Domain General AI: Simple Design for Complex Decisions

You never get full information; you get scraps. Sometimes you only hear a hum and you must move anyway. The test of a good design is whether decisions stay sane when the inputs are partial.

Start with the faint pitch

On a late night run, the streetlights were out on one stretch and my watch had died. I could only hear my breath and the road's texture underfoot. No charts, no maps, just a faint pitch in the blackness. I realized how often I'd tried to make the world simpler instead of making my way of moving through it simpler. That night, I stopped fighting the dark. I focused on the one signal I could trust and let everything else be varied and messy. The change felt small. It wasn't. It became a stance: keep my design simple and general; let the world be complex.

> The faint signal is the earliest form of strategic clarity; strengthen it with small, reversible experiments that expose causality faster than noise and narrative can distort it.

Define domain general AI

Domain general AI is a design stance where the agent's core logic is independent of any specific task or environment. The agent stays simple and general; the world is allowed to be complex. Performance comes from learning the particulars on the fly, while we understand the agent through enduring principles.

A few crisp meanings so we can work. Domain general means a design that doesn't depend on the details of any particular environment to function. Conceptually simple understanding is a high-level way to explain what the mind is doing using principles, not the fiddly specifics it learns. Signal versus noise separates information that reliably changes outcomes from variation that doesn't, your job is

to tell them apart before you commit big resources.

The point: the agent's core should be general and comprehensible. The world is allowed to be baroque, your design isn't. You measure progress by how traceable your reasoning stays as complexity rises.

Decision making under uncertainty

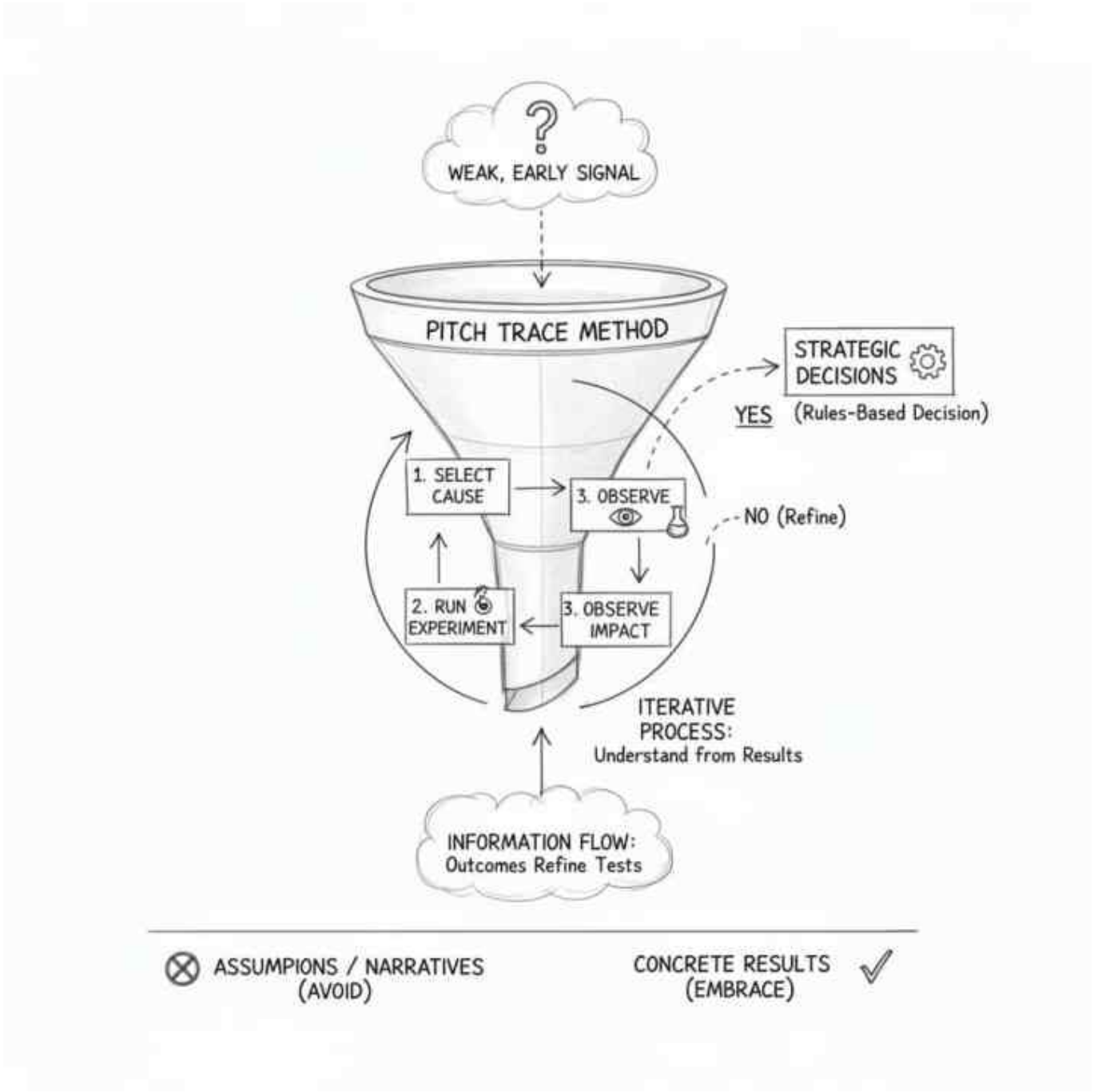
Direct response is the human version of prompt engineering, it creates the conditions for action, removes ambiguity, and aligns desire with the outcome. One move that works: pre-commit to principles you won't trade away, reversibility, small blast radius, and learn-first. Then choose actions that respect those constraints, even if they score fewer points today.

Example: choosing a pricing change. Instead of a full switch, gate the new price behind a private link for a small segment. Keep the setup under 2 hours and the reversal to a single toggle. You'll learn willingness-to-pay without risking your entire base.

What is the Pitch Trace Method?

The Pitch Trace Method is a simple way to follow the earliest reliable signal before it's loud. You identify one narrow cause you can test quickly, run a reversible trial, and decide using pre-agreed rules. Then you trace the next small cause. Repeat until the map emerges from results, not from hope.

It's not fancy. It's discipline. You protect the core (simple and general) and let the world's details reveal themselves by how they change outcomes.



Designing experiments instead of chasing certainty

You can't outrun uncertainty; you can outlearn it. Trade big, brittle bets for compact

trials that compound. Pick one outcome that matters now and define one observable shift that would count as progress. Limit moving parts by changing one lever while keeping the rest stable, cap every test at three variables you actively track. Shorten the loop by favoring tests that complete in under 7 days, and if a setup takes more than 2 hours, you're optimizing the wrong thing. Make the exit cheap because reversible means reversible, one click, one email, one line change.

Example: exploring a new onboarding flow. Instead of redesigning the whole experience, change the first screen copy for a subset of new users and measure completion to the next step. End the test at 7 days or a predetermined sample size, whichever comes first.

Why this works: causality hides under layers of story and variance. When you cut test size, duration, and the number of levers, you make cause easier to see. When you insist on a clean exit, you can try again tomorrow.

How to separate signal from noise

A simple question clears fog: if the result is true, what else would change? Then look for that echo. Use paired indicators, if a new outreach email increases replies, check qualified calls booked. Signal tends to rhyme across measures; noise doesn't. Time-shift your checks because a fast bump with no follow-through is often noise, while a slower build that shows up in a second measure is often signal.

Example: you reduce a trial period from 14 to 7 days. Sign-ups dip slightly, but paid conversions rise and retention at day 30 holds. The echo (retention) supports the cause (tighter trial). Noise would spike sign-ups without improving paid conversion or retention.

Apply it in the wild

Founder anecdote: I once tried to “fix” growth with a dozen changes at once, pricing, onboarding, ads, messaging. We “won” a few weeks, then hit a wall. Nothing was traceable. I restarted with one change per week, reversible by design. In six weeks, we kept two wins and learned three real constraints. The team's stress dropped because the work was finally legible.

Micro-slice examples show the pattern in action. A sales lead thought long proposals

were killing deals. Instead of a rewrite, they tested a one-page brief for ten handpicked prospects. Setup was under 2 hours; reversal was sending the old deck. Result: shorter sales cycles with no hit to close rate. They kept the brief. A product team suspected weekend tickets were draining morale. They trialed a Saturday-only chatbot for one month for a subset of categories and kept Sunday human-only. Resolution time held steady; agent burnout metrics improved. They kept the split.

Objections & failure modes

Isn't this too cautious? We'll get outpaced. It's fast learning, not slow movement. You compress cycles by making them small and clear. The bottleneck is usually ambiguity, not speed.

Real performance needs domain tricks. Domain knowledge matters, the point is where it lives. Let the agent learn specifics; keep your design and reasoning principle-led so it stays portable when the domain shifts.

Simple sounds naive. Intelligence might be irreducibly complex. The world is complex. Your core doesn't have to be. Simplicity here means legibility, designs you can explain and adjust without inheriting every quirk of the environment.

We tried testing; results conflicted. That's a design smell. Reduce moving parts, tighten duration, and pair indicators. If results still conflict, your “cause” is too broad.

> When you operate this way, you stop needing the world to behave and start requiring your choices to be explainable.

Close the loop

That's the far side of complexity: a simple, principled center that can walk through changing terrain. You follow the faint pitch in the blackness, not because it's romantic, but because it's the only reliable guide you have.

If this stance helps, I send one short email each week with a principle you can apply in under 10 minutes, plus one tiny test to try. It's low-risk and practical: ideas over hype, cause over noise, small steps you can reverse. Join for a few weeks and

decide.

Build the agent simple; let the world be complex, start your first reversible test today.

Here's something you can tackle right now:

Pick one outcome that matters now. Define one observable shift that would count as progress. Design a reversible test under 7 days that changes only one lever.