PRODUCTION AI CHEAT SHEET

Production Reliability & Observability

Debug from the request forward. Find the first divergence, not just the final bad answer.

Core idea: A production AI failure can start anywhere in the chain. Make each boundary observable so you can localize the defect instead of guessing from the final response.

The production path

Browser user action → **Frontend** request + UI state → **API** validation + orchestration → **Model** probabilistic interpretation → **Database** durable records

Where failures hide

Boundary	Typical failure	First evidence to inspect
Browser / frontend	Stale local state, rerender race, wrong API base, double submit.	Network request, console, rendered state, reproduction video.
API / app logic	Wrong status, unsafe retry, bad validation, duplicate side effect.	Request ID, route logs, response body, persisted records.
Model provider	Timeout, refusal, malformed/poor interpretation, latency spike.	Provider timing, model ID, token counts, structured result.
Database	Constraint failure, stale version, rollback, migration drift.	Transaction logs, row versions, migration state, affected records.
Cross-boundary	Works locally, fails deployed; config/environment mismatch.	Environment, deployed commit, service logs, DB target, CORS.

Trace, log, metric - different jobs

Trace
What happened for this one request, in what order?

Log
What technical event, warning, or error happened?

Metric
How often, how slow, how expensive, or how widespread is it?

Correlation/request ID ties the browser request, API logs, provider call, and database activity together.

PRODUCTION AI CHEAT SHEET

Production Reliability & Observability

Safe recovery matters because AI calls are slow, fallible, and can fail after partial work already happened.

Safe failure and retry design

Situation	Bad recovery	Better recovery
Evidence saved; model call fails	Ask user to submit the Note again.	Keep the same Evidence; offer retry analysis.
Review races with newer State	Overwrite newer truth.	Reject stale proposal with version checks.
Valid shape, bad references	Persist partial downstream records.	Reject deterministically; preserve failed interpretation.
Frontend waits for server update	Sleep 300 ms and hope.	Use the authoritative response and deterministic revalidation.
Provider unavailable	Expose internal exception or mutate anyway.	Return safe service failure; preserve authoritative data.

Latency: measure the parts

Frontend / network + **API work** + **Provider call** + **Validation / persistence** + **Rehydration**

> If inference takes 15 seconds and persistence takes 20 ms, optimizing SQL will not meaningfully improve the user experience.

Production checklist

- ✓ **Identity:** deployed version/commit and model identifier.
- ✓ **Configuration:** API base, provider keys, CORS, database URL, environment.
- ✓ **Durability:** which records must survive restart/redeploy.
- ✓ **Retries:** idempotency and version checks where needed.
- ✓ **Logging:** safe request IDs, timing, provider/model, usage, categorized errors.
- ✓ **Regression:** production defects become tests/evals when feasible.
- ✓ **Graceful degradation:** preserve authoritative data when AI is unavailable.

Reliable AI products make every boundary observable, every consequential write controlled, and every retry safe.