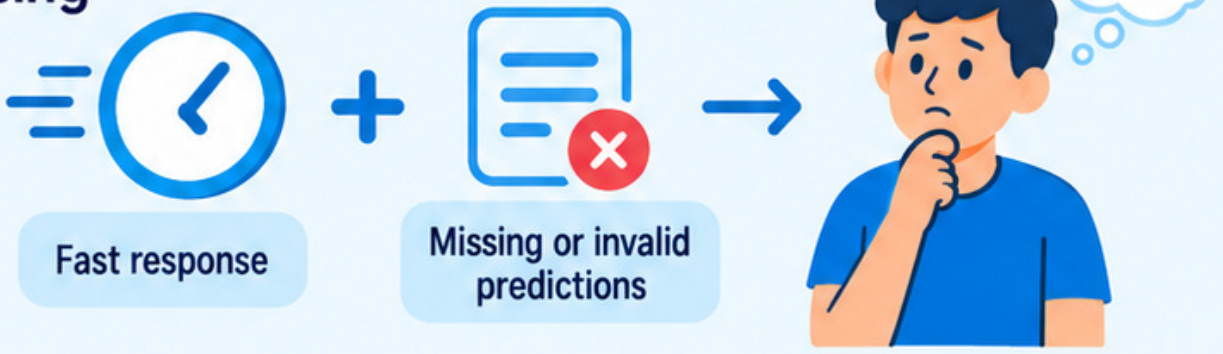
Measure end-to-end p50 and p95 under a stated traffic pattern and concurrency.



Report errors, timeouts and fallback rate alongside latency. Separate cold-start and steady-state observations.

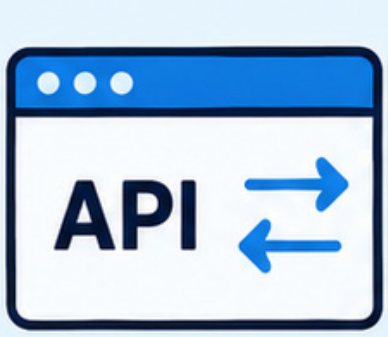
Load (traffic pattern and concurrency)	p50	p95	Error rate	Timeout rate	Fallback rate	Quality (outcome)	Cold-start / Steady-state
Cold-start							
Steady-state							

A fast response is not useful if required predictions are missing or invalid.

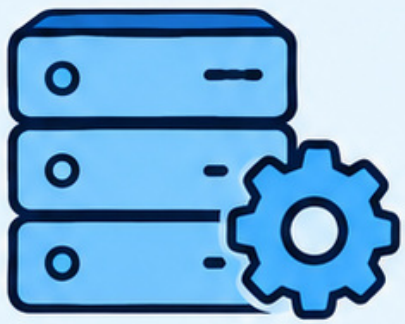


USE TOOLS AT THE RIGHT BOUNDARY

FastAPI: request handling and API integration.



A fitted pipeline or model server: inference.



A feature source: approved feature values and timestamps.

OpenTelemetry: traces, metrics and logs for observation.

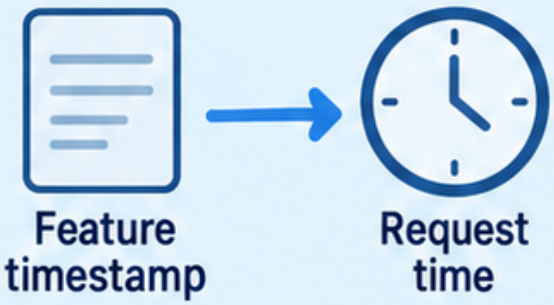
Application code still defines deadlines, eligibility and fallback rules.

@learn.machinelearning

YOUR TURN: HANDLE A STALE FEATURE

Toy service policy: maximum feature age is 15 minutes.
Request time: 09:00. Feature timestamp: 08:40.
Lookup duration: 2 ms.
Approved fallback: currently eligible popular products.

1. Is the feature usable?

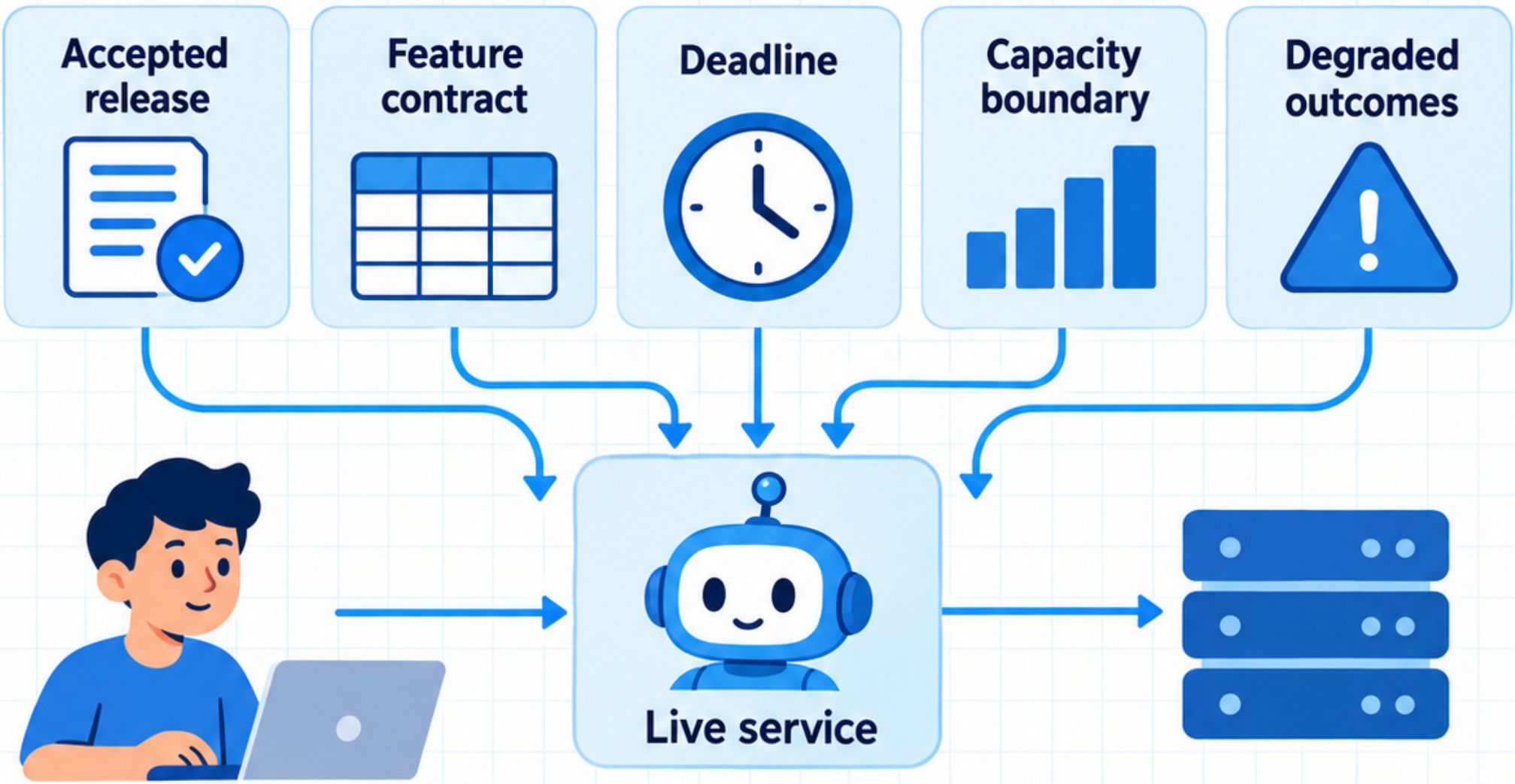


2. What outcome should the response show?

3. What should the trace record?

DEFINE WHAT THE SERVICE DOES UNDER PRESSURE

A serving design includes the accepted release, feature contract, deadline, capacity boundary and degraded outcomes.



DAY 15: Do you need a feature store?

Decide when shared feature infrastructure earns its cost.