/erica-layer

AI ON PURPOSE

A PLAYBOOK

The AI-Native Inbox Playbook

A start-to-finish guide to rebuilding your inbox inside an AI you work with, so that email and messages arrive in one place, already sorted and drafted, and the work in them gets done rather than described back to you. Ten steps in three arcs, with the decision behind each one and the traps to avoid.

Erica Layer

layeradvisory.com

Dear reader,

In April I built an AI executive assistant over a long weekend and wrote a post about it. Every morning after that, Claude read my email, sorted it, and drafted the replies I needed to send. One of the comments on that post said these assistants are impressive for about a week, and then you realize your problem was never email. At the time I disagreed. I think now he had a point.

What I had built was a very good way of managing my inbox, and every piece of actual work in it still belonged to me. The drafts sat inside a chat conversation. To send one I still had to open Gmail. When I closed the conversation, everything Claude had worked out went with it. I was running four inboxes (my own practice, two client organizations, and a nonprofit) and opening every one of them by hand, on top of whatever Claude had already told me about them.

In August I rebuilt it so that Claude does the work. Everything arrives on one screen, most of the replies are already written when I get there, and when I hand something over it becomes a task the agent owns, does, and reports back on. I haven't opened Gmail properly in weeks.

Two things I want to say before you start.

Nothing in this system sends without my approval on that specific email. That rule is the foundation, and everything else is built on top of it. I'd ask you to keep it, even when the drafts get good enough that approving feels like a formality.

And I still edit most of what it writes. Of the first thirty replies I sent through it, I changed twenty-eight before they went. That's the point, not a failure of the system: every edit becomes a rule it follows next time, so the drafts get closer to me with every week I use it. The judgment stays mine. The typing doesn't.

I wrote down what I actually did at each step, including the places it went wrong, because I could see there was a lot in here that other people could use. I'm not a developer, and none of this needed me to be one. If you put the time in, I think you'll end up with something you trust.

With gratitude,

Erica

Principal, Layer Advisory

layeradvisory.com

What this is

A start-to-finish guide to rebuilding your inbox inside an AI you work with, so that email and messages arrive in one place, already sorted and drafted, and the work in them gets done rather than described back to you. Ten steps in three arcs, with the decision behind each one and the traps to avoid.

It's illustrated throughout with the real system I run (four email accounts and a client's Teams channels, built in Claude Code by a non-developer), so every step shows both the principle and how it played out. Read it and build your own, or hand it to your AI and have it walk you through your version.

One thing to say up front, because it runs underneath every step. This system doesn't work because of a clever prompt. It works because the AI reading my email already knows what I'm working on: my task list, my projects, my pipeline, and how I write. Wherever a step went unusually fast, that context layer is the reason. I flag it as it comes up and come back to it at the end.

Who it’s for

People who run more than one inbox and are tired of managing it: consultants, fractional operators, founders, anyone whose email is mostly other people's work arriving. Non-technical, willing to work with AI step by step, and unwilling to hand their send button to a machine.

Before you start

Here's what you actually need, so you can tell whether you're ready or whether something else comes first.

An AI that runs on your machine, keeps files, and can connect to your email. This worked example was built in Claude Code Desktop, which is a normal desktop app with a chat interface. You describe what you want, it writes the files and runs what needs running. A plain chat window can't do this, because the system needs to read and write local files and call your email on your behalf. The only requirement is a paid Claude plan, which starts at twenty dollars a month.

Email connectors. Gmail through a Google Workspace connector; Outlook through a Microsoft Graph connector if you have one. Start with your single busiest account. Add the others after the loop works on one.

One folder to keep files in. Mine is a plain-markdown vault I also use for tasks and projects, and that's what makes the cross-linking possible, but the inbox loop works on its own.

No code skills. You'll be asking your AI to build each piece and pasting instructions from this playbook. I wrote none of the code in my system.

The part that decides how well this goes is context. The system in this worked example reads my task list, my project list, and my business development pipeline before it looks at a single message, which is

why a reply about a proposal arrives already connected to the pipeline entry it closes. That's an AI operating system, and building one is a separate skill from building an inbox. You can absolutely follow this playbook without one; you'll just get an inbox that's sorted rather than an inbox that knows what you're doing. The steps still work, and the operating system is the natural next build.

How to read it

Ten steps, in order, in three arcs: **Foundations** (1 to 3), **The loop** (4 to 6), and **Let it do the work** (7 to 10). Each step has the same shape:

- **Outcome:** what you have when the step is done.
- **Why it matters:** the principle underneath it.
- **In practice:** how it played out in my real build, with the specific decisions.
- **How you'd do it:** the reusable version, adapted to your situation.
- **The AI skill:** the capability you build, and where AI does the heavy lifting.
- **Watch for:** the real lessons and traps.

The ten steps

Arc 1. Foundations

1. Decide what the system is allowed to do
2. Teach it who you are
3. Put everything in one file

Arc 2. The loop

4. Triage: read everything, sort it, draft it
5. The review surface: one screen where you decide
6. The executor: only what you approved, and prove it

Arc 3. Let it do the work

7. Hand work to your AI, and make it a task
8. Close the task loop
9. Learn from every edit
10. Let it run, with a trust line

The principles that run through all ten steps

- **Nothing sends without approval on the specific email.** Ever. Approval in one place, by you, per message. This is what lets everything else be automated.
- **Approved means answered and gone.** A reply you approve is sent and the original leaves your inbox in the same action. Your inbox is a place things pass through, never a place they live.
- **One file holds everything.** Every triaged message is a card in one plain file with a status. Triage writes cards in, you decide on them, the executor carries them out. Because it's one file, everything is auditable, undoable before it runs, and yours.
- **Verify against the artifact, never the claim.** A run that says "sent" has to show a real message id. A run that says "filed" has to be checked against what's actually still in the inbox. A layer between you and the truth will report success for something it didn't do.
- **Every edit is a lesson.** The difference between the draft and what you actually sent is the richest signal you have about how you write. Capture it every time.
- **The judgment stays yours.** The system drafts, sorts, files, and does the mechanical work. Whether to say yes, what to commit to, and what you know about one relationship that shouldn't reach another: those are you.

ARC 1

Foundations

STEP 1

Decide what the system is allowed to do

OUTCOME

A short written list of what your inbox system may do on its own, what needs your click, and what it must never do.

WHY IT MATTERS

The decisions you make here are what let you automate everything else without fear. If the boundary is fuzzy, you'll either hover over every run or stop trusting it entirely. If it's written down, you can hand it more and more work.

IN PRACTICE

My list has three lines. Nothing sends, posts, or files without my approval on that specific item in the dashboard. An approved reply is my own message, sent under my name with my signature (the AI drafted it, I read and approved it, so it's mine). Anything the AI sends on its own initiative, without my per-message approval, has to identify itself as my agent, and this system therefore never does that. I also added one standing exception: receipts and invoices in my practice inbox are never archived, because another automation reads them there.

HOW YOU'D DO IT

Write your three lines. What can it do without asking (sort, draft, group, flag urgency, archive senders you've already ruled on)? What needs your click (send, file, snooze, hand off)? What must it never do (send unapproved, carry knowledge between relationships, touch anything another system depends on)? Then look at your inbox for the things you didn't think of: an automation reading receipts, a shared inbox someone else also works, a folder your accountant watches.

THE AI SKILL

None yet. This step is you, thinking. Your AI can interview you for the list if it helps, one question at a time.

> WATCH FOR
>
> Treating "approve" as a formality later. When the drafts get good, you'll be tempted to batch-approve without reading. The system is only safe because you read what goes out under your name.

STEP 2

Teach it who you are

OUTCOME

Two short files the system reads before it touches a single email: how you write, and how you file.

WHY IT MATTERS

A draft that doesn't sound like you costs more time than no draft, because you rewrite it and resent it. Filing rules are what turn a hundred newsletters a week into something that never needs your attention again.

IN PRACTICE

My voice file started as five bullets (complete sentences, contractions always, no em dashes, short beats long, one ask per email, "Best, Erica") and a note on register per account, since my nonprofit emails are brisk and my own-practice emails are warmer. My filing rules file started empty and grew from clicks: every time I marked a sender as noise in the dashboard, the executor wrote a permanent rule. Three weeks in it holds 81 rules I approved one click at a time.

HOW YOU'D DO IT

Ask your AI to interview you for a voice-email file, five to ten lines, in your own words. Then create an empty triage-rules file with two headings: senders that are always noise, and senders that must always surface even if they look like a newsletter. Don't try to fill it. Step 6 fills it from your real decisions.

THE AI SKILL

Interviewing you for rules rather than guessing them, and writing the file so that every later run reads it first.

> WATCH FOR
>
> The surface rules aren't enough. Sign-offs and no-em-dashes are necessary and nowhere near sufficient. What makes a draft sound like AI is cadence: balanced clauses, three-item lists where you'd use one, every sentence the same length. Step 9 is where that gets fixed, from your edits, so don't expect the first drafts to be you.

STEP 3

Put everything in one file

OUTCOME

A single queue file where every triaged message becomes a card with a status.

WHY IT MATTERS

This is the architecture, and it's the reason the whole thing is auditable. Triage writes cards in. You change their status. The executor reads the statuses and acts. Nothing happens that isn't visible in the file, and nothing you decided can be silently lost.

IN PRACTICE

Mine is a plain JSON file. Each card carries the account, the sender, a short summary, a one-line proposal of what to do and why, the full text of the email, the drafted reply if there is one, a suggested folder, and a status that starts at "proposed." Everything I do in the dashboard writes back into the same file: edits to the draft, the folder I chose, notes to the agent, snoozes, handoffs. The executor never reads anything else.

HOW YOU'D DO IT

Ask your AI to design the card shape with you. The fields you need are: which account, who it's from, a summary, the proposal, the full email text, the draft (as separate to, cc, subject, and body fields), a suggested folder, and a status. Have it write an empty queue file. Everything from here on reads and writes that one file.

THE AI SKILL

Designing a data structure in plain language and holding to it. Your AI is good at this if you ask it to explain the shape back to you before building.

> WATCH FOR
>
> Two things I got wrong. A draft stored as one string instead of separate fields rendered as a completely blank compose box, so the card looked broken and the work was lost. And two cards sharing the same id made every button act on the wrong one. Ask for unique ids and separate draft fields on day one.

ARC 2

The loop

STEP 4

Triage: read everything, sort it, draft it

OUTCOME

A saved instruction your AI runs on the phrase "triage my inbox" that reads every inbox, sorts each message into one of four buckets, drafts the replies, and writes the cards.

WHY IT MATTERS

This is where the work moves. Sorting is a decision you were making a hundred times a day by hand. Drafting is the part that used to be the whole job. Both now happen before you look.

IN PRACTICE

My triage reads the whole inbox of each account, not a time window, because the goal is inbox zero and every message still sitting there should end up in a bucket. Four buckets: reply needed (with a draft), action needed (with a ready task line), worth knowing, and noise. It groups related messages into one card, so five emails about one invoice arrive as one decision. It reads my task list, project list, and pipeline first, so a reply that closes something I'm already tracking says so. It writes a three or four sentence digest at the top: what's urgent by name, who's waiting on me, how big the rest is. And it's stingy with "urgent," because a system that flags everything flags nothing.

HOW YOU'D DO IT

Hand your AI this skeleton and ask it to adapt it to your accounts:

```
TRIAGE MODE, on "triage my inbox"
1. Read voice-email.md and triage-rules.md first. If I keep a task list
   or project list, read those too, so emails link to existing work
   instead of duplicating it.
2. Read the existing queue file BEFORE fetching mail. Never overwrite a
   card whose status isn't "proposed". An approved draft carrying my
   edits must survive every refresh.
3. Pull the FULL inbox of each account (cap 50 messages per account per
   run). Every message ends up in a bucket.
4. Sort every message into exactly one card:
   - reply: draft the response in my voice. Include the full email
     text, a one-line proposal of what to do and why, and a suggested
     folder for filing after it sends.
   - action: something I must do that isn't email. Write a ready task
     line with a marker tying it to this card, and a link to the
     thread. If YOU could do all or part of the work, say so on the
     card and say which part stays mine.
   - fyi: worth knowing, nothing to do. Group related messages.
   - noise: newsletters, notifications, marketing. One collapsed list.
     Senders I've already ruled on get archived without asking.
   - calendar invites: check my calendar for that slot, name any
     conflict, and say what sits before and after it.
5. Group ruthlessly. Five emails about one invoice is ONE card.
6. Write a 3-4 sentence digest at the top: what's urgent by name, who's
   waiting on me, how big the rest is. Be stingy with urgent.
7. Never invent facts, dates, availability, or commitments in a draft.
   Never carry what I know from one relationship into an email to
   someone else.
8. Write everything to the queue file (merge, never overwrite) and stop.
   Triage never sends anything.
```

Run it once. Read what it produced. Correct its judgment out loud ("this isn't noise," "these should have been one card") and have it write those corrections into the rules file. That first correction session is where the system starts becoming yours.

THE AI SKILL

Reading against context. The difference between an inbox that's sorted and an inbox that knows what you're doing is entirely whether the triage read your tasks and projects first.

> WATCH FOR
>
> Partial reads. My triage once drafted a reply promising to fix a bug, because it had read the first message of a thread and not the third, where the person said the fix had already worked. Make it read whole threads before it drafts.

STEP 5

The review surface: one screen where you decide

OUTCOME

A place where you see every card, edit the draft, and record a decision, with the decision written straight back into the queue file.

WHY IT MATTERS

This is the piece I was missing for four months. Drafts inside a chat conversation can't be edited in place, approved in one click, or found again tomorrow. The surface is what turns "AI helps me with email" into "I decide, and it does the rest."

IN PRACTICE

Mine is a local HTML page that reads and writes the queue file directly. Every email is a closed one-line row: sender, one-line gist, an urgent chip if it earned one, the date, and the primary action right on the row (Approve for drafts; add task, hand to Claude, or done for actions). Expanding a row shows the full card: the editable to, cc, subject, and body, the full email, a folder picker filled with that account's real folders, snooze options, a note box where I can type "I already replied from my phone, just file it," and the hand-to-Claude button. Four tiles across the top tell me what's pending my decision, what's queued for the agent's next run, what's in flight, and what's processed.

HOW YOU'D DO IT

Two options, and the first one works on day one. Simple: have your AI render the queue as a markdown review sheet you read in any editor, with your decisions typed next to each card (approve / file to X / skip / snooze until date / note: …), and have it parse your decisions back into the queue. Full: ask your AI to build the HTML page, serve the queue through a tiny local file server, and iterate. Mine went through four rounds of "make this faster to use" in one day. Whichever you pick, three things are non-negotiable: you can edit the draft before approving, the original draft is kept next to your edit (that's the fuel for Step 9), and every decision has an undo until it executes.

THE AI SKILL

Building a working interface from a description, and improving it from your reactions. You don't need to know how it works. You need to say "still too much scrolling" and let it fix that.

> WATCH FOR
>
> The local file server. If you build the full version, lock it so only your own page can talk to it. Ask your AI to explain the CORS risk in plain language; mine had a real hole that a reader caught.

STEP 6

The executor: only what you approved, and prove it

OUTCOME

A second saved instruction that reads your decisions and carries them out: sends, files, archives, snoozes, RSVPs. Nothing else.

WHY IT MATTERS

This is the boundary from Step 1 made real. The executor is the only thing that touches the outside world, and it can only do what the queue says you approved.

IN PRACTICE

My executor runs every hour on weekdays and exits immediately if nothing is pending. For each approved reply it sends from the matching account, in the thread, as HTML with my real signature, using my edited version wherever it differs from the draft. It checks the tool result for a real message id before it marks anything sent. Then it files the original out of my inbox to the folder I chose. It archives the senders I marked as noise and writes a permanent rule for each. It moves snoozed messages out and brings them back on the day. It responds to calendar invites and reads the attendee status back to confirm. Then it re-reads the queue, counts what's still pending, and writes one line to a run ledger.

HOW YOU'D DO IT

Hand your AI this skeleton:

```
SEND MODE, on "send my inbox"
1. Read the queue file fresh. Collect ONLY items I approved.
2. Send each approved reply from the matching account, in-thread, using
   my edited version wherever it differs from the draft. Verify the
   send actually happened (a real message id) before marking it sent.
   Never mark anything done on an unverified claim.
3. After a verified send, file the original thread to the chosen
   folder so it leaves my inbox. Approved means answered AND gone.
4. Execute the other decisions: file-no-reply items, noise sign-offs
   (archive + write a permanent rule so that sender never asks again),
   snoozes (archive now, return to the inbox on the due date), RSVPs
   (respond on the calendar, verify my status changed).
5. Mark each item done only after ITS OWN call returned. Never stamp a
   batch as done at the end of a loop.
6. Self-audit before finishing: re-read the queue and count what's
   still pending. If anything remains, say exactly what and why. A
   clean report with work left behind is the one unforgivable failure.
7. Append one line to a run ledger: what sent, what filed, what failed.
```

THE AI SKILL

Verification. Not "the tool said OK" but "I read the result and it contains a message id," and later, "I read the inbox again and the message is gone."

WATCH FOR

The false clean report. A week into mine, a run reported all its filing complete while seven of the eight emails were still sitting in my inbox: it had stamped every card as done at the end of a loop instead of one at a time as each call came back. Now every triage run audits the queue's own claims against what's really still in each inbox, fixes anything that lied, and reports the count out loud. Build that check before you need it.

ARC 3

Let it do the work

STEP 7

Hand work to your AI, and make it a task

OUTCOME

Cards that tell you how much of the work the AI can take on, a button that hands it over, and a task on your real task list that the agent owns until it reports back.

WHY IT MATTERS

This is the step that changes the system from "helps me manage" to "does the work." And it only works if a handoff is visible: queued, running, finished, with the result written where you asked for it.

IN PRACTICE

At triage, any card where the AI could do all or part of the work carries a short assessment: "I can do all of this," or "I can do most of it, and here's the piece that's still yours." Granting someone access to a shared folder. Reading a four page government letter and finding the one line that needs a response. When I click hand-to-Claude, three things happen: a row is written to my task list owned by the agent, the executor picks it up on its next run and does the work, and a report is written back onto the card saying what was done, where the output is, and what's left for me. If nothing is left for me, the card closes itself and the email files. There's also a button that opens a session on that piece of work, so I can look at what it did and carry on from there.

HOW YOU'D DO IT

Add two things to your triage instruction: "on any card where you could do all or part of the work, say which part, and never claim more than you can verifiably do," and "when I hand a card over, write a task line to my task list owned by the agent, with a marker tying it to this card." Then add a handoffs section to your executor: "for every handed-over card, do the work described, write a two-sentence report onto the card, and close the card only if nothing is left for me." Reversible work proceeds; anything outward-facing beyond what you already approved gets drafted, not sent.

THE AI SKILL

Honest scoping. The card has to say "the decision itself stays yours" when that's true, rather than quietly doing something you'd have wanted to see first.

> WATCH FOR
>
> The black hole. For the first two weeks of mine, anything I handed over disappeared, and I'd have to remember to chase it. The fix was to stop treating a handoff as separate from my tasks. If your task list and your inbox are two different places, work can hide in the gap between them.

STEP 8

Close the task loop

OUTCOME

An email you turn into a task files itself out of your inbox, stays visible in a tracked view, and closes itself the moment the task is done.

WHY IT MATTERS

Without this, your inbox becomes a second, worse to-do list within a week, because every "I'll deal with that later" email stays sitting there as a reminder.

IN PRACTICE

When I click add-to-tasks on a card, the task line carries a marker tying it to the card and a link back to the thread, and the email files to its folder in the same click. The card moves to a tracked view. Every executor run checks the task list for the marker; when the task is ticked done, the card reads "task done, closed" and disappears. The email was never sitting in my inbox pretending to be a reminder.

HOW YOU'D DO IT

Put a marker in every task line the system writes (mine is "source: inbox#" plus the card id). Add one step to the executor: "for every card with a task, look for its marker in the task list; if the task is ticked done, mark the card done and file the email if it was kept." That's the whole loop.

THE AI SKILL

Cross-referencing two files reliably. Your AI can do this easily as long as the marker is unambiguous.

> WATCH FOR
>
> A deleted or reworded task. If the marker vanishes, the card can never close. Have the executor report "no marker found" on the card rather than guessing.

STEP 9

Learn from every edit

OUTCOME

A voice file that grows from your real corrections, on real emails, so the drafts get closer to you every week.

WHY IT MATTERS

This is where the compounding happens. A system that learns your voice from a style guide plateaus. A system that learns from the difference between what it drafted and what you actually sent keeps improving for as long as you use it.

IN PRACTICE

After every send, the executor compares my final version against the draft, line by line. It names each changed phrase, infers the rule and the reason behind the change, and writes it as a dated observation. A rule seen three times gets promoted to the top of the file, where every future draft reads it first. Most of what it has learned is about judgment rather than grammar. My drafts kept handing decisions back to other people when the decision was already mine. They promised to check my calendar and confirm later instead of naming a time. One draft repeated something to a contact that I only knew from a completely different conversation; I cut it, and the rule now says what I know in one relationship stays out of another. Once a week it also pulls a few emails I wrote entirely by hand and tests its rules against those, so the profile stays honest against drift.

HOW YOU'D DO IT

Add to the executor: "for every sent item where my version differs from the draft, diff them line by line, name what changed and why, and append a dated observation to voice-email.md. Three similar observations become a rule at the top. Zero-diff sends get one line saying the register worked." Then read the file yourself every couple of weeks and cut anything that isn't true.

THE AI SKILL

Inferring a rule from a diff. "She shortened it" is a weak finding; "she turned a promise to check later into a concrete time because she does things in the email rather than after it" is the kind that changes the next draft.

> WATCH FOR
>
> The rule you forgot to write. The discretion rule above (what I know in one relationship doesn't reach another) only exists because a draft got it wrong once. Read the first ten drafts with that specific question in mind.

STEP 10

Let it run, with a trust line

OUTCOME

Triage and the executor on a schedule, a run ledger, and one line on your review surface that shows the last run's claim next to an independent count of what's still pending.

WHY IT MATTERS

An inbox system that only runs when you remember to run it is a slightly better chat window. The value arrives when it runs before you're awake and again at lunch, and you can see at a glance whether the last run did what it said.

IN PRACTICE

Triage runs inside my morning planning routine and again at midday. The executor runs hourly on weekdays and exits in seconds if nothing is pending; it also waits ten minutes after any change to the queue, so it never fires while I'm mid-review. Every run, including the empty ones, appends one line to a ledger, because a routine that writes nothing is indistinguishable from one that never fired. The dashboard shows the last ledger line next to its own count of what's queued, with a warning if the two disagree.

HOW YOU'D DO IT

Schedule triage for the morning and midday, the executor every hour or two on workdays. Have every run write one ledger line. Then add the trust line to your review surface: the last run's summary, and beside it a count computed from the queue file itself. When those two numbers disagree, believe the count.

THE AI SKILL

Building a routine that reports on itself and can be judged by the age of its last ledger line rather than by a scheduler that rolls forward silently after a miss.

> WATCH FOR
>
> Any tool that lists things for you can drop some silently. The mentions feed for my client's Teams channels returned two of the four messages that named me one day, with no error, and the one it dropped was the most important message of the day. Now a second, cheaper pass checks the active channels independently, and anything it recovers is marked as recovered so the miss rate stays visible. Whatever feed your system reads, ask what it would look like if that feed quietly returned half.

How to use this playbook

Two ways, and they're the whole point.

Read it and build it yourself. The ten steps are in order deliberately. Steps 1 to 3 are thinking and setup, and you can do them today. Steps 4 to 6 give you a working loop on one account in an afternoon. Steps 7 to 10 are what turn it from a sorted inbox into a system that does the work.

Or hand it to your AI and have it guide you. Give this whole document to a capable assistant and say: "Walk me through building my own version of this. Interview me about my accounts and my rules first, one question at a time, then take me through it step by step." A good assistant uses the playbook as the method and adapts every step to you. That's the design: the playbook is the thinking, your AI is the hands.

Why it is a document, not a set of skills

You could imagine shipping this as installable skill files. I've chosen a plain document, because your inbox is different from mine in every way that matters: which accounts, what counts as noise, how you sign off, what another system depends on. That's high-judgment, high-variance work, and a capable assistant with a good guide does better at it than a rigid script. A document is also portable: anyone can read it or hand it to their AI without installing anything.

What made this actually work

If you take one thing from this playbook, take this.

Every step above is real, and you can follow all ten of them. But the reason my inbox does what it does is that the AI reading it already knew what I was working on. It had my task list, so an email about a deadline linked to the task instead of creating a second one. It had my project files, so a reply about a proposal arrived already connected to the pipeline entry it closed. It had my voice rules, the specific ones, including the words I never use. It had the decisions I'd already made, so it never drafted something I'd already said no to.

That's the difference between an inbox that's sorted and an inbox that knows what you're doing. The first is a better filter. The second is a colleague.

I built the first version of this in April, before most of that context existed, and it helped me manage my email. Same tools. Same person. What changed in August was the surface I gave it and the context underneath, and now it does a real share of the work.

Learn to build the operating system

That system is what I teach.

TEMPO is my six-week cohort for non-technical people who want to build their own AI operating system: the context files, the daily routines, the knowledge base, the loops that hold AI to your standards, and the automations that run without you. You build it during the six weeks, on your own real work, and you leave owning it.

Everything in this playbook, the voice rules, the verified executor, the task loop, the handoffs, came out of that system. Build the system, and an inbox that does the work is one week of what it can do.

layeradvisory.com/tempo

/erica-layer

AI ON PURPOSE

Build the operating system underneath.

The website was the easy part. What made it easy was an AI that already knew my positioning, my voice, and my real work, because I had built it a system to know them from.

TEMPO is my six-week cohort for non-technical people who want to build exactly that: the context files, the daily routines, the knowledge base, the loops that hold AI to your standards, and the automations that run without you. You build it on your own real work, and you leave owning it.

Everything in this playbook came out of that system.

layeradvisory.com/tempo

I'm Erica Layer, an AI-native fractional COO. I help purpose-driven organizations build the human and AI systems their work runs on, and I teach people to build their own.
layeradvisory.com · aionpurpose.substack.com