

FREE EMAIL COURSE

The Agentic AI Playbook

Seven lessons on taking an AI agent from prototype to production without it ever making a decision you cannot undo.

Nimit Solanki · nimitsolanki.com · 2026

Which decisions an agent should be allowed to make

Start with the decision, not the model. Most agent projects fail because nobody wrote down what the agent is allowed to do.

By Nimit Solanki · 5 min read

Every agent project I have seen go wrong went wrong at the same step: someone connected a capable model to a real system without first deciding which decisions that model was allowed to make on its own. The model was fine. The permission was missing.

The decision inventory

Before you write a line of orchestration code, list every decision the agent will touch. For each one, answer four questions:

1. **What does a wrong answer cost?** A wrong product recommendation costs a click. A wrong refund costs money. A wrong price costs money and trust.
2. **Can it be undone?** An email cannot be unsent. A price can be reverted. A refund can be reversed within a window if you design for it.
3. **How often does it happen?** A thousand decisions an hour cannot have a human on each one. Ten a day can.
4. **Who owns it today?** If a person owns it now, that person is your evaluator later.

Three tiers of autonomy

TIER	THE AGENT MAY	EXAMPLE
------	---------------	---------

Suggest	Draft, score, rank. A human acts.	Reply drafts, lead scoring, price suggestions
Act within bounds	Act alone inside hard limits set in code	Refunds under a cap, price moves under a step limit
Act and report	Act freely, log everything, humans audit samples	Search ranking, internal routing

Every decision starts in Suggest. It earns its way to the next tier with evidence, which is what lessons 3 and 4 are about.

Do this today: write the decision inventory for one agent you are building or buying. One page. If you cannot fill in the "wrong answer cost" column, you are not ready to automate it.

Guardrails that live outside the model

A rule inside the prompt is a suggestion. A rule in code is a guardrail. Know the difference and build both.

By Nimit Solanki · 5 min read

"We told the model not to do that" is the sentence I hear most often after an incident. Instructions in a prompt shape behaviour; they do not guarantee it. A guardrail is a check the model cannot talk its way around because the model never gets to see it.

Where guardrails sit

1. **Before the model:** input validation, allow-lists, rate limits, PII stripping.
2. **Around the model:** tool permissions. The agent only has the tools you hand it, scoped to the minimum.
3. **After the model:** output checks in plain code. Caps, floors, ceilings, policy rules, format validation.
4. **At the action:** the system that executes the action re-validates. Never trust the orchestrator alone.

The refund example

In the demo on my homepage, the agent proposes a refund amount. Three checks run after it, in code, with no model involved: a per-agent cap, a fraud score threshold, and a policy check. If any fails, the money does not move. The model's opinion is an input to that pipeline, never the final word.

Writing a guardrail

```
def check_refund(proposal, order, customer):  
    if proposal.amount > CAP_PER_AGENT:  
        return clamp(proposal, CAP_PER_AGENT, reason="cap")  
    if customer.fraud_score > FRAUD_LIMIT:  
        return route_to_human(proposal, reason="fraud_score")  
    if not policy_allows(order, proposal):  
        return reject(proposal, reason="policy")  
    return approve(proposal)
```

Notice what is not here: no prompt, no model call, no natural language. That is the point.

Do this today: for each decision in your inventory, write the three checks that must pass in code before the action executes. If you can only think of prompt instructions, you have a suggestion, not a guardrail.

Step limits, so a wrong call stays small

The most useful guardrail is boring: no single decision may move anything by more than a bounded amount.

By Nimit Solanki · 5 min read

A pricing agent that can set any price is terrifying. A pricing agent that can move a price by at most 8% per cycle is a colleague. Same model, same data. The difference is one number.

Why step limits work

Agents fail in two ways: they are wrong, or they are wrong repeatedly in the same direction. A step limit makes the first harmless and the second visible. If a price drifts 8% a day for four days, a human sees a 30% move in the dashboard and asks why, long before it becomes a catastrophe.

Choosing the step

- Start with the largest change a competent human would make without asking anyone.
- Halve it for the first month.
- Widen it only when your evaluation data says the agent's moves were right more often than the humans' were.

Step limits for things that are not prices

DOMAIN	STEP LIMIT
Outreach	Max sends per day, per segment, per agent

Refunds	Max amount per decision, max total per hour
Inventory	Max units reallocated per run
Content	Max pages changed per publish

At estaie, step limits let the pricing agent automate roughly 70% of pricing work. The remaining 30% is exactly the moves that exceeded the limit and went to a person. That split is a feature.

Do this today: add one number to each decision in your inventory: the maximum change per cycle. Put it in config, not in the prompt.

Evaluation before anything ships

If you cannot score a change against a fixed set of real cases, you do not know what you shipped.

By Nimit Solanki · 5 min read

Teams evaluate agents by trying a few prompts and nodding. Then they change the model, the prompt or a tool, try a few more prompts, and nod again. Nobody can say whether the agent got better. The fix is an evaluation suite, and it is cheaper than it sounds.

Build the suite from real cases

1. Collect 50 to 200 real inputs the agent will face. Not synthetic. Real orders, real tickets, real listings.
2. For each, record what a good decision looks like. Often this is what your best human did.
3. Write a scorer. Exact match where possible, a rubric where not.
4. Run every change through it. Prompt edit, model swap, new tool, all of it.

What to measure

- **Decision accuracy:** did it choose what the human would have?
- **Guardrail hits:** how often did a check fire? Rising hits mean the agent is drifting.
- **Cost per decision:** tokens and tool calls.
- **Latency:** customers notice.

The eval suite is the spec

Once it exists, the eval suite becomes the contract between the model team and the business. The business says "it must get these cases right". Engineering says "it does, here is the score". Arguments about whether the agent is "good enough" turn into a number.

Do this today: pull 50 real cases from last month, label them, and run your current agent. The score will surprise you.

Cutting AI cost 10x with a model cascade

Most of the work an agent does is sorting, not thinking. Do not pay frontier prices for sorting.

By Nimit Solanki · 5 min read

Revana, the lead intelligence agent at estaie, reads thousands of signals to find a handful of real buyers. Sending every signal to a frontier model would cost more than the leads are worth. Instead it runs a cascade, and the cost per qualified lead dropped ten times with better precision.

The three tiers

TIER	WHAT RUNS	SHARE OF TRAFFIC	COST
1	Embedding similarity filter	100% in, ~28% out	Near zero
2	Small classifier or small model	28% in, ~9% out	Cents
3	Frontier model with tools	9% in	Real money, worth it

Rules for a good cascade

- Each tier must be allowed to say "not sure" and pass the item up, never down.
- Measure precision and recall at every tier with the eval suite from lesson 4. A cheap filter that drops good leads is not cheap.
- Log what each tier dropped. Sample the drops weekly. That is where your false negatives hide.

Where cascades apply

Support triage, lead qualification, content moderation, document routing, search re-ranking. Any flow where most inputs are obviously irrelevant and a few deserve real thought.

Do this today: look at your agent's last 1,000 calls. What fraction needed a frontier model to decide? If it is under half, you have a cascade to build.

Reversal paths and audit trails

Every action an agent takes should be undoable for a window, and explainable forever.

By Nimit Solanki · 5 min read

The question a board asks after an AI incident is not "why did the model do that?" It is "can we undo it, and can we see what happened?" If both answers are yes, the incident is a Tuesday. If either is no, it is a crisis.

Reversal paths

- **Snapshot before acting.** The price before the change. The ticket state before the reply. The ledger before the refund.
- **A rollback window.** 24 hours is a good default. Within it, one click restores the snapshot.
- **Batch rollback.** If an agent made 400 bad decisions overnight, you need to reverse them as a group, by agent and time range, not one at a time.

The audit record

For every decision, store: the inputs the agent saw, the proposal it made, every guardrail that ran and its result, the final action, who or what approved it, and the snapshot. Keep it in a place the agent cannot write to.

```
{
  "decision_id": "px-20261017-0412",
  "agent": "pricing",
  "inputs": {"demand_index": 0.84, "competitor_gap": -4},
  "proposed": {"change_pct": 11},
  "guardrails": [{"step_limit": "clamped to 8"}, {"floor": "ok"}, {"parity": "ok"}]
  "action": {"change_pct": 8, "new_price": 412},
```

```
"snapshot": {"old_price": 381},  
"reversible_until": "2026-10-18T04:12:00+04:00"  
}
```

This record is also what makes governance possible. Compliance, finance and customers can all be answered from it.

Do this today: pick one agent action and write the snapshot and audit record for it. If the action cannot be snapshotted, it should not be autonomous yet.

Real incidents, and what we changed

Three things that went wrong running agents in production, and the fix for each. Names and numbers simplified.

By Nimit Solanki · 5 min read

Nobody posts about the incidents. They are the most useful part. Here are three patterns I have lived through, described generically so the lesson survives.

1. The confident wrong move

What happened: an agent proposed a large change on thin data, with a perfectly reasonable explanation. The explanation was convincing. The data was not.

What we changed: step limits (lesson 3) so the move was capped, and a data-sufficiency check before the model ran. Fewer than N signals means the agent may suggest but not act.

2. The drift nobody noticed

What happened: after an upstream data source changed its format, the agent's decisions slowly got worse over two weeks. No single decision looked wrong.

What we changed: guardrail-hit rate on a dashboard with an alert. Drift shows up as rising hits long before it shows up in revenue. Plus the eval suite (lesson 4) runs nightly against fresh cases, not just at release.

3. The tool it should not have had

What happened: an agent used a tool for a purpose nobody intended, because the tool was available and the task was ambiguous.

What we changed: minimum tool scope per agent (lesson 2), and a tool-use log reviewed weekly. Every tool call is a decision too.

The pattern

In all three, the model was not the problem. The missing piece was always a boring control around it. That is the whole playbook in one sentence: production agentic AI is about 20% model and 80% trust design.

Do this today: write down the incident you are most afraid of with your agent. Then check which of lessons 1 to 6 would have caught it. If none, that is your next task.

What next

If you want this applied to your company, that is what I do: AI safety reviews, AI strategy, and mentoring the engineers who have to build it. [Book a call](#).