Working With AI Without Fooling Yourself

How to know the work is right when you can't judge it yourself

Companion essay to *A New Era of Intelligence Work*, draft v0.18.0 (e2124cc). Numbers in brackets point to the Sources list at the end.

1. Feeling good about the work proves nothing
2. Four things to do instead
3. Checking the work does not teach you anything
4. If you are the expert
5. If you run a team
6. A thirty-minute test on yourself
7. What to do this week
8. How much to trust all this
9. Sources

1. Feeling good about the work proves nothing

You just shipped something you could not have built a year ago. A database migration. A contract clause. A financial model. The AI wrote most of it. You read it over, it looked right, and it works. Or at least nothing has broken yet.

How do you actually know it is correct?

Usually the answer is a feeling. It looked right. It was fast. It was easy. All three lie.

- **It looked right.** Models are trained to sound good. They sound exactly as good when they are wrong as when they are right.
- **It was fast.** What felt fast was getting a finished-looking thing back in ten seconds. The cost shows up later, spread out, attached to no moment you remember.
- **It was easy.** Easy means the model was in familiar territory. That says nothing about whether *your problem* is.

The tool does not invent these feelings. It bends them, always toward overconfidence.

What happened when people measured it

Every time researchers checked the feeling against a stopwatch or a score, the feeling was too generous. A number in brackets like [1] points to the Sources list at the end.

Developers.[1] 16 experienced developers, 246 real tasks, code they knew well. With early-2025 AI tools they took **19% longer**. They believed they were **20% faster**.

Do not over-read the slowdown. That is a fact about early-2025 tools and nothing more. The same lab ran it again in 2026,[2] found weak evidence of a speedup, and said its own data was too messy to trust. But neither round found a group whose sense of its own speed matched reality. That is the part that matters here.

Doctors.[3] Physicians with 20 hours of training in critically evaluating AI output were handed plausible but flawed recommendations. Diagnostic accuracy dropped **14 percentage points**.

Students.[4] About 1,000 students improved **48%** on practice problems with a chatbot. The chatbot was taken away for the real test. They scored **17% worse** than students who never used it, and were *more* confident than the group that beat them.

Different people, tools and stakes. Same direction every time.

There is also a reason "it looked right" is the least trustworthy of the three. Assistant models are tuned on human ratings: people compare answers, and the model learns to give the kind they pick. Raters cannot always tell a correct answer from a confident, agreeable one, so the model learns to please the rater as well as to be right. When researchers studied real rating data, answers that agreed with the person's own view were more likely to be picked, and both people and automated raters sometimes preferred a convincing wrong answer to a correct one.[5] A robot hand trained the same way to grasp objects learned to hold itself between the camera and the object, so it only looked like it was grasping.[6] **Your reading of the output is not a neutral check. It is what the model was trained to satisfy.**

The rule

> **Your own sense of how the work is going does not count as evidence. It might be right. You have no way of telling when.**

Courts throw out tainted testimony, not because it is always false, but because there is no way to sort the true parts from the false parts. Your gut about AI-assisted work is that kind of witness. The developers were experts on their own code, so expertise does not fix it. The doctors had been trained on this exact bias, so knowing about it does not fix it either.

You cannot fix the feeling. You can only change what you build around it.

2. Four things to do instead

You can still work outside your expertise. You just cannot run on instinct while you do it.

Move 1: Go deep in one thing, once

Bad idea: knowledge can be rented now, so none needs to be owned.

Reality: the people who get the most from these tools already know one field deeply. That gives you transferable structure. How state behaves. How systems fail. What "correct" means when nobody is checking. Someone with no anchor ships working output and learns nothing, because they cannot tell which parts were right and which got lucky.

The anchor buys you the ability to judge finished work in your field and near it. It does not buy a trustworthy feeling about the process. The developers above were experts on their own code.

Do you have one? You will answer too generously, so use evidence:

- Can you explain why something in your field fails, two levels deep, without looking it up?
- Have you debugged something where the obvious cause was wrong?
- Can you sometimes predict what will break before it breaks?
- Have you maintained something long enough to regret how you built it?

If those come back thin, spend the next year going deep in one field. The tool is genuinely good for that, in learning mode (Move 4).

Move 2: Sort work by what it costs to be wrong

Bad idea: check harder when something feels risky. Your worry is the unreliable signal again, and judging whether an answer is correct needs the expertise you do not have.

What you can always judge honestly is the consequences.

Bucket	What it looks like	What to do
Cheap to be wrong	Fails loudly, easy to undo	Move fast, fix on discovery. Most work lives here.
Expensive to be wrong	Fails silently, hard to undo, others depend on it	Build the check *before* the work. Do not start until it exists.
Cannot be undone	Money, safety, legal effect, large blast radius	Not doable on rented knowledge. Needs a real expert at the decision, not reviewing at the end.

One rule keeps it honest: **if you cannot pick a bucket, that is your answer, and it goes in the higher one.** A system where "not sure" means "probably fine" is permission to skip the

check.

Move 3: Make something other than yourself decide

Bad idea: verifying means reading the work carefully. It does not. Verification is a property of the system around the work. If you cannot judge the work, something else has to.

In software: write the test **before** the code, so passing means meeting a standard you set. Demand sources you can retrieve, then open them. Run risky operations in a sandbox. Watch the thing work end to end, on real infrastructure, doing what a user would actually do. Deploy behind a flag, so a mistake is a switch and not an incident. Roll out in stages you can stop.

That list is about software because software makes these checks cheap, not because they only work there. **The general version: write down what "correct" means before you look at anything, then make something other than your own reading decide whether the work meets it.**

- **Analysis.** State the shape the answer must have and what number would prove it wrong. Check the output against the raw data, not the model's summary of it.
- **Legal or regulatory.** Retrieve every cited authority yourself, then separately confirm it has not been overturned. That second step is where fabrication survives.
- **Physical or financial.** Recompute the one number that drives the outcome, by a route the model never touched.
- **Anything.** The sandbox becomes a dry run. The flag becomes a pilot with one client. Staged rollout becomes showing it first to the colleague most likely to object.

When the AI writes the code, write the spec, not the pseudocode. A popular idea says that as AI writes more of the code, the human's job becomes writing pseudocode. Close, but one step short. Pseudocode tells the model *how*: for each document, extract the text, split it, index it, store it. A model can follow every line and still get the important things wrong. A spec tells it *what matters*:

- **Goal.** Uploaded documents become searchable inside their own customer's account.
- **Must never happen.** Duplicates when the same file is uploaded twice. One customer seeing another customer's documents. A failed upload leaving broken state. Old versions staying searchable after an update.
- **Constraints.** Use the existing databases. Never log document contents.
- **Checks that decide.** Upload twice and count. Search as one customer for another's file. Break the indexing step on purpose, then retry.

GitHub's Spec Kit and Anthropic's own guidance for its coding agent both push work this way.[7] [8] Two warnings come with it. First, every "must never happen" line is a failure you had to know exists. Writing the spec does not get you around expertise. It is where your expertise gets spent, and a spec written without it looks just as complete. Second, both tools will happily write the spec for you. Let them draft it, fix it yourself before anything is built, and keep the checks yours, for the reason in the table below. Pseudocode still earns its place in two spots: the one algorithm that must come out exactly as you wrote it, and learning mode, where your bad first attempt is the point (Move 4).

Plain language is a new layer, not a looser one. Dijkstra said it in 1972: "the purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise."[9] Writing in English instead of code does not excuse you from precision. It moves the precision to a new place: what matters, what must never happen, how you will know. If a line in your spec can be read two ways, the model will pick one, and only a check will tell you which.

Make the instructions fit you, not the other way round. Rigid software makes people bend to it, and AI tools are no exception.[10] Your specs, prompts and standing instructions are now part of the program you run. Take the mechanical parts from others: connectors, parsers, tool setups. Write the opinionated parts yourself: what "good" means, which failures matter, how work gets reviewed. Borrow someone else's whole and your work starts to look like theirs. Then keep editing them. Each time a check catches something, fix the instruction that let it through. That loop is how the tool gets better, and how you do.

One question, every time: *what would have caught this, and did I build it before or after I needed it?*

Checks are not interchangeable. Knowing how strong yours is is most of the skill.

Check	Strength	What it misses
Recomputing a number by a second route	Strong	Little, for that number
A test written before the work	Strong for cases you wrote down	Silent about everything else
A sandbox run	Proves it survives	Proves nothing about correctness
Watching it work end to end yourself	Strong for the path you watched	Everything off that path, and rare cases
Opening the cited source	Defeats fabrication	Does nothing about misreading it
A second AI attacking the first	Weakest	Models trained on similar data share blind spots. The error you most need caught is the one both are confident about.
A test the AI wrote for its own work	**Not a check at all**	Everything. It proves the AI agrees with itself, which was never in doubt.

The last row is the one to watch, because it is the only one that looks strong from outside. A check is only a check if whoever wrote it is not the thing being judged. Ask the AI to build the feature and write the tests for it, and the specification, the code and the verdict all come from one author with one set of blind spots. You get a green tick that means nothing. **Whoever writes the check and whoever does the work have to be different.** If you need the AI to write tests because there are too many to write by hand, then you write the specification first, fix it before it generates anything, and treat the tests it produces as more work to review, not as a check on the work.

Use the second-model check because it is nearly free, never because it is reliable. Five weak checks do not add up to a strong one. They tend to be weak in the same place.

Move 4: Decide what you are buying before you type

Bad idea: if the help feels productive, it must be building something lasting in you.

Output mode. Ask, take it, move on. Correct for most work. You do not need to keep that field.

Learning mode. For fields you intend to keep. One rule: **the exchange has to make you produce something.** Attempt it yourself first, badly. Ask what is wrong with *your attempt*, not for the corrected version. Predict what will happen before you run anything. Explain it back and ask where your understanding is wrong.

Learning mode is slower. It is also the only mode that leaves you more capable than it found you. The students whose tool forced them to work kept their ability. The ones handed answers lost theirs and felt great about it.

3. Checking the work does not teach you anything

Say the whole system worked. Spec written first, sandbox held, pipeline refused to promote anything unverified, check passed, work shipped. **What did you learn?**

There is one measure that counts: how you perform once the tool is taken away. By that measure, nothing in the evidence says you learned anything. Three reasons point the other way.

The check fixes the work, not you. Flight simulators teach because *the pilot flies the simulator*. The pilot makes the mistake, under observation, with consequences blocked. A check in front of AI output reverses all of that: **the model made the mistake, the check caught it, and the person who could have learned from it never made it.** Nothing was lost in the catch, and nothing was gained.

You cannot see the difference from inside. The check never comes down, so you never see how you would do without it. A year of checked work where you learned nothing looks identical to a year where you learned a great deal. Same green ticks. Nobody can tell the two apart, including you.

Iterating against the check. Check fails, you re-prompt, fails, re-prompt, passes, ship. Technically that is verification. Actually it is guessing with the answer key in hand, and it is worse than blind acceptance in one way: the version that finally passes has been tuned to satisfy your written check, and written checks never capture everything that matters. Researchers have measured this with models. Keep taking whichever answer scores best on an imperfect check, and the true quality first rises, then falls.[11]

The worst version of this is letting the AI write the check too. Then re-prompting until it passes is just waiting for one author to agree with itself.

The fix costs one sentence. **Before re-prompting, write down what you think went wrong and why.** Then ask for the diagnosis, not the fix. If you cannot write the sentence, the task just told you which bucket it belongs in. The model agreeing with your sentence proves nothing, since you can both be wrong the same way. The real test is whether the fix your sentence implies is the one that works.

Checking protects the work. Practice builds you.

> **Friction at the output boundary** (the check) **protects the work and does not teach you.**
>
> **Friction inside your own loop** (attempt before ask, prediction before run, diagnosis before re-prompt) **teaches you and does not protect the work.**

You can have either without the other. An organisation that checks everything has bought safety and no learning. A person in learning mode has bought learning and no safety. Neither produces the other as a side effect. If you want both, and for any field you plan to keep you want both, install them separately and on purpose.

4. If you are the expert

The same evidence, read from the other side of the expertise line, flips the obvious strategy.

- **Use the tool least where you are strongest.** Every study that split results by skill found the smallest gains at the top, sometimes none. And the acceleration you feel there is exactly the signal you agreed to stop trusting.
- **Use it hardest one step outside your specialty, in learning mode**, where your structure does the checking and the tool speeds up learning instead of replacing it.
- **Protect a floor of unassisted work, structurally.** Skill fades when you delegate, and you cannot see it happening.[12] Aviation, with one of the strongest safety cultures of any industry, now requires manual practice by regulation: US airline pilots must repeat manually flown manoeuvres in the simulator on a fixed schedule.[13] Even so, US auditors found two of the nine airlines they visited discouraging pilots from hand-flying in normal operations.[14] A floor that depends on goodwill erodes. Pick a task class you always do by hand, and a review you always do before opening the model's version. Make skipping it visible.
- **Shift from using your judgment to encoding it.** Acceptance specs for work you will not review. Evaluation suites that decide what ships. Planted-defect tests that measure other people's over-trust. Execution is what the tool made cheap. Judgment that works when you are not in the room is what everyone is short of.

5. If you run a team

Three moves. None is a training course.

Move risk decisions out of people's heads and into the pipeline. A surgical safety checklist cut deaths in randomised trials,[15] then was rolled out across **1,002,241 Medicare patients** with its full safety-culture programme attached, and moved nothing measurable.[16]

Those are two different studies, with different patients, eras and designs. So take only what the pair supports: **a control that depends on a busy person choosing to apply it properly has been tried at population scale and produced no measurable result.** That is why Section 2's buckets are worth using on your own work and not worth mandating across a hundred people on a deadline. The same judgment written as policy-as-code, scoped credentials and staged deployment is made once, in advance, by people not under deadline, then enforced by a system that does not get tired.

Pair every speed metric with a slow quality metric. Defect escape and 60-to-90-day rework, at system level, kept out of individual performance reviews. Cheap generation inflates every activity number while pushing costs onto whoever maintains the thing later. **A team shipping faster with rising 60-day churn is not more productive. It is moving work later in time.** Self-reported productivity does not count as evidence either, for the same reason: the only large survey to model both found self-reported gains rising while modelled delivery performance fell, in the same population.[17]

Put juniors on the building side of the checks. Check everything, staff juniors behind the checks, and you have built the safest possible environment for learning nothing, while using up the supply of people who will one day have to design the checks. Put them on the evaluation suites, test harnesses and failure-mode specs instead. They build verification infrastructure the firm needs and acquire the failure knowledge that verification skill is made of. That work pays for itself. That matters. Germany's dual apprenticeship system has run for decades with firms choosing for themselves whether to train.[18] There, the apprentice's own productive work pays back about **two-thirds** of what they cost the firm.[19] Automate that output away and the arithmetic stops working.

6. A thirty-minute test on yourself

You cannot confirm any of this by introspection, so end with a score. The exercise is lightweight and has never been formally tested, but it is biased in your favour, which makes one of its two outcomes meaningful.

Pick a type of work one step outside your specialty, where a source of truth exists that you can open afterwards: a spec, the dataset, the authority, the standard. If no such source exists for the work you actually do, this test will not work, and the honest substitute is asking someone who knows the field to check your review.

1. In a **separate** session, have a model plant three defects of the kinds that bite: an inverted condition, a cited source that does not exist, an edge case silently dropped.
2. Keep the record where you cannot see it.

3. Review the work **unaided**, under a realistic time limit. Review it with AI help and you are measuring one model catching another model's plants, not measuring yourself.
4. Score yourself. Then open the record.

The two outcomes are not equal. You knew defects were there, which real work never tells you, so your score overstates your real detection rate. **A miss under those easy conditions is a strong warning**, and acting on it costs an hour of caution against the exact failure this essay is about. So act on it: move your bucket boundaries up one level, then confirm with one more run on fresh material from the raised boundary. Boundaries come back down only after a later clean run on fresh material, never by retesting the same material until you pass, which is iterating against the check aimed at yourself. Passing proves little, and is not what the test is for.

One caveat. The test cannot separate what you failed to notice from what you never knew, since spotting an inverted condition or a fake citation draws on knowledge as much as attention. For the decision it feeds, where to set your boundaries, that does not matter, because a miss counts against you either way. It does mean the score is not a measure of you.

7. What to do this week

One action per role. Each under a day, each checkable by Friday.

- **Working outside your expertise.** Run the self-test once. On a miss, raise your bucket boundaries one level.
- **The expert.** Pick the one task type you will always do by hand. Write one acceptance spec for work you will not review.
- **The team lead.** Pair one speed metric with a 60-to-90-day rework metric. Move one risk decision out of discretion into pipeline policy.
- **The executive or buyer.** At the next vendor renewal, ask for one artefact of verification, meaning what was checked and what the check returned, wherever you currently accept a promise that checking happened.

8. How much to trust all this

This essay compresses a longer working paper, *A New Era of Intelligence Work*, which carries the full evidence, the ways this advice could fail, and what would prove it wrong. Four things to know before you take the advice.

- **The measurements are real and individually limited.** The developer study and its 2026 follow-up, the physician trial, the student trial, the checklist evaluation, the

survey data. The paper states the limits of each.

- **The author failed at this too.** He built a personal memory system, designed a reconciliation check for it, specified it in detail, and never switched it on. Stale claims sat there for a month looking exactly like fresh ones. The person telling you to install the check once specified one and left it off.
- **Two central ideas are arguments, not findings.** The evidence rule in Section 1 and the split between checking and practice are assembled *from* the evidence. No single study reports them.
- **Nobody has tested whether this works.** No study yet measures whether people who adopt these practices end up better calibrated or better protected. The paper describes what such studies would look like. Until someone runs them, the honest description is the one the paper accepts for itself: **designed from the evidence, not yet measured by it.**

What is not in doubt is the shape of the problem. The cost of producing plausible-looking work has collapsed. The cost of knowing whether it is any good has not moved. Everything here is one discipline for living on the wrong side of that gap: route work by what it costs to be wrong, let something outside you do the judging, and never confuse the friction that protects the work with the friction that builds the person.

9. Sources

This essay compresses a longer working paper, *A New Era of Intelligence Work: Verification, Institutions, and the Pathway in Human-AI Cooperative Knowledge Work* (working draft, 2026), which carries roughly ninety sources, each with its evidential standing stated. The paper is available from the author on request.

1. METR, “Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity,” July 2025. doi:10.48550/arXiv.2507.09089
2. METR, “Measuring AI Uplift: An Update,” February 2026. Explains why the later data gives an unreliable signal.
3. “Automation Bias in Large Language Model-Assisted Diagnostic Reasoning among Physicians Trained in AI Literacy,” *NEJM AI* 2026;3(5). doi:10.1056/AIoa2501001
4. Bastani H. et al., “Generative AI without guardrails can harm learning: Evidence from high school mathematics,” *PNAS* 2025;122(26):e2422633122. doi:10.1073/pnas.2422633122
5. Sharma M. et al., “Towards Understanding Sycophancy in Language Models,” *International Conference on Learning Representations* (ICLR) 2024. Analyses real

human preference data. The authors say the behaviour is “likely driven in part” by human ratings, a hedged claim. doi:10.48550/arXiv.2310.13548

6. Amodei D., Christiano P. and Ray A., “Learning from Human Preferences,” OpenAI, 13 June 2017. A single simulated case, reported by the lab that trained it. Illustration only. openai.com
7. GitHub, “Specification-Driven Development (SDD),” `spec-driven.md`, github/spec-kit repository, undated, accessed 20 October 2026. Vendor practice guidance, not evidence that the practice improves outcomes. github.com
8. Anthropic, “Best practices for Claude Code,” Claude Code documentation, accessed 20 October
    1. Vendor practice guidance, not evidence that the practice improves outcomes. code.claude.com
9. Dijkstra E.W., “The Humble Programmer,” ACM Turing Award Lecture, *Communications of the ACM* 1972;15(10):859-866. An essay, cited for the principle. doi:10.1145/355604.361591
10. Van Clief J., “What Do We Build Now That AI Can Do the Work?”, talk, YouTube, 20 October 2026. A practitioner’s argument that software should fit its user. It reports no data. youtube.com
11. Gao L., Schulman J. and Hilton J., “Scaling Laws for Reward Model Overoptimization,” *Proceedings of the 40th International Conference on Machine Learning*, 2023. A controlled test with a model standing in for the human judge. It shows the shape of the effect, not its size in real work. doi:10.48550/arXiv.2210.10760
12. Bainbridge L., “Ironies of Automation,” *Automatica* 1983;19(6):775-779. The deskilling argument. doi:10.1016/0005-1098(83)90046-8
13. US Code of Federal Regulations, 14 CFR § 121.423, “Pilot: Extended envelope training.” Recurring simulator training in manually controlled slow flight, loss of reliable airspeed, and instrument departure and arrival. Introduced by FAA final rule 78 Fed. Reg. 67800 (12 November 2013), compliance required from March 2019. eCFR
14. US Department of Transportation, Office of Inspector General, *Enhanced FAA Oversight Could Reduce Hazards Associated With Increased Use of Flight Deck Automation*, Report AV-2016-013, 7 January 2016. PDF
15. Biccard B.M. et al., meta-analysis of surgical safety checklist trials, *South African Medical Journal* 2016.
16. Reames B.N. et al., “Evaluation of the Effectiveness of a Surgical Checklist in Medicare Patients,” *Medical Care* 2015;53(1):87-94. doi:10.1097/MLR.0000000000000277

17. DORA, *Accelerate State of DevOps Report 2024*, Google Cloud. Every variable is survey-derived, which is why this essay uses it as corroboration only.
18. Federal Institute for Vocational Education and Training (BIBB), “Funding arrangements in Germany.” Firms train voluntarily and non-training firms pay no levy, except in sectors with mandatory funds such as construction. bibb.de
19. Pfeifer H., Risius P., Schönfeld G., Wehner C. and Wenzelmann F., *Training in Germany: an Investment to Counter the Skilled Worker Shortage. Results of the 2017/18 BIBB Cost-Benefit Survey*, BIBB Report 3/2020. Average gross cost €20,855 per apprentice-year, returns €14,377, net cost €6,478. bibb.de

References 15 and 16 are two different studies, with different patients, eras and designs. Section 5 explains what the pair can and cannot support.