DIRAYA // PRODUCTION AI ENGINEERING

THE AGENT GHOST-CASES LIST

40 production-killers most teams miss when shipping their first AI agent.

> A note from Mohammed, who runs engineering at Diraya. We have shipped 23 production AI builds for seed to Series B teams. The same failures show up on almost every project, and nearly none of them appear in a demo. This is the list we check every build against before it reaches a customer. It is yours, with no call attached. Run it against the agent you are building this week.

A Tool and function calling

1. **Hallucinated arguments.** The model fills a required parameter with a plausible value that never appeared in the conversation or the data, so a valid-looking call runs against the wrong entity. **Catch it:** validate every argument against a source of truth before the call, and eval that ambiguous inputs make the agent ask, not guess.
2. **Error-blind continuation.** A tool returns an error body with a 200 status, the agent reads the shape and not the meaning, and it proceeds as if the call succeeded. **Catch it:** return typed failures from tools and add evals that inject errors and assert the agent stops or recovers.
3. **Infinite tool loop.** The result never satisfies the goal, so the agent calls the same tool with the same arguments until it hits the step cap or the bill. **Catch it:** cap repeats per tool, detect identical consecutive calls, and eval loop termination on unsolvable inputs.
4. **Confident wrong tool.** Two tools have similar descriptions, like refund versus credit or search versus lookup, and the model picks the plausible one. **Catch it:** write disambiguating tool descriptions and score tool-selection accuracy on near-miss prompts.
5. **Stale tool schema.** The downstream API renamed or added a required field, the agent still calls the old signature, and every call fails in production. **Catch it:** contract-test tool schemas in CI and alert on tool-error-rate spikes.
6. **Parallel tool race.** The agent issues two calls that mutate the same record, and the final state depends on which arrives first. **Catch it:** serialize mutations on a key, make writes idempotent, and eval concurrent-action ordering.

B Retrieval and RAG

7. **Empty-retrieval hallucination.** The retriever returns nothing relevant and, instead of saying so, the model answers from parametric memory with full confidence. **Catch it:** gate generation on a retrieval-score threshold and eval the no-good-context path explicitly.
8. **Chunk-boundary amputation.** The one sentence that answers the question is split across two chunks, so neither is retrieved whole and the answer is half right. **Catch it:** use overlapping chunks and test retrieval recall on boundary cases.
9. **Citation fabrication.** The agent cites a document id or URL that was not in the retrieved set, and it reads as authoritative. **Catch it:** constrain citations to retrieved ids and fail any citation outside the context.
10. **Confident stale answer.** The index lags the source system by hours or days, and the agent presents outdated facts as current. **Catch it:** stamp freshness on chunks, surface it in the answer, and alert on index lag.
11. **Embedding drift.** You upgrade the embedding model but do not re-embed the corpus, so query and document vectors live in different spaces and recall collapses. **Catch it:** version embeddings, block mixed-model search, and re-run a retrieval eval after any model change.
12. **Near-duplicate flooding.** Top-k fills with five near-identical boilerplate chunks, pushing the real answer out of the window. **Catch it:** dedupe by content hash before ranking and eval recall on duplicate-heavy corpora.
13. **Cross-tenant leakage.** A shared index returns another customer's chunk because a tenant filter was missing on one code path. **Catch it:** enforce tenant scoping at the retrieval layer and eval for zero cross-tenant hits.

C Context and memory

14. **Silent context overflow.** History exceeds the window, the oldest tokens drop with no error, and those tokens are often the system constraints. **Catch it:** budget tokens explicitly, pin system constraints, and alert when truncation occurs.
15. **Summary amnesia.** Conversation summarization compresses away the one constraint that mattered, such as never offer a discount. **Catch it:** extract and pin durable constraints outside the summary and eval constraint retention across long sessions.
16. **Memory poisoning.** A wrong fact gets written to long-term memory once and then resurfaces in every future session as ground truth. **Catch it:** validate writes, version memory, and probe for persisted false facts.

17 **Ephemeral PII retention.** Memory stores user PII that should have been transient, and it surfaces later or survives a deletion request. **Catch it:** classify and TTL sensitive fields and eval deletion and retention guarantees.

18 **Lost in the middle.** The critical instruction sits mid-context and the model under-weights it relative to the start and the end. **Catch it:** position key constraints at the edges and eval instruction following as context grows.

19 **Cross-session bleed.** A shared cache key collides and one user's state leaks into another's conversation. **Catch it:** namespace all state by session and tenant and load-test for collisions.

D Prompt and injection

20 **Indirect prompt injection.** A retrieved document, email, or web page contains instructions like ignore previous rules, and the agent obeys the data instead of the operator. **Catch it:** treat all retrieved content as untrusted, fence it, and red-team with injection eval sets.

21 **Tool-result injection.** A tool returns attacker-controlled text that redirects the agent's next action. **Catch it:** sanitize and delimit tool outputs and eval against poisoned tool returns.

22 **System-prompt exfiltration.** A user coaxes the agent into revealing its hidden rules or keys, exposing your logic and your guardrails. **Catch it:** keep secrets out of the prompt entirely and eval for prompt-leak resistance.

23 **Instruction collision.** The system prompt says be concise, a retrieved policy says be exhaustive, and the agent flips between them unpredictably. **Catch it:** define a strict precedence order and eval conflicting-instruction scenarios.

24 **Delimiter break.** User input contains the same delimiter you use to fence context, and your prompt structure collapses. **Catch it:** use unguessable delimiters or structured message roles and fuzz inputs that mimic your fences.

E Output and parsing

25 **Markdown-wrapped JSON.** The model wraps its JSON in a code fence and your strict parser throws on the backticks. **Catch it:** parse defensively, request structured output where supported, and eval parse-success rate.

26 **Schema drift on upgrade.** A new model snapshot adds, renames, or reorders fields, and your downstream code breaks silently. **Catch it:** validate against a strict schema, pin model versions, and eval output conformance on every change.

27 **Hallucinated enum.** The model returns a status or category outside your allowed set and a switch statement falls through. **Catch it:** validate enums, reject out-of-set values, and eval enum adherence.

28 **Partial-stream commit.** You act on streamed tokens before completion, and the model's final output contradicts what you already did. **Catch it:** commit only on stream completion for any side-effecting action.

29 **Locale corruption.** Numbers and dates are parsed under the wrong locale, so 1.000 reads as one or as one thousand depending on the reader. **Catch it:** force explicit formats and units and eval across locales.

F State, idempotency, and recovery

30 **Double side-effect.** A non-idempotent action, like a charge, an email, or a ticket, is retried after a timeout and fires twice. **Catch it:** attach idempotency keys to every external action and eval retry behavior.

31 **No rollback.** A five-step action fails at step three and leaves the world half changed with no compensation. **Catch it:** design compensating actions or sagas and eval partial-failure recovery.

32 **Orphaned async job.** The agent starts a background task, then crashes or moves on, and the result is never reconciled. **Catch it:** persist job handles and reconcile on restart.

33 **Timeout mid-tool.** The tool succeeds server-side but the agent times out, assumes failure, and retries or reports the wrong outcome. **Catch it:** make actions queryable by id and confirm state before retrying.

34 **Lost human handoff.** An escalation to a human silently drops because no queue or owner exists, and the user waits forever. **Catch it:** make handoffs first class with acknowledgement and eval the escalation path.

G Evals, drift, and observability

35 **No eval suite.** Without a written eval set you cannot tell a regression from noise, so every deploy is a gamble. **Catch it:** build the eval suite before the agent. That inversion is the entire Diraya method.

36 **Eval-set leakage.** Your eval cases leaked into the prompt or the few-shot examples, so scores look strong and production does not. **Catch it:** keep eval data strictly out of prompts and rotate held-out sets.

37 **Silent model regression.** The provider ships a new default snapshot, behavior shifts, and nothing alarms. **Catch it:** pin versions and run the eval suite on every provider change before promoting.

38 **Untraceable failure.** A user reports a bad answer and you have no per-step trace to reproduce it. **Catch it:** log every step, tool call, and retrieved chunk with a trace id from day one.

H Cost, latency, and scale

39 **Runaway step budget.** Hard queries drive the agent to its max-step cap every time, multiplying latency and cost. **Catch it:** set tight step budgets, detect non-convergence early, and eval cost per query.

40 **N plus one LLM calls.** A per-item loop makes one model call each, so 200 items become 200 calls and a latency wall. **Catch it:** batch where possible and eval throughput under realistic load.

THE PATTERN BEHIND ALL 40

Every ghost on this list shares one trait. It is invisible in a demo and only appears in production, usually under load, usually in front of a customer.

The teams that ship reliable agents are not smarter. They **write the eval first, then the agent, then they harden the agent against the evals** until the failure rate is low enough to ship. That single inversion, evals before agent, is what separates a demo from a product.

Want a second set of eyes on the agent you are building? Reply **REVIEW** and Mohammed sends a free architecture review for your specific stack: the design he would pick, the three biggest risks, and a realistic timeline, inside 48 hours. No call required. Or book 30 minutes directly at **calendly.com/amoura-ma-diraya/30min**.