NO CODE BOOTCAMP FOR BUSINESS LEADERS

ADVANCED AGENTIC AI AND LLMS

Raj Lal · Emily Mao · Yash Sanghvi - Introduction by Yuliya Roshko

No-code bootcamp for business leaders · Igniter Silicon Valley

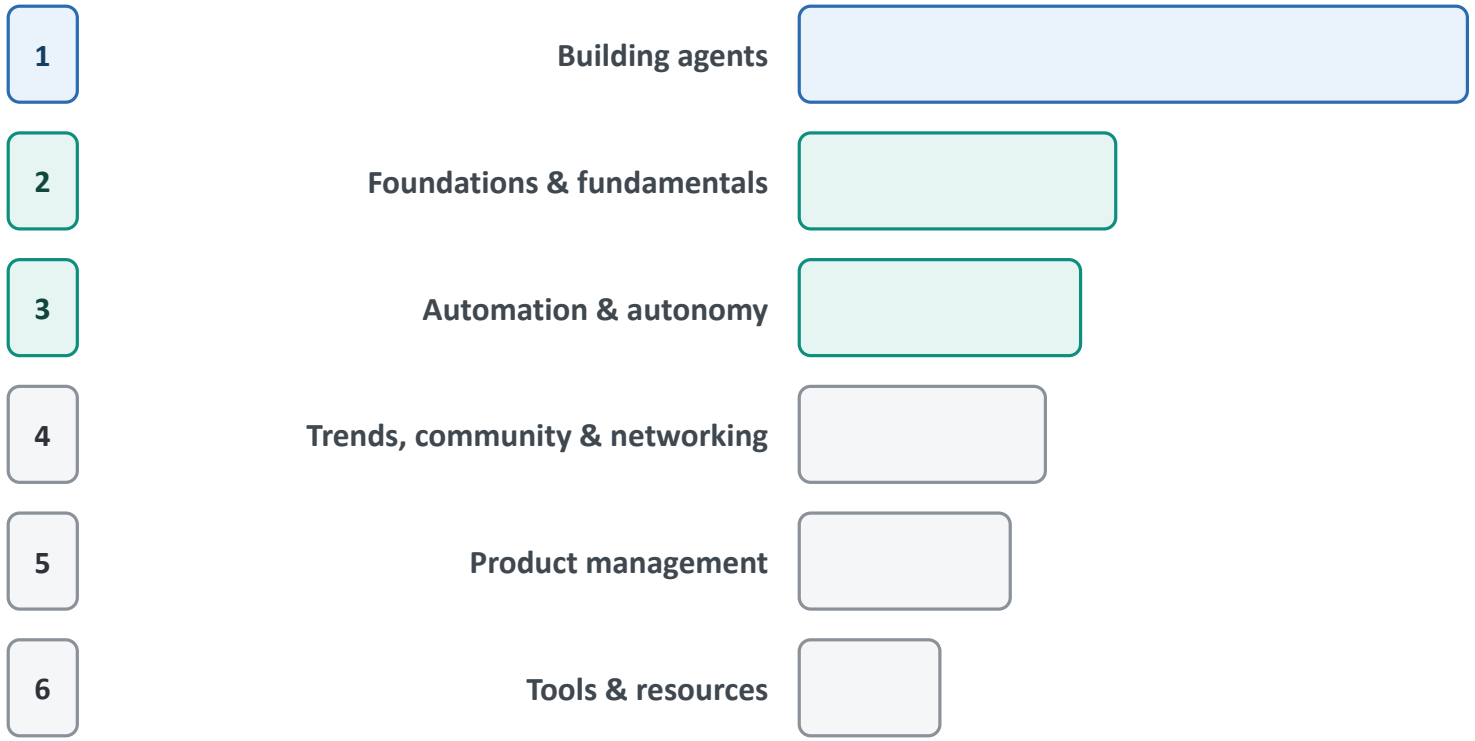
anci.app

IGNITERSV.COM · IGNITERCON.COM

A startup community in the Silicon Valley

YOU ARE PRODUCT MANAGERS, ENGINEERS AND FOUNDERS

Your - What do You want to Learn?



READ THIS

- **Nearly a third said "build agents"**

  19 of 66, often with no further detail. That is not a lack of interest, it is a missing vocabulary.

- **Only 6 asked about the hard part**

  Guardrails, evals, production and lifecycle combined — where projects actually die.

Every slide in this deck came from an answer one of you wrote on the registration form.

SEVENTY MINUTES

What we are doing today

20 min

AI Agents

Where are we now

20 min

Parallel Agent

Hands on

20 min

Router Agents

Hands on

20 min

Dynamic Agents

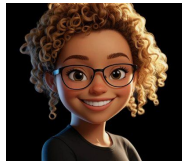
Hands on

Twenty minutes of background, sixty minutes of building. If you only remember one thing from the theory, make it the slide marked CORE.

Everything after the hands-on sections is reference material — use it when you get back to your own codebase.

WHO IS TALKING

My story — from Chatbot to AI agent



Zara and Mia
AI Agents
2026

Meetbot 2025

ANCI

Welcome to ANCI AI
The World's Most Advanced
AI Scheduling Software

RAJ LAL

- **Founder & CEO, ANCI AI**

  Agentic AI developer. Building scheduling agents that actually run unattended.

- **ADI MeetBot (chatbot)**

  A Scheduling Chatbot, Available NOW in **Slack** and **Microsoft Teams**

- **Zara and Mia - AI Agents for Scheduling**

  An agent that joins the chat, suggests agenda items and reschedules around the team.

- **ANCI AI**

  The world's most advanced AI scheduling software — anci.app

  And along the way a lot of production failures that turned into the slides after the break.

Everything in this deck is something that broke on me first.

Raj Lal · Founder / CEO - ANCI AI www.anci.app

01

TEN MINUTES OF THEORY

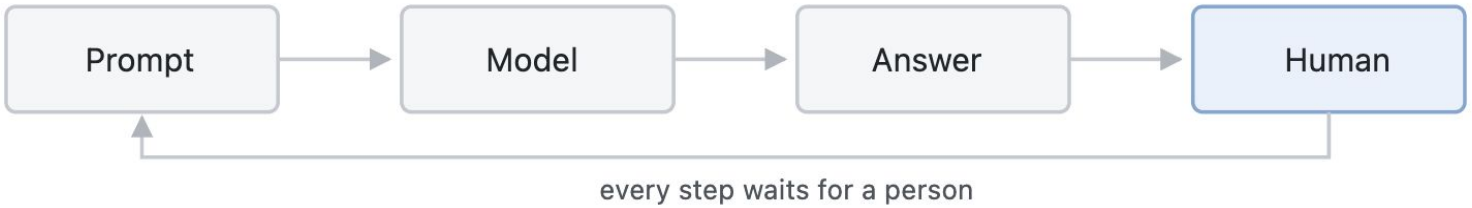
AI Agents … and Chatbots

The distinction that makes every other decision in this workshop make sense.

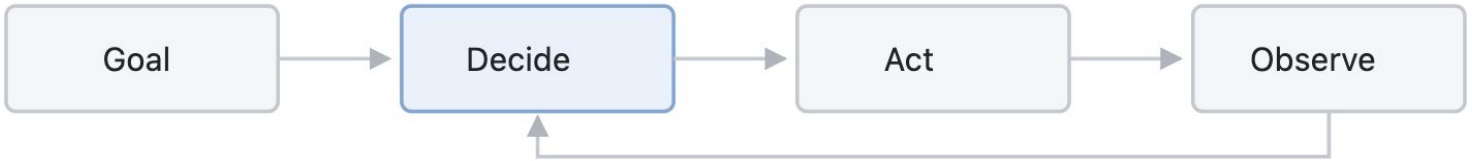
CORE 1 / 7

What actually is an agent?

CHATBOT — you are the loop



AGENT — the loop closes itself



human optional

HITL — human in the loop is optional, and that is the whole point

THE FOUR LEVELS

- **AI as a tool**

  Grammarly, Ideogram. You are in the driver's seat; it executes, you decide.

- **AI as an assistant**

  Drafts the email, suggests the meeting time. You make the final call.

- **AI as a partner**

  Claude Code, drug discovery. It proposes, you refine together.

- **AI as a teammate**

  Zara AI joins the thread, takes ownership, reports progress.

The practical test: remove the human from the middle. If anything still happens, it is an agent.

What are the different components of an AI Agent

CORE 2 / 7

The five components — and the one everyone forgets

1 · Model

Decides the next action. Swappable — design assuming you will swap it.

2 · Tools

How it touches the world. Most agent quality actually lives here.

3 · Memory

Run state, plus what survives between runs.

4 · Orchestration

The loop: continue, stop, retry, escalate.

5 · Evaluation

The forgotten one. Without it every change is a guess.

FOUR THINGS WORTH SAYING

- **Great tools beat a great model**

  Narrow, well named, structured results, clear errors. get_customer_by_email beats query_database.

- **Route per step, not per project**

  A cheap model for the classification steps, an expensive one for the single hard step.

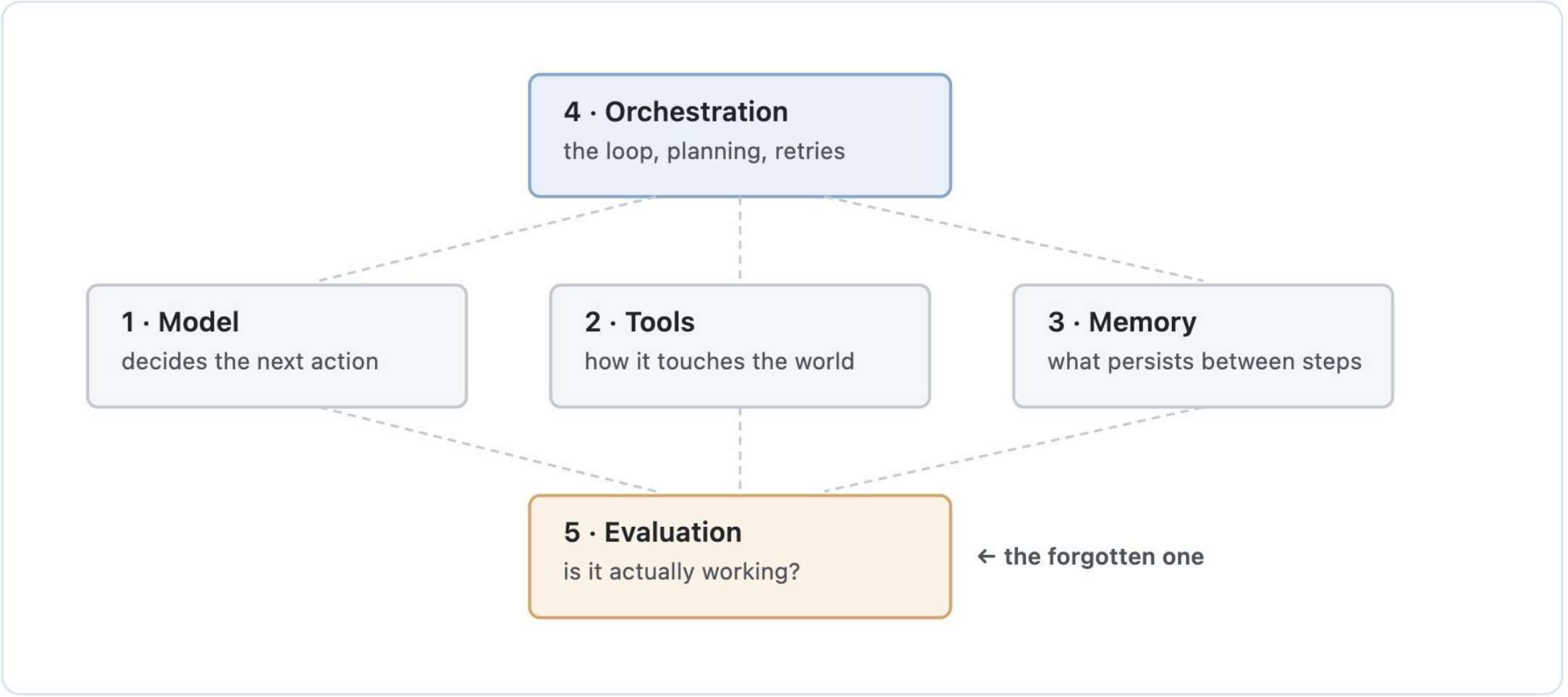
- **Models are stateless with no Memory**

  Everyone builds run state and skips what survives between runs. That is why it has amnesia every morning.

- **The swap test**

  Can you change the model in an afternoon? If not, model assumptions have leaked into your tools.

Nearly everyone builds 1 to 4, ships, and then cannot answer "is it getting better or worse?"



Nearly everyone builds 1 to 4, ships, and then cannot answer "is it getting better or worse?"

What are the Challenges with building an AI Agent

THE HONEST LIST

Challenges with building an agent on an LLM

Hallucination

Confident, well-formed, wrong. Usually it was asked something it had no way to answer.

Expensive

1 token ≈ 1 small word ≈ cost. You pay for the whole context on every loop iteration.

Context rot

Curation is needed at every single step, not once at the start.

Trained on past data

It does not know your Q3 numbers, your codebase, or what happened last Tuesday.

Bad at arithmetic and time

Dates and numbers are lookups, not reasoning. Route both to code.

Non-deterministic

Same input, different output. "It worked when I tested it" stops meaning anything.

AND THREE NOBODY LISTS

- **Silent failure**

  Normal code screams when it breaks. An agent produces a confident, well-formed, wrong answer and moves on. Nothing throws.

- **Prompt injection**

  The moment it reads a document, a webpage or an email, that input can contain instructions.

- **Uniform confidence**

  It sounds exactly as sure when guessing as when certain. That is what makes the errors dangerous.

Everything on this list is why the system around the model exists. Half of it is solved by tools, not a better model.

What LLMs are genuinely excellent at?

THE OTHER SIDE OF THE LEDGER

What LLMs are genuinely excellent at

Language

Rewriting, translating, tone, summarising, drafting. Unmatched, and cheap.

Messy input

Typos, half-sentences, three date formats in one file. A parser breaks on all of it. This does not.

Zero-shot

Handles categories you never enumerated. No labelled data, no retraining, no new class definition.

Never bored at item fifteen

Ask for twenty ways a plan fails and you get twenty, all considered. Humans are worst at exactly this.

THE DESIGN RULE

- **Play to the strength**

  If the task is "read this and tell me what it says", you are on solid ground.

- **Route the rest to code**

  Maths, dates, sorting, lookups. Cheaper, exact, and testable.

- **Cheap to prototype**

  Idea to working demo in an hour, with no data collection phase in front of it.

Everything on this list is what the model is for. The best agent tasks are the ones a smart intern could do, if only they had the context.

How to do LLMs correctly?

TWO HABITS THAT FIX MOST PROBLEMS

How to do LLMs correctly

Let LLM solve the language problems

Extraction, classification, triage, summarisation, reconciliation of two documents that should agree. Every one of those is a language problem wearing a business costume — and every one of them is a task nobody enjoys doing by hand.

Ask for output in a structured format

Stop asking for prose you will parse with a regex. Ask for a defined schema and validate it. If it does not validate, retry automatically. You have just turned a soft failure into a caught error — which is the entire game.

What LLM should do on failure

Most prompts describe success only. Add the line that says what to do when it cannot: "if the document has no renewal date, return null — do not infer one." That single sentence removes a large share of hallucinations.

Use Guardrails

Input Constraints, Execution Constraint, Output Validations, Judge Layer.

AND FOUR MORE, BY PAYOFF

1 · Errors are inputs

"Error: 400" teaches nothing. "date must be YYYY-MM-DD, you sent next Tuesday" produces a correct retry.

2 · Ask for a confidence field

{answer, confidence, reason}. Low confidence routes to a human. Accuracy becomes a workflow problem.

3 · One task per call

Five instructions in one prompt gets you three and a half. Split them.

4 · Version prompts like code

Change one, re-run the eval. Nobody edits a production prompt at 11pm.

If you change only two things after today, change these two.

02

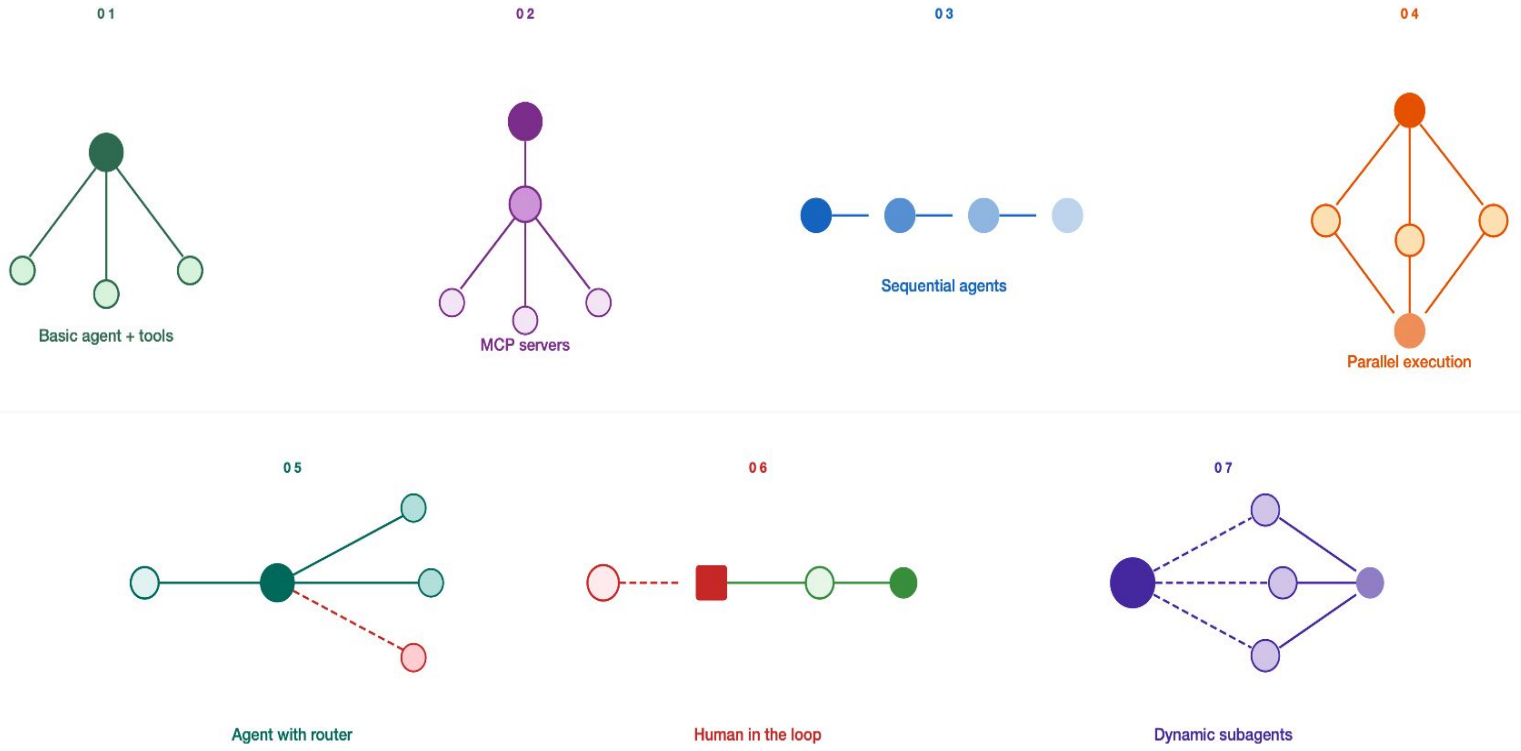
ARCHITECTURE

Types of Agents

Seven architectures. Three of them are what you will build in the next hour.

7 AI Agent Architectures

GPT · GEMINI · CLAUDE · ANY AI MODEL



TODAY YOU BUILD THREE

- **Parallel agent**

  Independent subtasks running at once, merged at the end. Wall-clock drops.

- **Router agent**

  One agent classifies and dispatches to specialists with different tools and permissions.

- **Dynamic agent**

  The plan itself is decided at run time rather than hard-coded.

AI TRENDS FOR LEADERS

Whatever the shape, every one of these decomposes into the same five components from CORE 2.

03

SIXTY MINUTES

Hands on

Three agents, three architectures. Build them, break them, and read the logs.

Router Agent

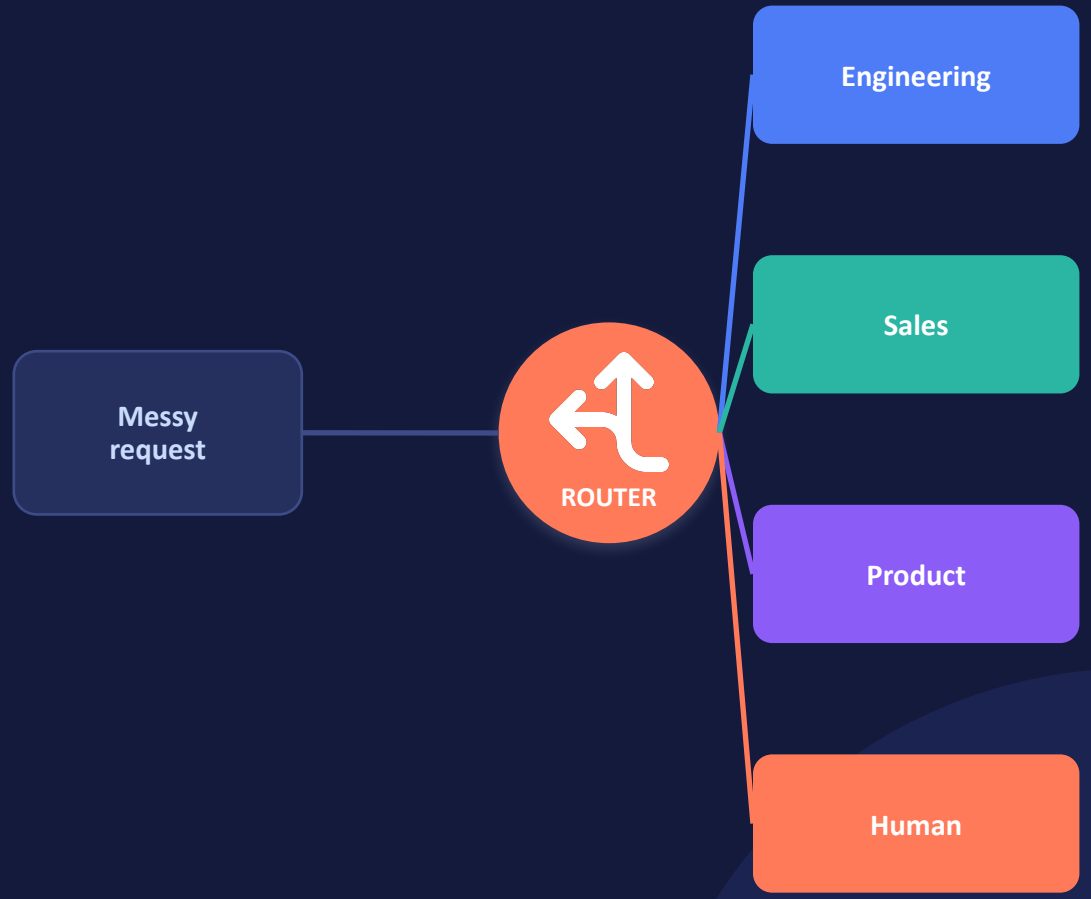
- With Yash Sanghvi

anci.app/agent/type5.html

Stop Being Your Team's Router

It doesn't forward your mail. A router agent reads one messy request, splits it apart, and sends each piece to the right specialist.

23% of executive time vanishes into this triage today.



HOW A REAL ROUTER AGENT THINKS

Three levels of routing.

1

Classify

Reads meaning and intent — not keywords. Knows “about to cancel” is churn, not a billing question.

2

Decompose

One message usually hides several intents. It pulls them apart into separate, distinct tasks.

3

Tailor & dispatch

Rewrites each piece for its specialist and sends only what’s relevant. Strips out the rest.

A “CC rule” stops at level 1. A router agent does 2 and 3 — that’s the difference, and what we’re about to watch.

LIVE DEMO · ONE MESSY EMAIL

How many issues are hiding in here?

From: a real customer

"Hi, I love your product but **the new dashboard is crashing when I export to PDF**. Also, because we are growing, **I need to add 50 more seats to our enterprise plan**. Finally, **your documentation on API rate limits is totally outdated and confusing**."

Shout it out — then watch the agent pull them apart and route each one.

Bug Upsell Docs

STEP 1 — IT ANALYZES

The routing log.

1	Dashboard crashes on PDF export	→ Engineering	conf 97%
2	Wants +50 enterprise seats ⚠️ revenue — needs human approval	→ Sales	conf 95%
3	API rate-limit docs are outdated	→ Product / Content	conf 92%

STEP 2 — IT DRAFTS

Three emails. Three audiences. Written.

→ Engineering

Subject

Bug: dashboard crashes on PDF export

Customer reports the new dashboard crashes on PDF export — likely blocking reporting. Please repro on the current build and triage.

→ Sales

Subject

Expansion: +50 enterprise seats

Growing account wants +50 seats. Warm upsell.

⚠️ hold for approval

→ Product / Content

Subject

Docs feedback: API rate-limits outdated

Customer flagged the API rate-limit docs as outdated and confusing. Worth a review and refreshed examples.

Each team gets only its slice — rewritten for them. Nobody triages anything.

DO IT NOW · YOUR OWN MESSY EMAIL

Try it — then point it at your messiest inbox.

PASTE THIS PROMPT

```
You are a Router Agent. I'll paste ONE messy
customer message with several different issues.

1) ROUTING LOG — for each distinct issue, give:
   intent · department (Engineering / Sales /
   Product / Success / Escalate-to-human) · confidence %

2) DRAFTS — one short, ready-to-send note per
   department, containing ONLY that department's
   issue, rewritten for them. Strip the rest.

Rule: tag any revenue or high-stakes item
"needs human approval before sending."

Message: [paste a messy, multi-issue email]
```

Scan → opens Claude

1. Scan the code (or open any AI chatbot).
2. Paste the prompt on the left.
3. Write a messy email with 2–3 issues.
4. Watch it split apart and draft each one.

It even flagged the revenue call for a human — your control layer, and exactly where the next talk begins.

HANDS ON · 20 MINUTES

Router agent

1 · Classify

The router reads the request and decides which specialist should own it.

2 · Dispatch

Hands off with just enough context — not the whole conversation.

3 · Specialists act

Each has different tools and different permissions. That isolation is the point.

TRY ALONG anci.app/agent/type5.html

The thing that will bite you

Misrouting is silent. The specialist will confidently answer a question it should never have received. Log the routing decision separately so you can tell "wrong answer" from "wrong agent".

WHAT TO WATCH

- **Routing accuracy is its own metric**

  Track it apart from answer quality, or you will debug the wrong layer.

- **Permissions are the real reason to split**

  Not personality. Access.

- **Ambiguous requests**

  What does your router do when the request fits two specialists? Decide that on purpose.

A router is a classifier with consequences. Treat it like one.

Parallel Agent

- With Emily Mao

anci.app/agent/type4.html

HANDS ON · 20 MINUTES

Parallel agent

1 · Fan out

A coordinator splits the goal into subtasks that do not depend on each other, and dispatches them all at once.

2 · Run concurrently

Each branch gets its own context and its own tools. No branch waits for another to finish.

3 · Merge

An aggregator reconciles the branches into one answer — and decides what to do when they disagree.

anci.app/agent/type4.html

The thing that will bite you

Partial failure. One branch times out and the merge either blocks forever, or quietly returns an incomplete answer that looks complete. Decide up front: does the merge wait, proceed with what it has and say so, or fail the whole run?

WHAT TO WATCH

- **Only fan out what is truly independent**

  If branch B needs branch A's output, this is a sequence wearing a parallel costume.

- **Cost multiplies, time does not**

  You pay for every branch but you only wait for the slowest one. That trade is the whole point.

- **The merge is the hard part**

  Reconciling branches that disagree is a JUDGE step, not a MOVE step. Budget for it.

Fan-out is the one multi-agent pattern that is always worth it — the win is wall-clock, and wall-clock is not a matter of opinion.

PARALLEL DEMO · TYPE 4

What we are building: a competitive intelligence dashboard

Stage 1 · Coordinator

Promise.all() launches three research operations at the same moment: Calendly, Reclaim.ai and Motion.

Stage 2 · Merge agent

Aggregates the three JSON files into one analysis — profiles, a scoring matrix, ranked signals, white-space gaps.

Stage 3 · Dashboard agent

Renders it as HTML: competitor cards, comparison matrix, urgency badges, positioning directives.

What each research agent gathers

GitHub activity · product positioning · pricing tiers · blog posts · job listings. Five public sources per competitor — no API keys and no authentication, just the GitHub REST API and web fetch.

WHY THIS SHAPE

Three competitors, zero dependencies

None of the three lookups needs another one's answer. That is what makes the fan-out legal.

Nothing to configure first

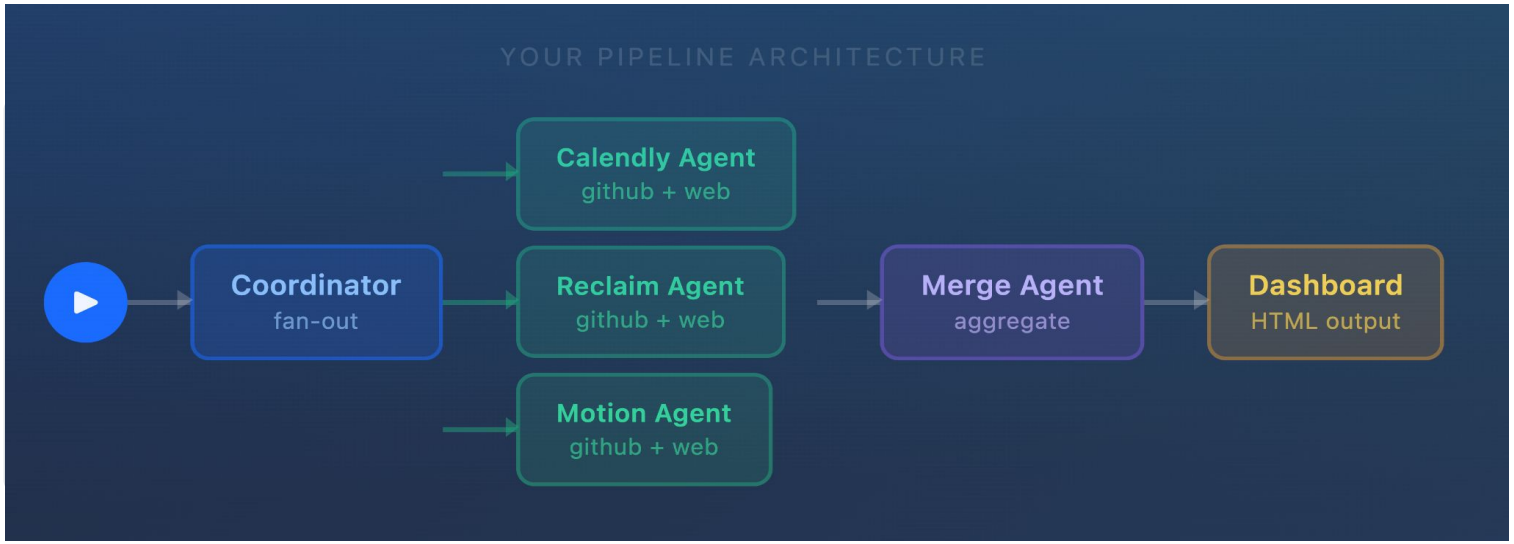
npm install -g @anthropic-ai/claude-code, then mkdir anci-intel. That is the whole setup.

The demo measures itself

It prints parallel time against the sequential equivalent at the end. Watch that number.

You are not building a chatbot. You are building something that produces three files you can open.

Input, pipeline, output



IN — a terminal and a folder

Claude Code installed, an empty working directory, and three competitor names. No keys, no auth, no config file.

OUT · raw data

competitor_data.json — fetch results + timing

OUT · analysis

competitive_analysis.json — merged strategy

OUT · dashboard

anci_competitive_dashboard.html

READ THE DIAGRAM

One coordinator, three branches

Calendly, Reclaim and Motion are queried at the same moment, not one after another.

They never talk to each other

That independence is what makes the fan-out legal. Add one dependency and it collapses to a sequence.

Merge, then render

The aggregator produces the answer. The dashboard is only the destination — swap it for Slack or Notion.

Three lookups that do not depend on each other. Running them in sequence would waste wall-clock for nothing.

PARALLEL DEMO · YOUR TURN

Four things to change once it runs

1 · Swap the competitors

Replace Calendly, Reclaim and Motion with the three companies that actually keep you up at night.

2 · Change what it gathers

Right now: GitHub, positioning, pricing, blog, jobs. Add a source, drop one, and see what the merge does with less.

3 · Widen the fan

Three branches is arbitrary. Try five. Cost rises, wall-clock stays almost flat — the pattern in one number.

4 · Break a branch on purpose

Point one agent at a URL that 404s. Does the merge still produce something useful, or does it stall?

THE REAL EXERCISE

- **Number four is the one that teaches**

  The first three teach you the pattern. Breaking a branch teaches you the failure mode, which is what you will actually hit.

- **Then change the destination**

  The dashboard agent is just a renderer. Point it at a Slack message or a Notion page instead.

- **Keep the timing line**

  Parallel versus sequential is your whole argument for this architecture. Do not delete it when you refactor.

If you only do one thing after the demo: break a branch and watch what the merge decides to do about it.

Dynamic Agent

- With Raj Lal

anci.app/agent/type7.html

HANDS ON · 20 MINUTES

Dynamic agent

The plan is decided at run time, not written by you

A dynamic agent works out its own sequence of steps from the goal and what it finds along the way. It is the most capable shape on the architecture map — and the one that most needs everything in the next section.

More capable

Handles cases you never enumerated.

Less predictable

Two runs of the same input can take different paths.

Needs hard limits

Step caps, cost caps, and a definition of done.

TRY ALONG anci.app/agent/type7.html

WHAT TO WATCH

- **Where does it stop?**

  A dynamic agent with no stopping condition is a bill, not a product.

- **Can you replay it?**

  If you cannot see what it saw at step four, you cannot debug step five.

- **This is why CORE 5 exists**

  Guardrails matter most exactly where the plan is not fixed.

Dynamic agents are where the power is and where the guardrails stop being optional.

HANDS ON · 20 MINUTES

anci.app/agent/type7.html

Dynamic agents are where the power is and where the guardrails stop being optional.

DYNAMIC DEMO · TYPE 7

What we are building: a sub-agent orchestrator for inbound leads

The main agent decides who wakes up

It reads one lead and writes spawn_plan.json — classifying deal_size_estimate, deal_tier, title_seniority and urgency. Only then does it choose which specialists to activate. Same input shape, different agents every time.

Stage 1 · Orchestrator

6 min. Reads the lead, classifies it, writes the spawn plan.

Stage 2 · Sub-agents

7 min. The activated ones run in parallel, each writing its own JSON file.

Stage 3 · Brief and email

4 min. Synthesises every output into a lead card and a draft email.

HOW THIS DIFFERS FROM PARALLEL

- **There, you chose the branches**

  Here the agent chooses them at run time. That is the whole distinction between Type 4 and Type 7.

- **A sample lead is provided**

  Sarah Chen, VP of Sales Operations at Meridian Health Partners, asking about clinical scheduling.

- **Watch the agents that did NOT run**

  The brief shows which fired (green) and which were skipped (grey). That trace is the most interesting part of the output.

Same architecture map as the parallel demo, one big difference: nobody wrote down in advance which agents would run.

Input, the decision table, output

IN — one lead profile

Name, email, job title, company, company-size signal, industry, how they found you, and a free-text message.

ICP Fit Agent · always

Scores the lead 0–100. This one and the email agent run every single time.

Company Intel Agent · if deal ≥ $10k

Funding, headcount, tech stack.

Buyer Persona Agent · if VP or above

Profiles the decision maker.

Competitor Usage · if ≥ $50k or >100 staff

Detects the tools they already use.

OUT · lead_brief.html

CRM card, ICP badge A/B/C/D, and the decision box showing what was skipped

OUT · first_email.txt

subject line naming their pain, CTA with [DEMO_LINK]

WHERE THE LOGIC LIVES

- **Four conditions. That is the dynamic part.**

  $10k, VP-or-above, $50k, 100 employees. Everything else in this demo is plumbing.

- **Keep thresholds in code, not the prompt**

  Business rules you can unit-test beat business rules the model re-derives on every run.

- **The spawn plan is the thing to read**

  If the orchestrator classified wrong, every downstream agent is confidently wrong too.

The orchestrator is not being clever. It is applying four numbers you chose — which means you can change them.

DYNAMIC DEMO · YOUR TURN

Four things to change once it runs

1 · Change the lead

Drop the title to "Analyst" or the company to twelve people. Watch agents switch themselves off.

2 · Move the thresholds

$10k, $50k, VP-or-above, 100 employees. All arbitrary. Replace them with your actual qualification rules.

3 · Add a sub-agent

A Security Review Agent, say. What would have to be true in the lead before it should activate?

4 · Feed it a lead that fits nothing

No condition triggers. Does it degrade honestly, or invent a reason to run something?

THEN TAKE IT OUTWARD

- **Wire the front door**

  A HubSpot webhook instead of a hand-typed lead. ICP definitions living in Notion rather than the prompt.

- **Wire the back door**

  Auto-send through Gmail — but only after a shadow period. This is a DECIDE step, so a human owns it first.

- **Then batch it**

  One lead is a demo. A list of two hundred overnight is the thing you actually wanted.

If you only do one thing after the demo: give it a lead that should trigger nothing, and see whether it says so.

04

THE REST OF THE DAY

From demo to production

Everything after this point is reference. It is the part six of you asked about — and the part that decides whether any of this survives.

CORE 4 / 7

Chat is a demo. Triggers are a product.

YOU ARE DOING THIS	BUILD THIS INSTEAD
Deciding when to run it	A trigger — schedule, webhook, file drop
Gathering the inputs by hand	A data connection — API, query, folder
Noticing it went wrong by reading	Validation and alerting that fail loudly
Copy-pasting the result somewhere real	A destination — doc, ticket, dashboard

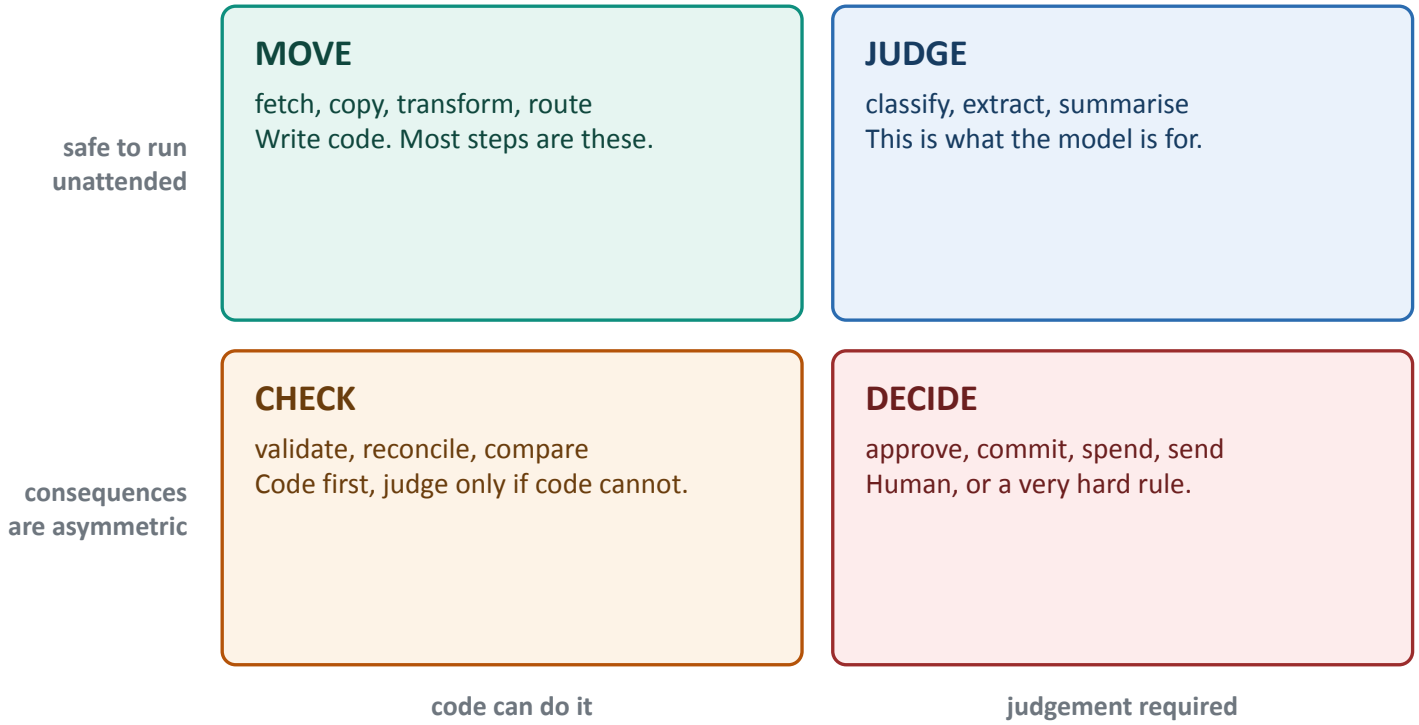
THE ONE-WEEK ON-RAMP

- **Take one repeated chat task**
  Write down the prompt you actually use today.
- **Answer four questions**
  What event starts this? Where does the input come from without me? What does obviously-wrong look like? Where should the answer land?
- **Start read-only**
  Produce something, put it where you will see it, take no action for two weeks.

In a chat session you are the scheduler, the input provider, the error handler and the delivery mechanism. Replace all four.

CORE 3 / 7

Label every step before you automate one



THE FOUR PASSES

- **List steps as they really happen**

  Including the informal ones. "And then someone eyeballs it" is a step.

- **Label each one, strictly**

  Most steps people plan to give a model are MOVE steps. Code is cheaper and correct every time.

- **Find where work queues**

  That is the bottleneck — not the step people complain about most.

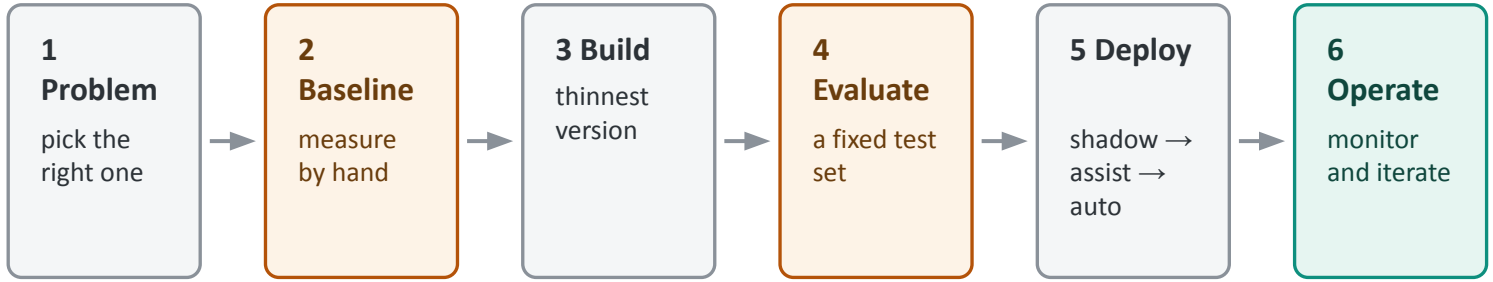
- **Automate up to a DECIDE**

  A contiguous run ending at a decision. That is your v1 and your human checkpoint.

The common error is automating one step in the middle. You add an integration on both sides and save thirty seconds.

CORE 6 / 7

The whole lifecycle



The loop that is the actual product

Every production failure becomes a new case in your eval set. Over a year that set encodes everything the system has learned — and it is worth more than your prompts.

THE SKIPPED TWO

- **Baseline: do it by hand 20 times**

  Otherwise you cannot tell whether 82% accuracy is a triumph or a disaster. Humans are often at 85%.

- **Evaluate: 50 to 200 real cases**

  With known-good answers, re-run on every change.

- **Deploy in three gears**

  Shadow, then assist, then auto. Most teams jump to gear three.

The demo is 20% of the work. Knowing that ratio upfront is what stops a project stalling at "nearly done" for three months.

CAN AI REPLACE HUMANS?

Can AI die?

Can AI imagine?

Can AI feel?

Can AI think?

Can AI want?

Can AI enjoy music or art?

> *"You can know the name of a bird in all the languages of the world, but when you're finished, you'll know absolutely nothing whatever about the bird. So let's look at the bird and see what it's doing — that's what counts."*
>
> **— Richard P. Feynman**

Stop naming it. Watch what it does.

WHERE YOU GO FROM HERE

One sentence for each of you

If you came to understand it

The model is the least important part. Learn the loop around it and you will understand every product you see for five years.

If you came to ship it

Your test set is the product. Build it before you build anything else and everything downstream gets easier.

If you came to govern it

Write down what correct means and attach a human name to it. That is more than most governance programmes manage in a year.

Pick one boring, repetitive task. Follow it all the way through to something that runs without you.
It will be broken by Thursday — and that is when you actually start learning.

THANK YOU

Stay in the know

Join our Newsletter

AI Edge for Leaders — Cutting-edge AI insights and productivity strategies.

Bay Area Community

Stanford Igniter — Check upcoming events.

[AI Edge for Leaders — newsletter]

[Eventbrite — upcoming events]

[anci.app]

Scan to sign up

Raj Lal

Founder & CEO, TEAMCAL AI

raj@teamcalendar.ai

Sign up for "AI Edge for Leaders" to stay in the know — and come back for the next Igniter session.