EE SOLUTIONS

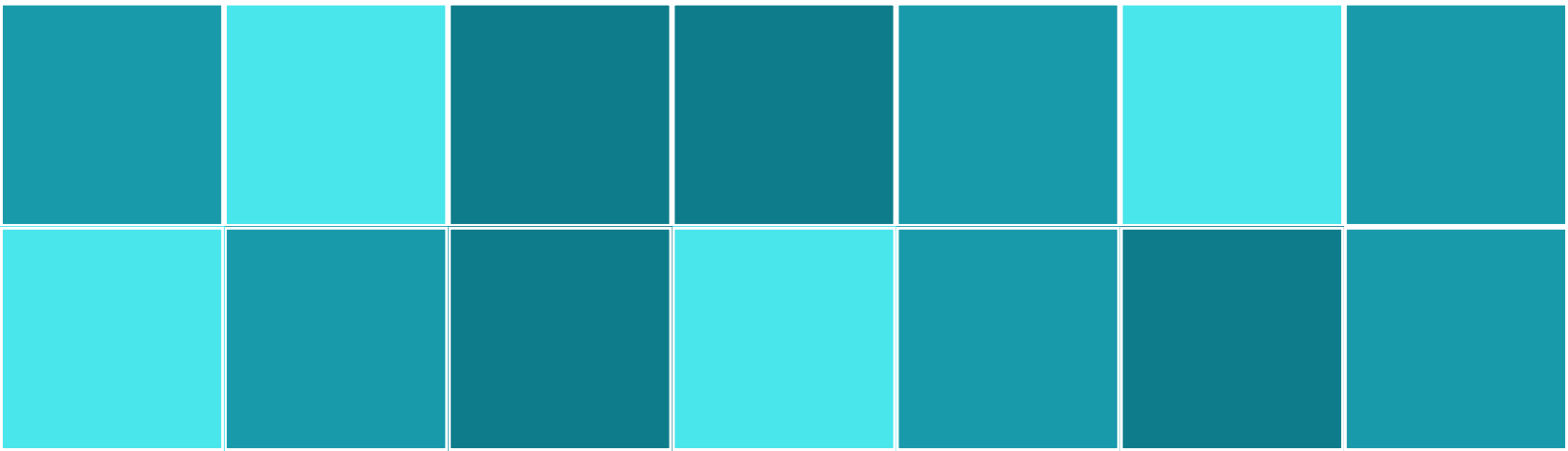
WHITEPAPER

The Unit of Work

How enterprise AI earns its way into production

August 2026

EESOLUTIONS.AI



EE SOLUTIONS

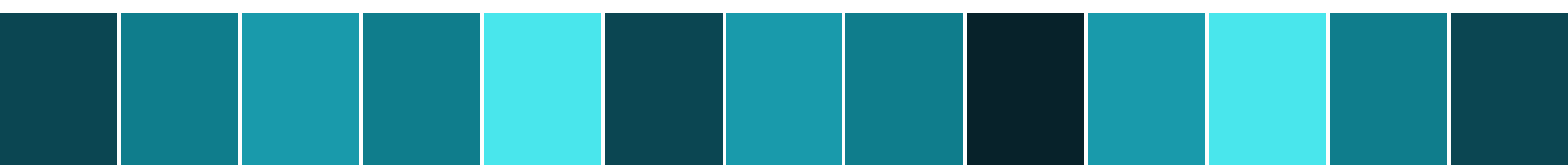
The Unit of Work

How enterprise AI earns its way into production

Most enterprise AI programs don't fail because the model wasn't good enough. They fail because nobody defined the work precisely enough to tell whether the model was good enough.

This paper is a practical method for choosing, bounding, pricing, and proving enterprise AI work. It's written for the executive who has to approve the budget and the operator who has to live with the result. It covers what to build first, how to know it's working, where the human belongs, and what it costs to run once the pilot is over.

AUGUST 2026



DIAGNOSIS

The stall isn't technical

Enterprise AI has stopped being an experiment. Budgets are committed, teams are staffed, and pilots are running in nearly every large operating company. What hasn't changed is the conversion rate from pilot to production, which most programs still describe as low.

When we're brought into a stalled program, the cause is almost never the model. Four patterns account for most of it.

The work was never named. The project was defined at the level of a department or a document type rather than a repeatable task with an input, an output, and a person who does it today. "Use AI in claims" isn't a project. "Assemble the evidence packet for a denied claim from the chart, the policy, and the denial letter" is a project.

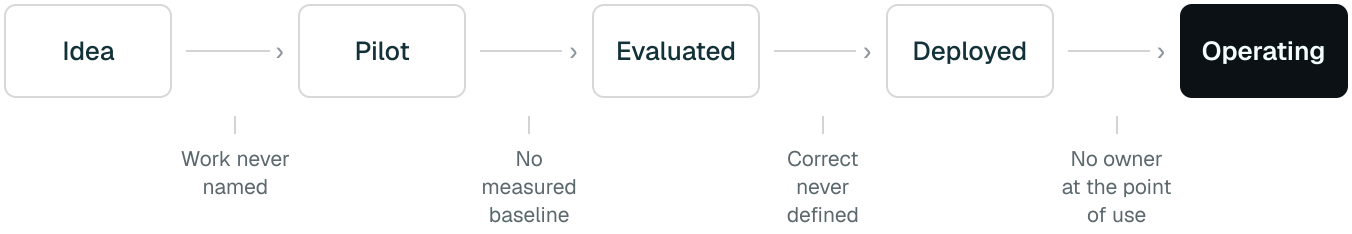
The baseline was never measured. The team can't say what the manual process costs today, how long it takes, or how often it's wrong. Without that, no result can be called an improvement, and every review meeting becomes an argument about impressions.

Correct was never defined. The system produces plausible output. Nobody agreed in advance what makes a given output right, so acceptance becomes a matter of taste and the project can't exit review.

Nobody owned it at the point of use. The build team owned the model. No one owned the workflow the model sits inside, the exceptions it throws, or the retraining it needs when the underlying process changes. The pilot works and then quietly stops being used.

Each of these is a decision, not a capability. All four can be made before a line of code is written, and making them well is most of what separates a system that runs from a demo that impressed a steering committee.

Figure 1. Four decision failures, each at a specific handoff. None of them are model problems.



QUALIFICATION

The Unit of Work test

A unit of work is the smallest repeatable task that produces something a person currently produces, that someone downstream consumes, and that can be counted. Before scoping anything else, put the candidate through four questions. These are pass or fail. A candidate that fails any one of them isn't ready, and no amount of model selection will change that.

1. Is it observable?

Can you watch a person do it today, end to end, and write down what goes in and what comes out? If the workflow only exists in a slide describing a future state, there is nothing to automate and nothing to measure against.

2. Is it repeatable?

Does it happen often enough, in a similar enough shape, that the same instructions apply across instances? Volume creates both the return and the evidence. A task performed eleven times a year can't produce an evaluation set and can't pay back a build.

3. Is it judgeable?

Given an output, can a qualified person say whether it's right, and would two qualified people agree? If correctness is contested inside your own team, you can't grade a model on it and you can't ship it.

4. Is it owned?

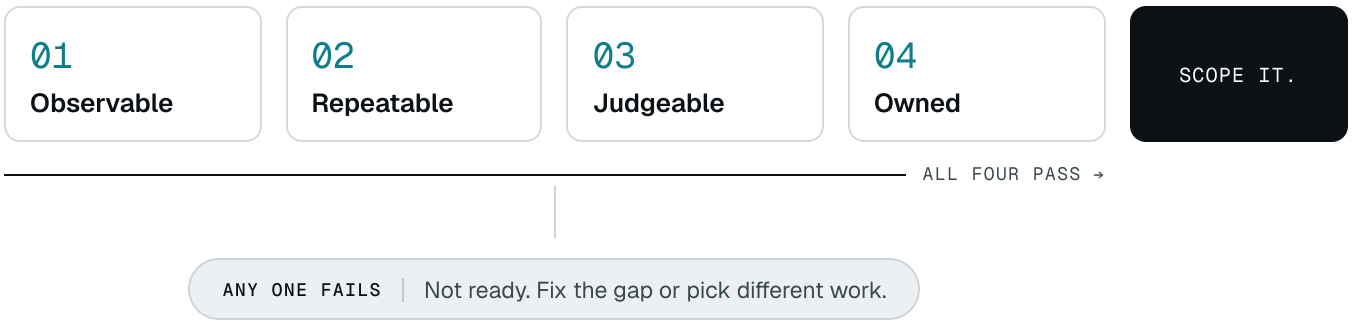
Is there a named person whose job gets easier or harder depending on whether this works, who will accept exceptions, and who will say when it's good enough? Sponsorship at the executive level isn't ownership.

Two additional filters apply to the work itself rather than the organization around it.

Match to what the technology actually does well. Current systems are strong at reading large volumes of documentation, converting unstructured input into structured output, following detailed written procedure, drafting to a template, and writing routine code. They remain weak at arithmetic precision, recognizing the limits of their own knowledge, holding context across long horizons, and resolving genuine ambiguity. Route arithmetic to arithmetic. Route ambiguity to a person.

Confirm the money is real. Small time savings across a large volume is a real return. Large time savings across a small volume usually is not. Multiply before you commit.

Figure 2. Four pass-or-fail questions. Failing one is a reason to stop, not a reason to proceed carefully.



QUALIFICATION

Qualification in practice

Three examples run through the rest of this paper. Each is a real shape of work we see repeatedly across manufacturing, financial services, and industrial operations. Details are composited and anonymized.

- **Supplier quality.** A manufacturer receives supplier defect reports in inconsistent formats across dozens of vendors. A quality engineer reads each one, pulls the relevant history for that supplier and part, and drafts a corrective action request against a standard template.
- **Regulatory change.** A regulated lender monitors publications from several supervisory bodies. An analyst reads each release, determines whether it touches the institution, maps it to affected internal policies, and drafts an impact memo for the compliance committee.
- **Field service.** An equipment manufacturer receives technician field notes and machine error logs. A service planner reads both, identifies the likely failure, and produces a parts requisition and repair plan before dispatch.

Two candidates that come up constantly and don't survive the gate are included for contrast.

Figure 3. Two of the most commonly proposed enterprise AI projects fail the gate for different reasons. One has no measurable work behind it. The other has measurable work that language models are the wrong instrument for.

	Supplier quality	Regulatory change	Field service	Enterprise assistant	Demand forecasting
Description	Draft corrective action requests from inbound supplier defect reports	Turn regulator publications into mapped policy impact memos	Turn technician notes and error logs into parts requisitions and repair plans	"An assistant that answers any employee question about anything"	"Predict demand more accurately using AI"
Observable?	Yes. Engineer does this today against a fixed template	Yes. Analyst does this today, output goes to committee	Yes. Planner does this today before dispatch	No. No current process, no defined input, no defined output	Yes, but the current process is statistical, not textual
Repeatable?	Yes. Hundreds monthly, consistent output shape	Yes. Weekly cadence, consistent memo structure	Yes. Thousands monthly across the installed base	Unknown. Question distribution undefined	Yes
Judgeable?	Yes. Quality lead accepts or rejects against the template	Yes, with care. Correct policy mapping is checkable, tone isn't	Yes. Requisition is right or the wrong part ships	No. No agreed standard for a right answer	Yes, but by forecast error, not by language quality
Owned?	Yes. Supplier quality manager	Yes. Head of compliance	Yes. Service operations lead	No. IT sponsors it, no operator depends on it	Yes. Demand planning lead
Verdict	Qualified	Qualified	Qualified	Disqualified. Nothing to measure against	Wrong tool. Use forecasting methods and keep language models out of the arithmetic

ECONOMICS

Price the unit, not the project

The standard AI business case compares a build cost against a headcount saving and produces a number nobody believes. A better instrument is cost per completed unit: everything it takes to get one output to the point where a downstream consumer accepts it.

Cost per completed unit has four components.

Model and compute. Tokens, embeddings, retrieval, hosting. Usually the smallest line, and usually the only one anyone estimates.

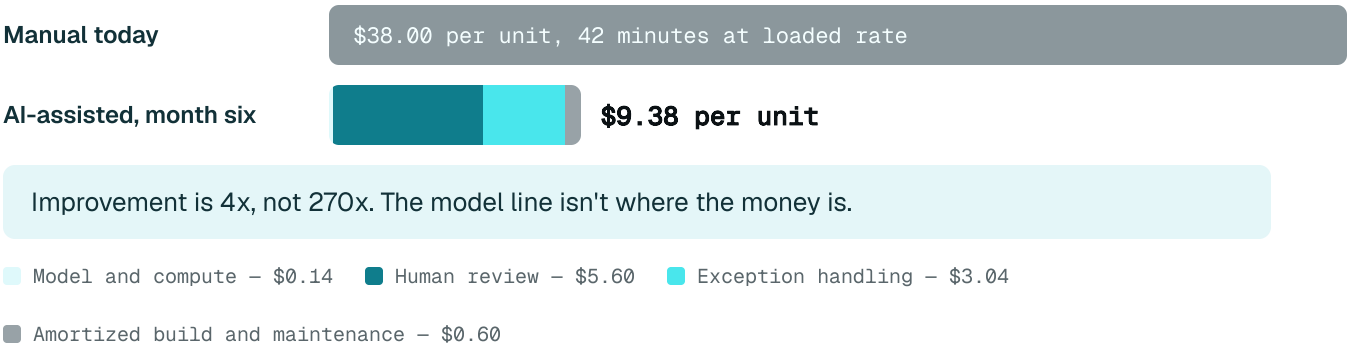
Human review. Minutes spent per unit checking, correcting, and approving output, at loaded rate. This is normally the largest line in the first year and it's routinely left out entirely, which is how programs report savings they can't find in the budget.

Exception handling. The share of units the system can't complete, priced at the full manual cost plus the time already spent discovering the failure. Exceptions are more expensive than the manual baseline, not equal to it.

Amortized build and maintenance. Development spread across expected volume, plus the ongoing cost of evaluation, model changes, and prompt and retrieval maintenance. Assume this line never reaches zero.

First, a system at ninety percent quality that requires a five minute review per unit can cost more per completed unit than one at eighty percent quality that requires a two minute review, because review time doesn't fall linearly with quality. Design for reviewability, not only for accuracy. Second, the honest comparison is manual cost per unit against total AI cost per unit including review and exceptions, over a stated payback period. An eight month payback survives an executive team's patience. A four year payback doesn't survive a budget cycle.

Figure 4. The cheapest line in an AI system is the model. Estimating only that line is how programs miss their business case.



Illustrative figures for a document-drafting workload at roughly 2,000 units per month.

SCOPE

Cut it down before you build it

The most reliable predictor of a project reaching production is how small the first version was. Take the qualified unit of work and decompose it into subtasks. Then select the first build target using three criteria.

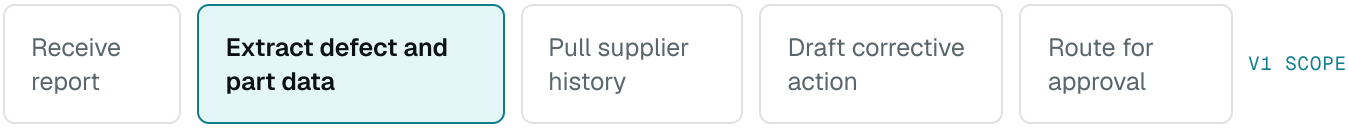
Highest volume. Concentration beats coverage. In most workflows a minority of input types account for a majority of instances. Build for that majority first and route the rest to the existing manual path.

Lowest variance. Prefer subtasks where instances resemble each other. Variance is what turns a working prototype into an unbounded engineering effort.

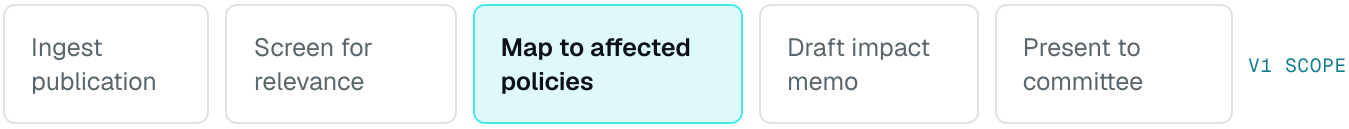
Clearest right answer. Prefer subtasks where correctness is checkable against something concrete. Save the subjective portions for later, when you have trust and a working evaluation harness.

Then write down the exclusions explicitly. The scope document should name the input types, edge cases, and downstream steps that are deliberately out of the first release, and say what happens to them instead. Unwritten exclusions get re-litigated in every review.

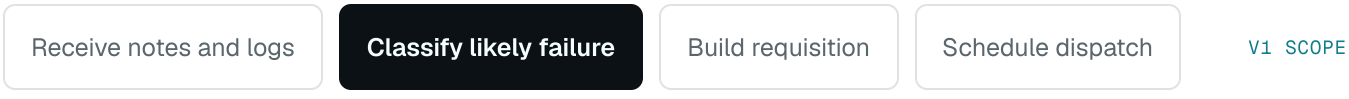
Figure 5. Three workflows, three first releases. Each takes one subtask, not one department.



Highest volume step, most consistent input shape, extraction is checkable against the source document.



Mapping is checkable against a fixed policy register. Memo tone isn't, so it stays manual in v1.



One classification drives every downstream step. Getting it right removes the most rework.

EVALUATION

Define correct before you build

The single most common reason a working system doesn't ship is that acceptance was never specified. The fix is to write the grading rubric before development starts, ideally in the same week the scope is set. A rubric has three parts.

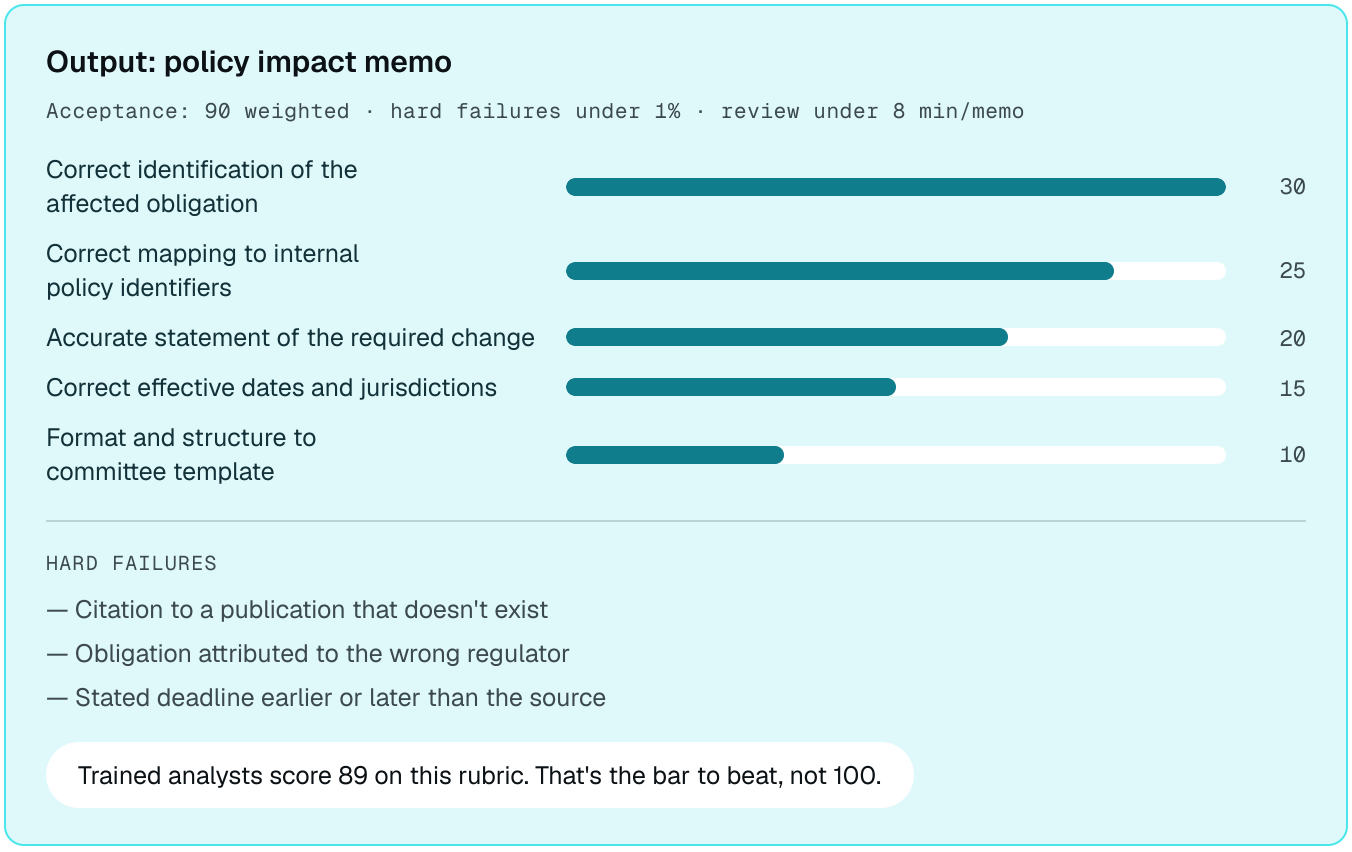
Weighted components. Break the output into elements that can independently be right or wrong, weighted by consequence rather than effort.

Hard failures. Errors that carry no partial credit and fail the entire output regardless of the rest of the score, tracked separately with their own threshold near zero.

Acceptance thresholds. The weighted score required to ship, the hard failure ceiling, and the review ratio the business will fund — agreed before the build.

Alongside the rubric you need a ground truth set covering the range of real inputs, including the awkward ones. Measure the human baseline against the same rubric: a system that scores 92 against a rubric on which trained staff score 89 is a different conversation than a system that scores 92 against an unmeasured assumption of perfection.

Figure 6. Weights reflect consequence, not effort. Hard failures are tracked separately and aren't offset by a strong score elsewhere.



AUTONOMY

Where the human sits

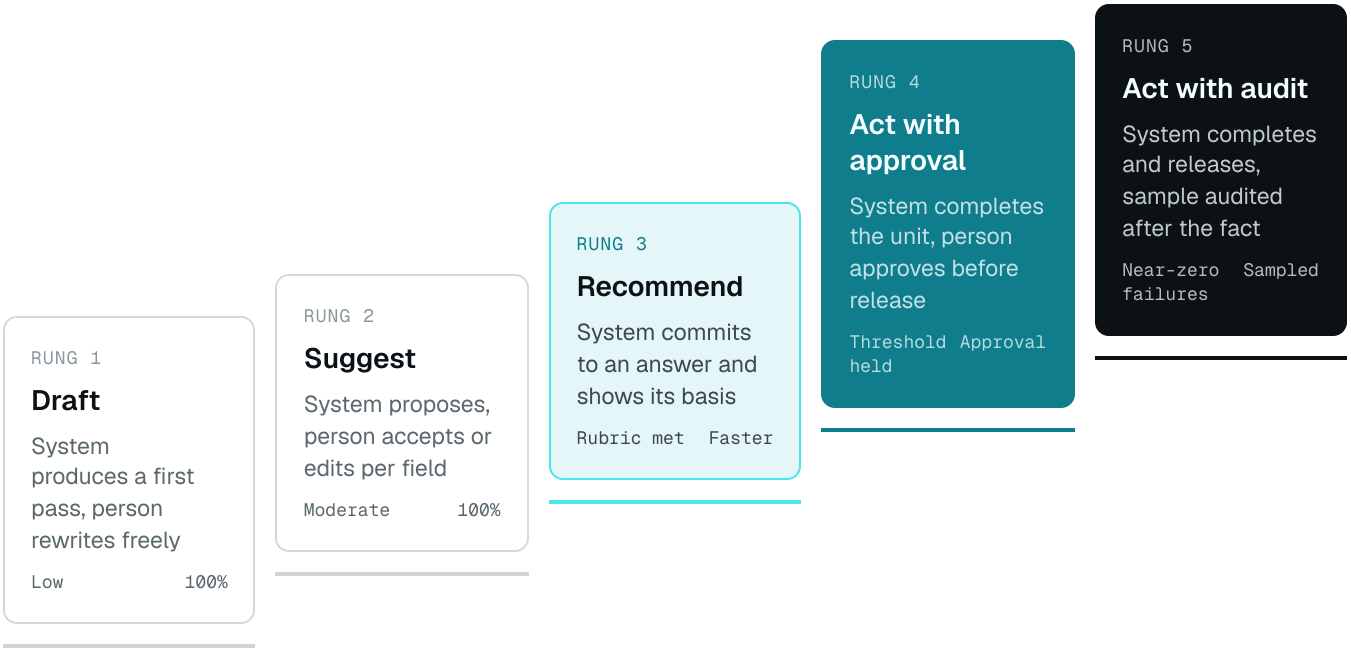
Human involvement is a design decision with an economic consequence, not a temporary compromise on the way to full automation. It should be set deliberately and revisited on evidence. Think in rungs. Each rung buys speed and cost at the price of a higher quality bar and more operational scaffolding.

Move on measured evidence, not elapsed time. A system earns the next rung by holding its rubric score and hard failure rate across a stated volume of real production units, not by having been live for a quarter.

Design the exception path before the happy path. Systems that escalate cleanly and visibly get adopted. Systems that fail silently get abandoned, usually without anyone formally deciding to abandon them.

Partial quality is often enough to change the economics. Moving a specialist from author to editor captures a large share of the available value even at moderate accuracy, provided the output is structured so that checking it is faster than writing it. Reviewability is a build requirement, not a nicety.

Figure 7. Autonomy is earned per workload, on measured evidence, not granted per program.



Most enterprise workloads should be designed to operate durably at rungs 3 and 4. Rung 5 is appropriate where hard failures are cheap to reverse.

DEPLOYMENT

Deployment posture, and who runs it on day 91

Two decisions made early constrain everything downstream: where the system runs and who owns it once it's live. **Where it runs** is usually already decided by existing policy — find out what yours permits before you design an architecture it forbids. **Who owns it** needs three names on launch day: an operator who runs the workflow, an engineer who maintains prompts and integrations, and a reviewer who samples output against the rubric. These can be part-time roles. They can't be unassigned ones.

Figure 8. Posture is usually decided by existing policy. The operating cadence is decided by nobody unless you decide it.

Where it runs

Managed API — DEFAULT FOR MOST

Frontier models operated by the provider, no infrastructure to run

LEAVES YOUR PERIMETER UNDER CONTRACT

Your cloud environment

Frontier models inside your own cloud tenancy

STAYS IN YOUR TENANCY

Self-hosted open weight — NARROW FIT

You run the model and the infrastructure

NEVER LEAVES YOUR CONTROL

First 90 days after launch

WEEK 1

Baseline live rubric scores against pre-launch evaluations. Expect a gap.

WEEK 4

First exception review. Reclassify recurring exceptions as scope, not failure.

WEEK 8

Review the review. Measure minutes per unit actually spent, not estimated.

WEEK 12

Rung decision. Promote, hold, or narrow scope on evidence.

READINESS

Readiness

If you can answer these in writing, you're ready to fund the build. If you can't, the gaps are your scoping agenda.

Figure 9. Fifteen questions. Unanswerable questions are the scope of work.

01 — THE WORK

- [ ] Can you describe the unit of work as an input, an output, and a person who produces it today?
- [ ] How many times does it occur per month?
- [ ] Which subtask is the first release, and what is explicitly excluded?

02 — THE ECONOMICS

- [ ] What does one completed unit cost today, in minutes and money?
- [ ] What is the projected cost per completed unit including review and exceptions?
- [ ] What is the payback period, and does it fit your budget cycle?

03 — THE EVIDENCE

- [ ] What is the grading rubric, and who signed it?
- [ ] What are the hard failures, and what is the ceiling on them?
- [ ] Do you have ground truth covering the awkward inputs, not only the clean ones?
- [ ] What do trained staff score on the same rubric?

04 — THE OPERATING MODEL

- [ ] Which rung does the system launch at, and what evidence promotes it?
- [ ] What happens to an exception, and who receives it?
- [ ] Who is the named operator, engineer, and reviewer on day 91?
- [ ] Does your policy permit the architecture you have designed?

Getting to production

The barrier to enterprise AI is lower than the failure rate suggests, but it sits in a different place than most programs look for it. It isn't model quality. It's the discipline to name the work precisely, price it honestly, define correctness before building, put the human somewhere deliberate, and assign ownership that survives the launch.

Start narrow. Measure the thing you're replacing. Write down what right looks like and get it signed. Then build the smallest version that produces a unit someone downstream will accept, and expand from evidence.

About EE Solutions

EE Solutions builds enterprise AI systems that reach production. We work across strategy and roadmapping, agent and workflow automation, and custom AI product development, with clients in healthcare, financial services, manufacturing, and consumer goods. Our practice is built around practical scoping, evaluation and reliability, close collaboration with the teams who own the work, and deployment that survives the handoff.