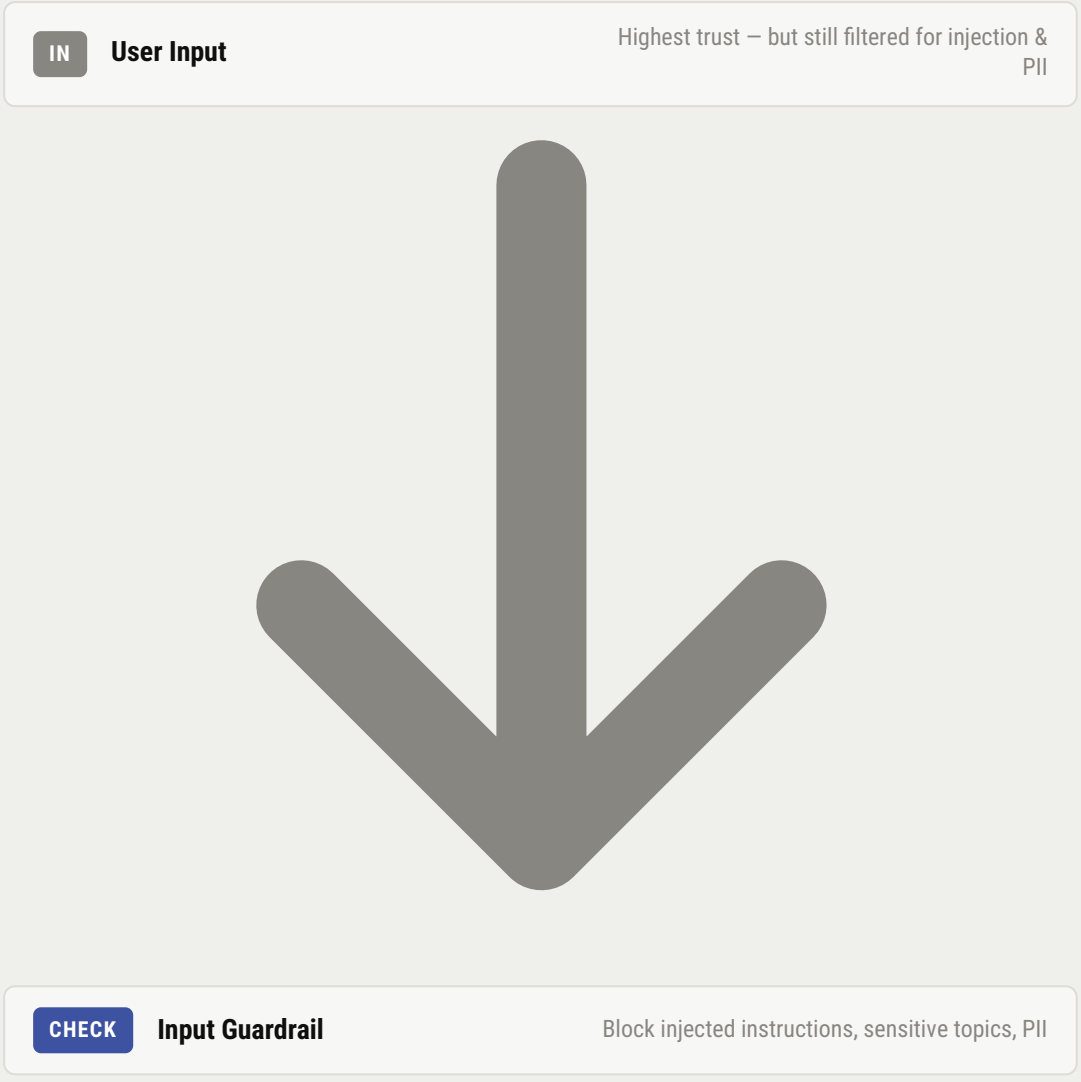
Grigoriy Dobryakov AI-DRIVEN HEAD OF ENGINEERING

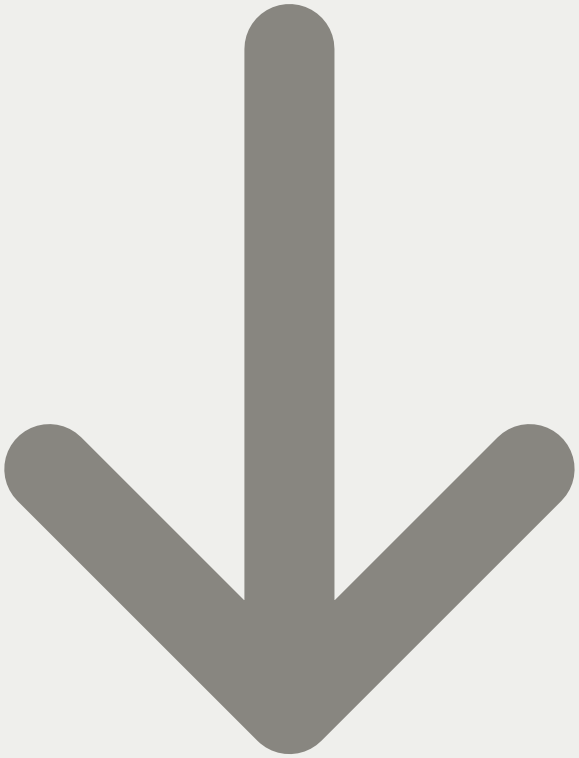
Production Readiness Checklist for AI Agents

Before your agent touches email, CRM, SQL, or production, run this checklist. It covers **guardrails** (boundary checks that stop a wrong answer from becoming real damage) and **harness** (the engineering wrapper that keeps the system reliable in production). If you can't tick a box, that's where your blast radius lives.

The Request Pipeline — Where Trust Drops

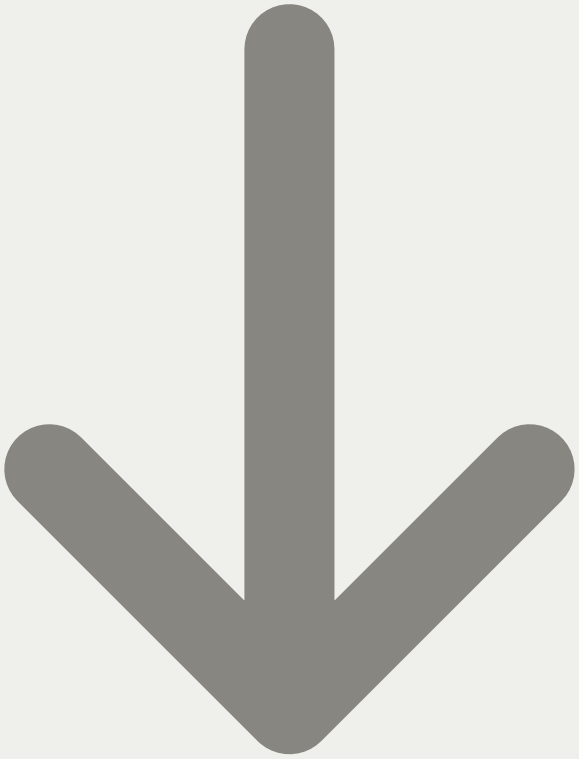
EVERY BOUNDARY GETS ITS OWN CHECK. TRUST FALLS LEFT → RIGHT.





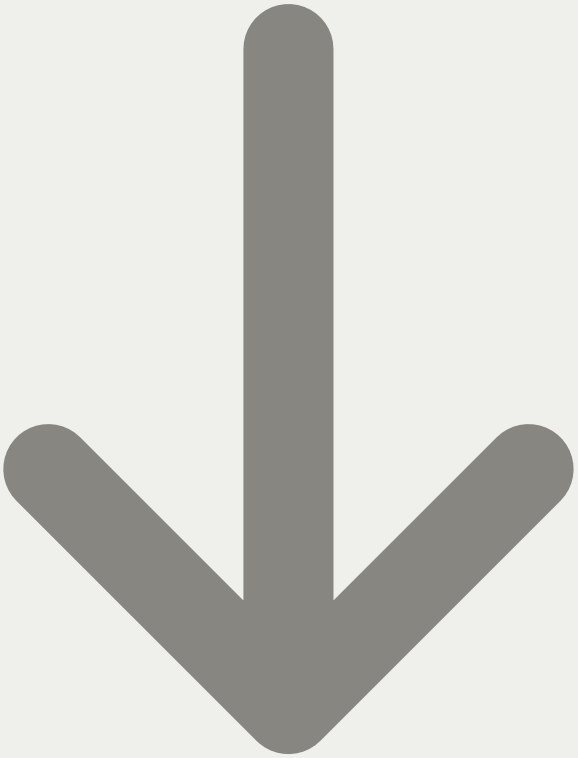
CORE **Orchestrator → Model**

RAG context filtered: secrets, keys, stale data removed

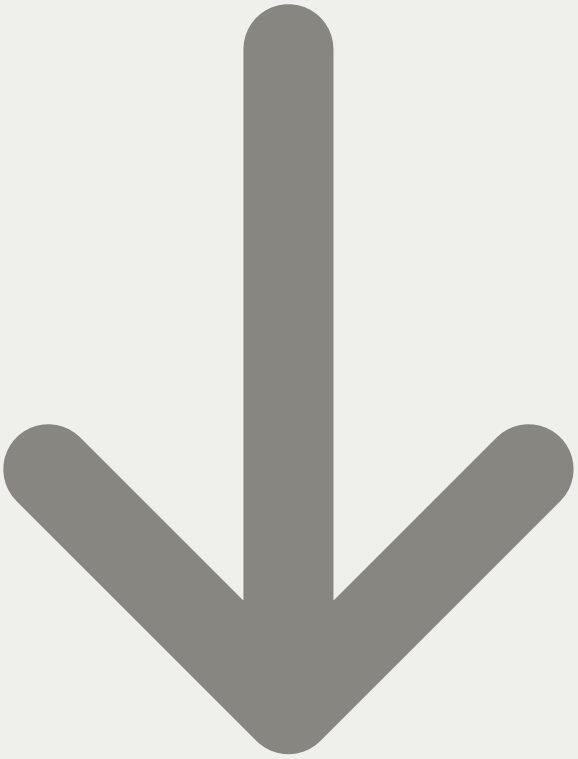


CHECK **Tool Guardrail (before)**

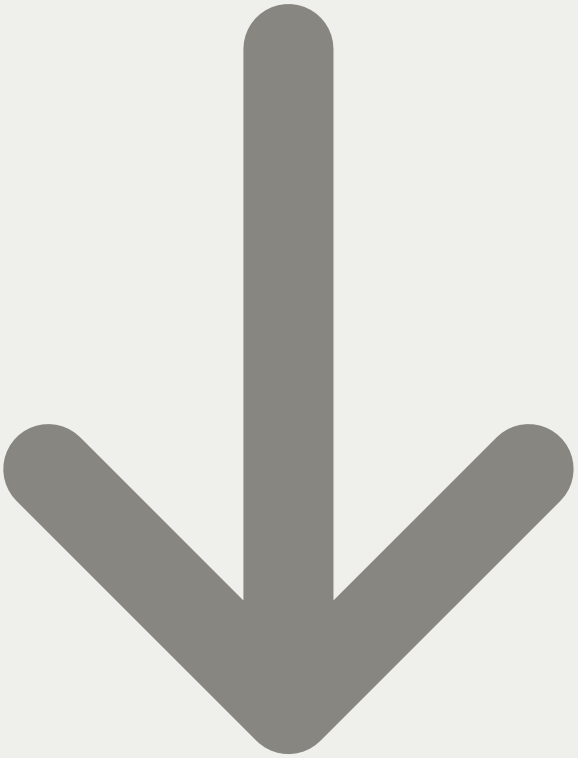
Verify params, permissions, policy compliance



CORE **Tool Execution** External service runs — lowest trust zone

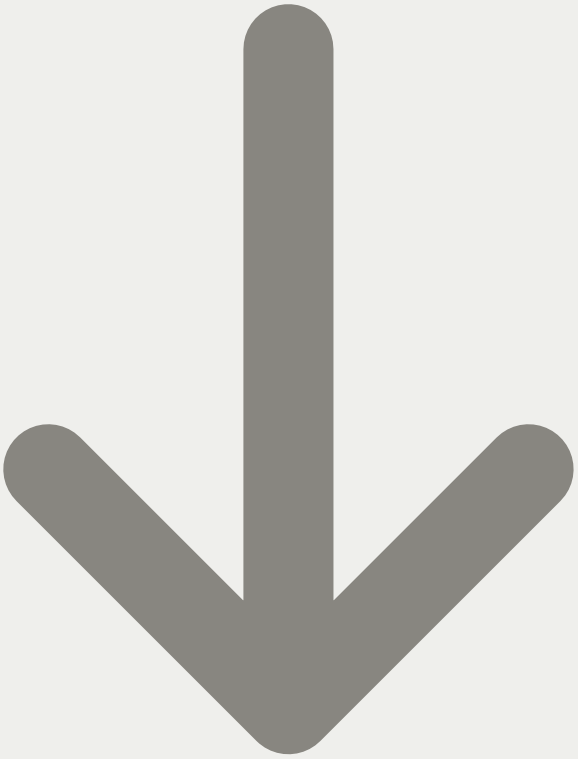


CHECK **Tool Guardrail (after)** Scan result for secrets, PII, volume; block or sanitize



CHECK **Output Guardrail**

Validate JSON schema; catch leaks & errors before release



OUT **To User**

Answer released — blast radius closed

The 8-Point Production Checklist

- ☐ **Trust boundaries drawn explicitly**

  Every boundary — input, context, tool results, output — has its own check. Guardrails live on the arrows, not "around the model" abstractly.

- ☐ **All external content filtered as untrusted**

  Email, PDF, search results, GitHub READMEs, SQL output — all treated as untrusted input. Classified and filtered before entering the model's context. Instructions and external data are architecturally separated (system prompt vs. `tool_result`).

- ☐ **Policy layer locked by access rights**

  Dangerous action classes — moving money, deleting data, touching production — are blocked by RBAC, not by prompt text. The most careful output filter won't help if the agent was granted `DROP TABLE` rights.

- ☐ **Irreversible actions require human approval**

  Money movement, data deletion, external publication, infrastructure changes. Not every step — only where the action is irreversible or expensive. Approval authorizes one operation now; it does not grant new access rights.

- ☐ **Prompts versioned; regression caught by evals**

  Prompts are versioned, A/B-tested, and rolled back if needed. Before each release, the model runs through eval scenarios — comparing quality, hallucinations, cost, and speed. Old sessions are replayed on the new version to catch regressions before users do.

- ☐ **Tracing, session replay, and cost monitoring in place**

  Full traces: requests, responses, tool calls, tokens, cost, errors, user ratings. Token cost is monitored proactively — it grows quietly and shows up in the invoice. Without this data, you can't diagnose degradation.

State lives in the system, not "in the model's head"

Dialog history, tool results, artifacts, memory, approval statuses, retry counters — all stored and controlled externally. The orchestrator manages the lifecycle; the model only proposes the next step.

Model failure has a plan

Retry, route to a different model, degrade gracefully instead of crashing. Simple tasks go to cheap models; expensive models engage only for complex ones. New versions roll out behind feature flags with fast rollback.

Guardrail Decisions: Code vs. Model vs. Human

WHAT YOU NEED TO CHECK	RIGHT MECHANISM	WHY NOT THE LLM?
JSON structure validity	**JSON Schema**	Deterministic — code is faster and certain
Secrets, API keys, card numbers	**Regex patterns**	A few lines of code beat expensive tokens
Access rights & permissions	**RBAC**	Rights belong to the system, not the prompt
Toxic content moderation	**Moderation model**	Specialized classifier, not the generation model
Jailbreak & injection detection	**Dedicated classifier model**	Separate model trained on attack patterns
Business rules (legal, financial advice)	**Business-rule layer**	Product-specific policy, not model judgment
Irreversible / expensive actions	**Human approval**	Only where action can't be undone — not every step
Natural-language reasoning tasks	**LLM**	Only where code genuinely can't do the job

Quick Win — 15 Minutes Now

Pick one agent or AI feature you're about to ship (or already run). Walk the pipeline above and answer:

1. Draw the trust boundaries for *your* system. Which boundaries have no check today?
2. List every tool the agent can call. For each: what permissions does it have? Could a successful injection escalate beyond that tool?
3. Find the one irreversible action the agent can trigger. Is it gated by human approval, or only by prompt instructions?

Next step: turn the gaps you just found into entries in your guardrail incident log. Guardrails aren't designed once upfront — they grow from real incidents and near-misses. Start the log today; every unchecked box above is your first entry.

Grigoriy Dobryakov

AI-DRIVEN HEAD OF ENGINEERING

dobryakov.net

grigoriydobryakov@gmail.com

t.me/grigoriydobryakov

linkedin.com/in/dobryakov

youtube.com/@IT-Head