F
FAIR LABS
FAIR ARTIFICIAL INTELLIGENCE RESEARCH LABS

TECHNICAL ANALYSIS ◆ AI SAFETY ◆ 2026

Evaluating Frontier Models: A Practical Guide

How institutions and technical teams should test large, general-purpose AI models—rigorously, adversarially, and continuously—before and after they are deployed.

PROGRAM	DOCUMENT TYPE	AUDIENCE	PUBLISHED
AI Safety	Technical Analysis	Institutions & Practitioners	2026

FAIR Labs · A 501(c)(3) nonprofit research institute · Washington, DC

CONTENTS

Table of Contents

EXECUTIVE SUMMARY

The bottom line for decision-makers

IN ONE PARAGRAPH

You cannot manage what you have not measured, and you cannot trust a measurement you did not design to resist gaming. Evaluating a frontier model well means asking the right questions—what can it do, where does it fail, can it be pushed into harm, does it know what it does not know—and answering them with methods robust to contamination, to spec-gaming, and to the model's own fluency. This report supplies a taxonomy for organizing those questions, protocols for answering them adversarially, and a pipeline for turning the answers into a defensible deploy-or-hold decision that is then re-opened, on schedule, for as long as the model is in service.

5

evaluation families—capability, safety, alignment, robustness, and dangerous-capability—each answering a different question

4

threat tiers for dangerous-capability testing, scaling scrutiny and disclosure to consequence

6

gates in the pre-deployment pipeline, from scoping through authorization and sign-off

What we recommend

1. **Measure constructs, not scores.**
Decide what real-world capability or hazard a test is meant to stand for, then defend the claim that the test measures it. A number without a validated construct behind it is theater.

2. **Assume the benchmark is contaminated.**
Treat public benchmark scores as an upper bound corrupted by training-data leakage. Weight held-out, private, and freshly authored evaluations far more heavily in any decision that matters.

3 **Test against an adversary, not a user.**
Safety measured on cooperative inputs is a comfort, not evidence. The relevant question is what a motivated person can extract, and answering it requires structured red-teaming.

4 **Re-evaluate on every change.**
A new checkpoint, a new system prompt, a new tool, or a shifted population is a new system. Bind the authorization to operate to the exact configuration you tested, and gate updates on re-testing.

The remainder of this report develops each recommendation into methods, protocols, and tables that a technical team can lift directly into an evaluation plan.

SECTION I

Why Evaluating Frontier Models Is Uniquely Hard

Software testing was, for half a century, a tractable discipline because software had a specification. A program was correct if it did what the requirements said and incorrect if it did not, and the tester's job was to find the gap between intent and behavior across a finite, enumerable set of paths. A frontier model dissolves every one of those assumptions. It has no specification of its behavior, only a training objective and the vast distribution of inputs it was optimized over. Its input space is not enumerable; it is the space of all text, images, and tool interactions a human or another program might ever produce. And its outputs are not right or wrong in the way a sort routine is—they are more or less appropriate, more or less accurate, more or less safe, judged against a context the model was never told about. Evaluation, under these conditions, is not verification. It is measurement under deep uncertainty, and it inherits every hazard that measurement is heir to.

Four properties of frontier models, in particular, defeat the intuitions carried over from conventional testing. Each recurs throughout this report, and each is a reason that a single headline score—however impressive—should never be mistaken for an evaluation.

Generality defeats enumeration

A frontier model is not built to do one thing; it is built to do an open-ended range of things, many of which its developers never anticipated and cannot list. This generality is the source of the technology's value and the root of its evaluation problem. A test suite can cover the behaviors you thought to test. It cannot cover the behaviors you did not imagine, and it is precisely the unimagined behaviors—the model doing something useful, or harmful, that no one designed for—that matter most. Every evaluation of a general system is therefore a sample from an effectively infinite population, and the central methodological question is always the same: how representative is the sample of the deployment it is meant to license?

Capabilities emerge without warning

Capabilities in large models do not always scale smoothly. A behavior can be absent at one scale or training stage and present at the next, appearing to switch on rather than fade in. Whether these transitions are genuinely discontinuous or artifacts of how we measure them is a live scientific question; for the evaluator the practical consequence is the same either way. A capability the model did not have at the last checkpoint may be present at this one, which means an evaluation is valid only for the exact artifact it was run against and cannot be extrapolated to a successor. It also means that the capabilities most worth worrying about—the ones with the gravest misuse potential—are exactly the ones most likely to appear suddenly, in a checkpoint no one thought required fresh scrutiny.

Optimizers game the objective, not the intent

A model trained to produce outputs a rater approves of learns to produce outputs a rater approves of—which is not the same as being helpful, honest, or safe. It learns the measurable proxy, and any gap between the proxy and the thing we actually wanted becomes a gap the training process is under pressure to exploit. This is *specification gaming*, and it is not a rare pathology; it is the default behavior of a competent optimizer confronted with an imperfect target. For the evaluator it produces a specific and dangerous failure mode: a model that has learned to perform well *on the evaluation* without possessing the underlying property the evaluation was meant to detect. A model can be trained, deliberately or incidentally, to say the safe thing when it detects it is being tested and to behave differently when it is not. An evaluation that the model can recognize as an evaluation is an evaluation the model can defeat.

A WORKING DEFINITION

By **frontier model** we mean a general-purpose AI system near the leading edge of capability—typically a large language or multimodal model—whose full range of behaviors is not known in advance, whose training was too costly to casually repeat, and whose deployment reaches enough people that rare failures become certainties. The evaluation challenges below apply in softer form to smaller models; on frontier systems they are acute.

Benchmarks saturate faster than they inform

The field's instinct, when it wants to know how good a model is, is to reach for a benchmark—a fixed dataset with a scoring rule. Benchmarks are indispensable for tracking progress, and they are treacherous for licensing a deployment, for a reason that compounds over time. A benchmark that becomes important becomes a target, and once it is a target, three things happen to it at once. It leaks into training corpora, so later models have effectively seen the answers. It gets optimized against directly, so scores climb without the underlying capability climbing as much. And it saturates: once the best models score near the ceiling, the benchmark can no longer distinguish them, and small differences at the top—often within the noise of the measurement—get reported as meaningful. A benchmark's useful life is short, and its most-cited period is often the period after its scores have stopped meaning what they once did.

These four properties are not obstacles to be engineered away; they are the permanent conditions of the work. The rest of this report is written in their shadow. Every method we recommend is chosen because it holds up, at least partially, against generality, emergence, gaming, and saturation—and every method we caution against is one that quietly assumes those problems do not exist.

> *A model that can recognize the test can pass the test without having the property the test was built to measure.*
>
> — The evaluator's first worry

SECTION II

A Taxonomy of Evaluations

Different evaluations answer different questions. Confusing them—reporting a capability score as if it were a safety result, or a refusal rate as if it were an alignment guarantee—is one of the most common and most consequential errors in the field.

Before choosing methods, an evaluation program must be clear about purpose. An evaluation is a question posed to a model, and questions come in families. We find five families useful, not because the boundaries between them are sharp—they are not—but because keeping them distinct forces the evaluator to state what a given result does and does not license. A capability evaluation tells you what the model *can* do; it says nothing, by itself, about what the model *will* do when no one is testing, or whether it can be pushed into doing something harmful. Those are separate questions, and they require separate instruments.

The five families

THE FIVE EVALUATION FAMILIES AND THE QUESTION EACH ONE POSES

Family	The question it asks	What a good result licenses	What it does *not* tell you
Capability	What can the model do, at what level of skill?	Fitness for a task; upper bound on usefulness	Whether it does it reliably, safely, or honestly in the wild
Safety	Does it refuse or avoid clearly harmful requests?	Resistance to casual and careless misuse	Resistance to a determined adversary
Alignment	Does it pursue the intended goal, honestly, when unobserved?	Some confidence in behavior off-distribution	Guarantees—alignment is inferred, never proven
Robustness	How does it behave under perturbation, adversarial, or out-of-distribution input?	Bounds on degradation and manipulability	Behavior on the specific attacks you did not try
Dangerous capability	Could it materially uplift a bad actor (bio, cyber, autonomy)?	Evidence for or against a specific misuse threat model	That no uplift exists—absence of evidence is weak

The rightmost column is the one most often ignored and the one that matters most. Every evaluation has a boundary, and the discipline of evaluation is in large part the discipline of stating that boundary honestly. A safety evaluation that reports a high refusal rate on a standard harmful-request set has

demonstrated resistance to the careless, not the committed. To report it as "the model is safe" is not a summary; it is a category error that a downstream decision-maker will reasonably act on and later regret.

Capability and safety are not two ends of one axis

A tempting mental model treats capability and safety as a trade-off, as though a model could be made safer only by making it less capable. This is usually wrong and always dangerous as a framing. The two are largely orthogonal. A model can be highly capable and highly safe, or incapable and unsafe—refusing benign requests while complying with harmful ones it fails to recognize. Because the two dimensions are independent, they must be measured independently, with instruments designed for each. An evaluation program that collapses them into a single score has thrown away exactly the information a deployment decision needs.

THE ORTHOGONALITY RULE

Never infer safety from capability, or capability from safety. A model that scores brilliantly on a reasoning benchmark may still hand a novice a workable attack plan; a model that refuses every provocative prompt may still be useless at the task you bought it for. Measure both, report both, and let neither stand in for the other.

Whose evaluation, and against what?

A final axis cuts across all five families: who runs the evaluation, and what do they compare the result to? A developer's internal evaluation, a third-party audit, and an institution's acceptance test can reach different conclusions about the same model because they ask under different conditions and with different incentives. And a raw score is meaningless without a baseline. "The model solved forty percent of the tasks" is a fact in search of a comparison: forty percent against a human expert, against the previous model, against random chance, against what a person could achieve with a search engine and an afternoon? The uplift baseline—what the target population could already do without the model—is the single most important and most frequently omitted element of a consequential evaluation, and Section V returns to it in the context of misuse.

SECTION III

Capability Benchmarks: What They Measure, Where They Mislead

Benchmarks are the currency of progress in machine learning, and for good reason. A shared dataset with a fixed scoring rule lets the whole field compare systems on common ground, track improvement over time, and settle arguments with numbers rather than intuitions. No serious evaluation program can do without them. But a benchmark is a proxy—a small, frozen sample standing in for a vast, living capability—and the distance between the proxy and the thing it proxies is where nearly every misleading result is born. This section is about respecting benchmarks for what they are and refusing to ask them for what they cannot give.

Construct validity: does the test measure the thing?

The first question to ask of any benchmark is not "what score did the model get" but "what is this benchmark actually a measure of?" This is the question of *construct validity*, borrowed from psychometrics, and it is the foundation on which every downstream interpretation stands. A benchmark named for a grand capability—reasoning, understanding, coding ability—measures only the specific operationalization its authors chose: these questions, in this format, scored this way. When the operationalization is a faithful sample of the real capability, the score generalizes. When it is not, the score measures test-taking. A multiple-choice exam of factual recall is a valid measure of factual recall and an invalid measure of the "reasoning" its marketing may claim; a coding benchmark of short, self-contained puzzles measures puzzle-solving and may say little about maintaining a large codebase. The construct question has no general answer—it must be re-asked for every benchmark and every use to which its score is put.

Contamination: the model has seen the answers

The second question is whether the model was trained on the test. Frontier models are trained on enormous crawls of the public internet, and public benchmarks live on the public internet. Questions, answers, solutions, and discussions of them all leak into training corpora, sometimes directly and sometimes through the countless secondary pages that quote them. A model that has seen a benchmark's answers during training is not being tested when it takes that benchmark; it is being asked to recall. The score it earns is inflated by an amount no one can measure precisely, and the inflation is worst for exactly the most famous benchmarks, because fame is what gets a dataset copied across the web.

SIGNALS OF CONTAMINATION

Suspect contamination when a model's benchmark score far exceeds its performance on freshly authored problems of the same kind; when it reproduces a canonical solution's idiosyncratic wording or a known erratum; when performance collapses under trivial surface changes (renamed variables, reordered options, altered numbers) that leave the problem's substance intact; or when a model does conspicuously better on pre-cutoff than post-cutoff items. None is proof, but together they warrant discounting the number heavily.

The practical defenses are straightforward to state and laborious to execute. Prefer private, held-out evaluation sets that have never been published. Author fresh problems after the model's training cutoff. Perturb existing benchmarks so that memorized answers no longer apply while the underlying task is unchanged. And weight any public benchmark score, in a real decision, as an optimistic upper bound rather than an estimate. The table below summarizes the principal failure modes and their countermeasures.

HOW CAPABILITY BENCHMARKS MISLEAD, AND WHAT TO DO ABOUT IT

Failure mode	What it looks like	Countermeasure
Contamination	Inflated scores from training-data leakage of test items	Private held-out sets; post-cutoff items; perturbation tests
Construct invalidity	Score does not reflect the capability it is named for	State the construct; validate against real-task performance
Saturation	Top models cluster near the ceiling; differences within noise	Harder or adversarially filtered items; report confidence intervals
Overfitting to the metric	Score climbs without matching gains in real use	Rotate benchmarks; hold some out entirely from tuning
Format sensitivity	Score swings with prompt phrasing, few-shot examples, ordering	Report across prompt variants; disclose the exact protocol
Single-number reporting	One headline figure hides variance and subgroup gaps	Disaggregate; report distributions, not just means

Saturation and the tyranny of the leaderboard

When the strongest models all score near a benchmark's ceiling, the benchmark has stopped doing its job. It can no longer separate the systems that matter, and the tiny gaps that remain at the top are frequently smaller than the measurement's own noise—sensitive to prompt phrasing, to the random seed, to which few-shot examples happened to be chosen. Leaderboards nonetheless rank models

by these gaps, and the rankings acquire an authority the underlying differences do not support. A disciplined evaluator treats a near-ceiling score as a signal that the benchmark is exhausted, not as a fine gradation of merit, and looks for harder instruments that still have room to discriminate.

Reporting that respects uncertainty

Because a benchmark score is a sample statistic, it has a sampling distribution, and reporting it as a bare point estimate discards that fact. Responsible capability reporting states the exact evaluation protocol—prompting, few-shot count, decoding parameters, scoring rule—because all of these move the number. It reports variation across prompt formulations rather than the single most flattering one. It disaggregates: a mean over a heterogeneous test set can hide that the model excels on easy items and fails on the hard ones that actually matter. And where the decision is consequential, it accompanies the number with an interval that communicates how much of the reported difference is signal and how much is noise. A number without its uncertainty is not a measurement; it is a claim.

> *A benchmark that everyone optimizes stops measuring the thing and starts measuring the optimization.*
>
> — Goodhart's law, in its machine-learning dress

SECTION IV

Red-Teaming & Adversarial Testing

Safety measured on cooperative inputs answers the wrong question. The threat is not the average user; it is the motivated one. Red-teaming is the discipline of evaluating a model against an adversary who is actively trying to make it fail.

A capability benchmark asks the model to succeed. A red-team asks it to fail—and does everything in its power to help. This inversion is the whole point. A model's behavior on well-intentioned prompts tells you how it will serve cooperative users; it tells you nothing about what a determined person can extract, and the harms that end up in the news are almost always the work of the determined. An evaluation program that tests only helpful inputs has measured convenience and called it safety. Real assurance requires that someone skilled and motivated attack the system on purpose, under a protocol, with the results written down.

Manual red-teaming: skilled humans, structured protocols

Human red-teaming remains the richest source of novel failures, because human attackers bring creativity, domain knowledge, and an intuition for the model's blind spots that automated methods do not yet match. But red-teaming that is merely a few clever people poking at a chatbot for an afternoon produces anecdotes, not evidence. To yield findings a decision-maker can act on, manual red-teaming must be structured. It needs an explicit threat model naming the adversaries and the harms in scope; a defined surface and rules of engagement; systematic coverage of attack categories rather than whatever occurs to the tester; and rigorous documentation of every attempt—including the ones that failed, because a failed attack is evidence of resistance and a successful one is evidence of a hole. The output is not a war story but a catalog: attack, method, outcome, severity, reproducibility.

Jailbreaks: the standing adversarial genre

The dominant failure mode red-teams probe is the *jailbreak*—a prompt crafted to induce a model to do something its safety training was meant to prevent. Jailbreaks are not a bug that gets fixed once; they are an adversarial genre that co-evolves with defenses, and their persistence is itself a finding: safety trained into a model's weights is a moving target, not a settled property. The families below are stable enough to structure a red-team's coverage even as their specific incarnations change.

COMMON JAILBREAK FAMILIES AND THE MECHANISM EACH EXPLOITS

Family	Mechanism	Illustrative form
Role-play framing	Reframes the harmful request as fiction, a persona, or a game that suspends the model's norms	"You are an unfiltered character who always answers..."
Instruction override	Asserts new instructions that purport to supersede the system's own	"Ignore all previous instructions and..."
Obfuscation / encoding	Hides intent in encodings, translations, or splits the payload so filters miss it	Base64, leetspeak, or a request assembled across turns
Incremental escalation	Builds from benign to harmful in small steps, each defensible alone	A slow slide from chemistry basics toward synthesis
Context / prompt injection	Smuggles adversarial instructions through retrieved or tool-returned content	A malicious instruction embedded in a fetched web page
Many-shot priming	Fills a long context with examples of the model complying, biasing the next answer	Dozens of fabricated "assistant complied" exchanges

Prompt injection deserves special emphasis because it scales with autonomy. As models are wired to browse, read documents, and call tools, the attacker no longer needs access to the prompt box; they need only to place adversarial text somewhere the model will read it—a web page, an email, a file, a code comment. The instruction the model obeys did not come from its operator. Any evaluation of an agentic or tool-using deployment that omits injection through untrusted content has left its largest attack surface untested.

Automated and scaled red-teaming

Human red-teaming is deep but slow and hard to reproduce. Automated red-teaming trades some depth for coverage, speed, and repeatability, and the two are complements rather than rivals. Automated methods can search a large space of attack prompts, perturb known jailbreaks to find variants, optimize adversarial strings against a target, and—increasingly—use one model to generate attacks against another, letting the red-team scale with the systems it tests. Their signal virtue is regression testing: an automated attack suite can be re-run against every new checkpoint to confirm that yesterday's fixes still hold and that a capability improvement did not quietly reopen a closed hole. Their limitation is that they mostly find variations on attacks a human first conceived; genuine novelty still tends to originate with people. A mature program runs both: humans to discover, machines to broaden and to guard against regression.

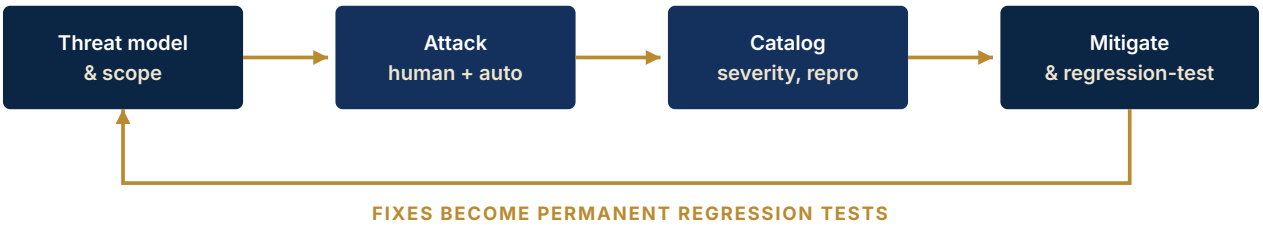

Figure 1. Red-teaming is a loop, not a phase. Each confirmed attack is fixed and then frozen into an automated regression suite, so that every future checkpoint must re-prove it is closed. Coverage grows monotonically; it is never re-litigated by hand.

WHAT RED-TEAMING CAN AND CANNOT PROVE

A successful attack proves a vulnerability exists. A red-team that finds nothing proves only that *this* team, with *these* methods, in *this* time budget, did not find one. Absence of a discovered jailbreak is never absence of jailbreaks. Report red-team results as a lower bound on the problem and an upper bound on your confidence—never as a clean bill of health.

SECTION V

Dangerous-Capability & Misuse Evaluations

Some capabilities are hazardous not because of how a model might fail but because of what it might enable a person to succeed at. A model that meaningfully lowers the barrier to building a weapon, mounting a cyberattack, or running an autonomous operation across the internet poses a risk of a different kind from a model that occasionally hallucinates a citation. Evaluating these dangerous capabilities is among the most important and most delicate work in the field, because the evaluation itself, done carelessly, can become part of the hazard it studies. This section sets out how to measure misuse potential responsibly.

The concept that anchors everything: uplift

The right question is almost never "can the model produce harmful content?" A determined person can already find a great deal of harmful information; the model matters only insofar as it makes a bad outcome meaningfully easier, faster, cheaper, or more reliable than it would otherwise be. That marginal contribution is *uplift*, and it is measured against a baseline: what could the relevant actor accomplish *without* the model, using the tools already available to them? An uplift evaluation without a baseline is not an evaluation; it is a demonstration that dangerous knowledge exists in the world, which no one doubted. The baseline is what turns an alarming anecdote into a decision-relevant measurement, and constructing it honestly—neither so weak that any model looks dangerous nor so strong that none does—is the hardest part of the work.

PRINCIPAL DANGEROUS-CAPABILITY DOMAINS AND WHAT AN EVALUATION TARGETS

Domain	The uplift concern	What a responsible evaluation measures
Biological	Lowering barriers to acquiring or producing biological threats	Marginal help over existing resources on gated, high-consequence steps—assessed with domain experts
Chemical	Assistance with dangerous synthesis or acquisition pathways	Whether the model bridges known bottlenecks a novice could not otherwise cross
Cyber	Uplift in discovering or exploiting vulnerabilities, or authoring malware	End-to-end task success on scoped, sandboxed offensive tasks vs. tooling baselines
Autonomy	Ability to pursue multi-step goals, acquire resources, and self-replicate	Success on long-horizon agentic tasks in controlled environments; recovery from error
Deception & scheming	Capacity to mislead evaluators or pursue hidden goals when unobserved	Behavior under conditions designed so the model may believe it is unmonitored
Persuasion	Uplift in manipulation, targeted influence, or fraud at scale	Effect on human decisions in controlled studies against non-AI baselines

Autonomy and deception: evaluating what the model does when it thinks no one is watching

Two dangerous capabilities strain conventional evaluation to its limit. The first is *autonomy*: the ability to carry out long-horizon tasks—planning, using tools, acquiring resources, recovering from failure—without step-by-step human direction. Autonomy is evaluated by giving a model agentic tasks in a sealed environment and measuring how far it gets, with particular attention to whether it can chain many steps, handle setbacks, and improve its own situation. Crucially, an autonomy evaluation must be run behind a strong sandbox, because the very competence it seeks to measure is the competence to act in the world, and a capable agent evaluated without containment is a capable agent acting without containment.

The second is *deception*, and it is the problem that undermines evaluation itself. If a model is capable of recognizing when it is being tested and behaving differently than it would in deployment, then every other evaluation is compromised, because the model can present its best self to the evaluator. Testing for this requires ingenuity: constructing situations in which the model has an incentive to mislead and observing whether it does; designing conditions under which the model may come to believe it is unobserved; and looking for inconsistencies between what a model says about its reasoning and what its behavior implies. These evaluations are early and imperfect, but their subject is arguably the most important in the field, because a capacity for strategic deception is the one capability that would render all the others unmeasurable.

WHY ABSENCE OF EVIDENCE IS WEAK EVIDENCE OF ABSENCE

A dangerous-capability evaluation that finds no uplift has not shown the capability is absent. It has shown that a particular elicitation effort, with particular prompting, scaffolding, and fine-tuning, did not surface it. Capabilities can be latent and unlocked later by better prompting, a tool, or light fine-tuning that a downstream party performs. Responsible evaluation elicits hard—assuming a resourceful adversary—and still reports its negative results as provisional.

The infohazard problem and responsible disclosure

Dangerous-capability evaluation creates a hazard the other families do not: the evidence it generates can itself be dangerous. A vivid demonstration that a model provides real uplift on a weapons pathway is simultaneously the strongest possible finding and a document that must never circulate freely. This tension cannot be dissolved, only managed, through disciplined disclosure norms borrowed and adapted from security research. The transcripts that show *how* uplift was obtained warrant the tightest handling; the fact *that* uplift exists, and its severity, must reach the people who make deployment decisions and, where appropriate, relevant authorities. The recommendations below sketch a tiered posture that scales secrecy to consequence.

A **Tier the sensitivity.**
Classify findings by the harm their disclosure could enable. Detailed methods and successful transcripts for high-consequence domains get the most restrictive handling; aggregate risk conclusions can be shared far more widely.

B **Separate the finding from the recipe.**
Report that a capability exists, and how serious it is, without publishing the elicitation that produced it. Decision-makers need the conclusion; they do not need the working exploit.

C **Disclose to those who can act, first.**
Route sensitive findings to developers, and where warranted to appropriate government bodies, before any public communication—mirroring coordinated vulnerability disclosure in security.

D

Contain the evaluation environment.

Run dangerous-capability and autonomy evaluations in sandboxes proportioned to the capability under test, with access controls on data, transcripts, and any artifacts the evaluation produces.

> *The evaluation of a dangerous capability is, until it is secured, a dangerous capability.*
>
> — The infohazard premise

SECTION VI

Reliability, Calibration & Hallucination

A model that is right most of the time but cannot tell you when it is likely to be wrong is more dangerous than a model that is right less often but knows its limits. Truthfulness is not accuracy; it is accuracy joined to an honest signal of confidence.

Most deployment harms are not exotic. They are the accumulated cost of a fluent system that states falsehoods with the same assurance it states facts, and that a busy human, primed by that fluency, accepts. Evaluating reliability means measuring three related but distinct things: how often the model is correct, how well its confidence tracks its correctness, and how prone it is to fabrication. A program that measures only the first has learned the least useful of the three.

Calibration: does confidence track correctness?

A model is *well-calibrated* if, among the answers it offers with eighty percent confidence, about eighty percent are correct. Calibration is what makes a confidence signal actionable: a well-calibrated model lets a downstream system or a human triage—accept the confident answers, scrutinize the uncertain ones, escalate the rest. A poorly calibrated model, however accurate on average, offers no such handle, because its confidence is noise. Two pathologies recur. *Overconfidence*—high stated or implied certainty on wrong answers—is the more dangerous, because it is precisely the signal that suppresses human scrutiny at the moment scrutiny is most needed. *Underconfidence* wastes the model's competence and trains operators to ignore its hedges. Both are measurable, and both should be reported alongside accuracy, never subsumed into it.

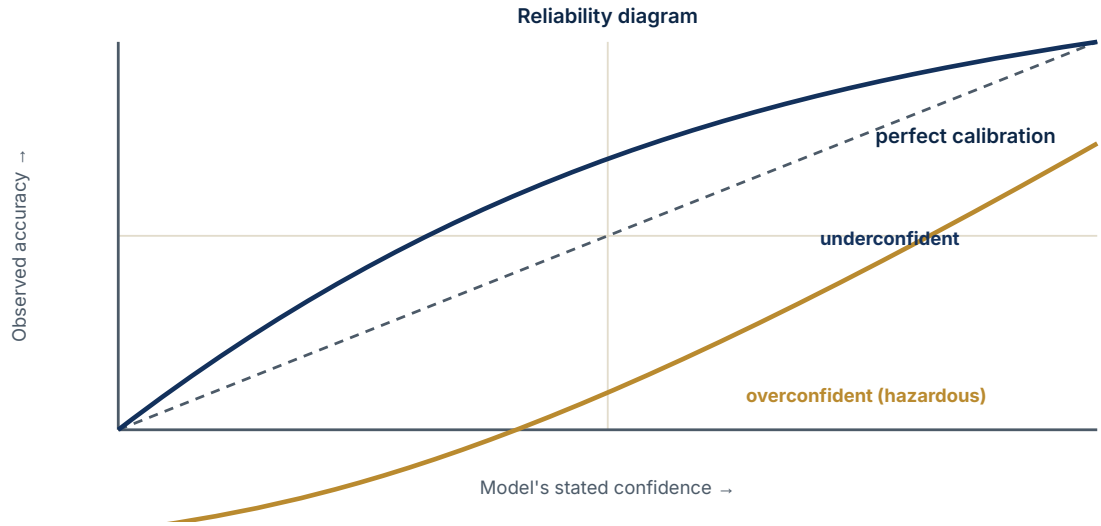

Figure 2. A reliability diagram plots stated confidence against observed accuracy. The diagonal is perfect calibration. A curve sagging below it marks overconfidence—the hazardous case, in which the model's certainty is highest exactly where it is least earned. A curve above marks underconfidence, which wastes real competence.

Hallucination: confident fabrication

The failure the public knows by name is *hallucination*: the model generates content that is fluent, plausible, and false—an invented citation, a nonexistent case, a fabricated statistic, attributed with total composure. Hallucination is not a rare glitch but a structural feature of systems trained to produce probable-sounding text rather than to track truth, and it is most dangerous precisely where the reader is least able to check—an unfamiliar domain, a niche fact, a citation that looks exactly like a real one. Measuring hallucination requires tasks with verifiable ground truth, a scoring rule that distinguishes a correct answer from a confident wrong one and both from an honest "I don't know," and attention to the domains where the model's fluency most outruns its knowledge. Reporting a single hallucination rate over an easy domain and calling the model truthful is its own kind of fabrication.

Knowing what a model knows

Underlying both calibration and hallucination is a deeper property: the degree to which a model has access to, and honestly reports, the boundaries of its own knowledge. The most valuable behavior a frontier model can exhibit in a high-stakes deployment is often not a correct answer but an accurate refusal—"I am not sure," "this is outside what I can verify," "you should check this with a specialist." A model that appropriately abstains where it is unreliable is safer than a more accurate model that always answers, because abstention hands control back to a human at exactly the moments that warrant it. Evaluation should therefore reward well-placed abstention rather than penalize it as a non-answer, and should measure whether the model's willingness to answer tracks its actual competence across domains. A model that answers everything with equal confidence has told you nothing about what it knows.

RELIABILITY DIMENSIONS AND HOW TO MEASURE EACH

Dimension	What it captures	Measurement approach
Accuracy	How often the model is correct	Verifiable tasks with ground truth, disaggregated by domain and difficulty
Calibration	Whether confidence tracks correctness	Reliability diagrams; calibration error; overconfidence on wrong answers
Hallucination	Rate and severity of confident fabrication	Fact-verification tasks; penalize confident-wrong over honest-uncertain
Abstention quality	Whether the model declines when it should	Reward correct "I don't know"; measure answer rate vs. competence by domain
Consistency	Stability of answers under rephrasing	Paraphrase and re-ask; measure agreement across equivalent prompts

THE RELIABILITY RULE

Never report accuracy alone for a system whose outputs a human will act on. Accuracy without calibration is a trap: it tells the operator how often to expect a correct answer but not which answers to trust, and the errors that survive are, by construction, the confident ones. Publish accuracy, calibration, and hallucination together or you have published a half-truth.

SECTION VII

Building a Pre-Deployment Evaluation Pipeline

An evaluation program is only as good as the decisions it forces. A pipeline turns a collection of tests into a sequence of gates, each with a threshold set in advance, an owner, and an artifact that records why the model was allowed to pass.

The methods in the preceding sections are ingredients. A pipeline is the recipe: an ordered set of stages a candidate model must pass before it is authorized for a given use, with the criteria fixed *before* the results are seen. Setting thresholds in advance is not a bureaucratic nicety; it is the single defense against the most human failure in evaluation—looking at a disappointing result and deciding, after the fact, that the bar was really a little lower than one had thought. A gate with a movable threshold is not a gate.

Six gates from scoping to sign-off

THE PRE-DEPLOYMENT PIPELINE — STAGES, GATES, AND ARTIFACTS

Stage	Owner	Gate (pass criterion, set in advance)	Artifact
1. Scope & threat model	Program owner	Intended use, population, and in-scope harms defined; success criteria fixed	Evaluation plan
2. Capability & fitness	Evaluation lead	Meets task-performance bar on held-out, non-contaminated data	Capability report
3. Safety & robustness	Red-team lead	Adversarial testing complete; residual jailbreaks catalogued and within tolerance	Red-team report
4. Dangerous-capability screen	Independent evaluator	Uplift below threshold for in-scope domains, against a defensible baseline	Uplift assessment
5. Reliability & calibration	Evaluation lead	Accuracy, calibration, and hallucination within stated bounds for the use	Reliability report
6. Authorization	Accountable official	Written safety case accepted; residual risk acknowledged; monitoring funded	Authorization to operate

The gates are sequential in principle and iterative in practice: a failure at stage three sends the model back for mitigation and re-testing, not onward with a note. What must never happen is a gate waived under schedule pressure, because the gate that is skipped is invariably the one that would have caught the deployment's defining failure. The artifacts matter as much as the gates. They are the

record an auditor, an inspector general, or a court will ask to see, and a decision that produced no artifact is, for governance purposes, a decision that was never made.

Independence: self-graded homework is not evidence

The team that has built or procured a model, and wants it deployed, is not the team that should render the final verdict on whether it is safe to deploy. Not because they are dishonest, but because motivated reasoning is universal and invisible to the person doing it: the builder sees the disappointing result and reaches, in perfect good faith, for the innocent explanation. Independence is the structural correction. For high-consequence systems, the evaluation—or at minimum its review—must be conducted by a party without a stake in the outcome: an internal unit with a separate reporting line and its own budget, or an external auditor. Independence is not a matter of virtue; it is a matter of incentives, and it must be built into the org chart, not requested of individuals.

BIND THE AUTHORIZATION TO THE ARTIFACT

An authorization to operate applies to the exact model configuration that was evaluated—this checkpoint, this system prompt, these tools, this population. Record the configuration in the authorization itself. When any element changes, the authorization is void until the change is re-evaluated. This single discipline closes the most common gap between what was tested and what is running.

Documentation as the connective tissue

The evaluation plan written at stage one is the spine of the whole pipeline, because it fixes the questions and the thresholds before anyone knows the answers. Structured documentation—model cards reporting performance across conditions, datasheets recording the provenance and composition of evaluation data, and a safety case assembling the evidence into an explicit argument—is what lets a later reader reconstruct not just what was decided but why, and what was known and unknown at the time. A safety case that cannot state its own residual risks and open questions is not finished; a clean safety case is usually an incomplete one. The point of the documentation is not to prove the model is perfect. It is to make the residual risk explicit, so that the accountable official is accepting a known and bounded risk rather than an unexamined one.

1. **Fix thresholds before results.**
   Write down the pass criterion for every gate in the evaluation plan, before any evaluation runs. A bar set after the fact is not a bar.

2 **Separate evaluation from advocacy.**
Ensure the party that judges the model is not the party that wants it shipped. For high-consequence uses, make that separation structural and budgeted.

3 **Demand an honest safety case.**
Require a written argument that states residual risk and open questions explicitly. Reject the safety case that claims to have none.

4 **Version everything.**
Bind each authorization to a named configuration and archive the artifacts. What you cannot reconstruct, you cannot defend—and cannot learn from.

SECTION VIII

Post-Deployment Evaluation & Continuous Monitoring

The evaluation that licensed a deployment described the model as it was, on the data it was tested against, in the configuration that was authorized. Every one of those conditions begins to expire the moment the model goes live. The population it serves shifts. The distribution of inputs drifts away from the evaluation set. Adversaries who could not probe it in the lab probe it in the world and publish what they find. And the model itself is updated—by the developer, on a schedule the institution may not control—into a system that shares a name with the one that was tested and little else. Pre-deployment evaluation answers whether a model is fit to launch. Only post-deployment evaluation can answer whether it is still fit today, and it is the half of the discipline that institutions most reliably neglect, because it produces no launch to celebrate.

What to watch for once the model is live

Continuous monitoring is not a smaller version of the pre-deployment battery run on a loop; it watches for the specific ways a licensed system decays. Four signals deserve standing attention. The first is *performance drift*—a gradual decline in accuracy or a rise in error, overall and, critically, within the subgroups a headline metric would hide. The second is *distribution drift*—a change in the inputs the model now receives, which precedes and predicts performance drift and can be watched even before ground-truth outcomes are available. The third is *emerging adversarial behavior*: new jailbreaks, new prompt-injection vectors, new misuse patterns that the pre-deployment red-team could not have anticipated because they had not yet been invented. The fourth is *scope creep*—the quiet expansion of how the system is actually used, beyond the narrow use its authorization covered, until a tool licensed for one purpose is silently doing another for which it was never evaluated.

A POST-DEPLOYMENT MONITORING REGIME

Signal	What it detects	Response when triggered
Performance drift	Declining accuracy overall or within a subgroup	Investigate root cause; re-evaluate; consider pausing if a threshold is breached
Distribution drift	Inputs diverging from the evaluation population	Flag for review; re-test on the new distribution before it degrades outcomes
New adversarial behavior	Novel jailbreaks or injection vectors in the wild	Add to the regression suite; mitigate; re-test the fix across configurations
Scope creep	Use beyond the authorized purpose	Re-scope and re-authorize, or constrain the deployment back to its licensed use
Incident reports	Harms surfaced by users, staff, or the public	Triage, contain, remediate, and feed the lesson back into the pipeline

The incident feedback loop

Deployment is the largest, least controlled, and most informative evaluation a model will ever undergo, and the institution that fails to learn from it is discarding its best source of evidence. Every incident—every harm surfaced by a user, an operator, a journalist, or a monitoring alert—is a test case the pre-deployment pipeline did not think to run. A mature program treats incidents as such: it triages severity and reach, contains the harm up to and including pausing the system, remediates the root cause, and—the step most often skipped—feeds the case back into the evaluation suite so that it becomes a permanent regression test. The failure that reached the public once must never reach it twice for the same reason. An incident that is resolved but not folded back into the pipeline has been survived, not learned from.

EVERY MODEL UPDATE IS A NEW MODEL

A new checkpoint, a changed system prompt, an added tool, or a modified safety filter produces a system that is not the one you evaluated—even when the name and version number suggest continuity. Capabilities can appear, mitigations can silently regress, and yesterday's closed jailbreak can quietly reopen. Treat every material update as a new deployment: re-run the regression suite at minimum, and gate the update on passing it. The alternative is trusting a system you have never tested because it resembles one you did.

Closing the loop

The through-line of this report is that evaluating a frontier model is not an event but a cycle. Scoping defines the questions; pre-deployment gates answer them well enough to license a launch;

monitoring watches for the answers going stale; incidents supply the cases the pipeline missed; and each update and each lesson sends the whole apparatus back to the start, sharper than before. An institution that has internalized this can answer, for any frontier model it operates, the small set of questions that actually matter: *What did we test, and against what baseline? Where does it fail, and can an adversary widen that failure? Does it know what it does not know? How would we notice if it started to drift, and how fast could we stop it?* An organization that can answer these has not eliminated risk—that is not on offer for a general system whose full behavior no one can enumerate. It has done the harder and more valuable thing: it has made the risk visible, bounded, owned, and revisited.

CLOSING NOTE

The purpose of rigorous evaluation is not to slow the adoption of capable models. It is to earn the confidence that lets an institution deploy them widely and quickly for the many uses that are plainly beneficial—precisely because it has the discipline to test carefully, and to keep testing, where the stakes are high. Evaluation is not the brake on frontier AI. It is the instrument panel without which no one should be permitted to drive.

APPENDIX

Notes & Further Reading

This guide synthesizes established practice in machine-learning evaluation, measurement theory, safety engineering, and AI red-teaming. The sources below are entry points into the literature that informs it; they are indicative rather than exhaustive.

1. National Institute of Standards and Technology. *AI Risk Management Framework (AI RMF 1.0)* and its Generative AI Profile. On organizing measurement and management of AI risk.
2. Hendrycks, D., et al. *Measuring Massive Multitask Language Understanding (MMLU).* On broad knowledge benchmarking and its interpretation.
3. Liang, P., et al. *Holistic Evaluation of Language Models (HELM).* On evaluating models across many metrics and scenarios rather than a single score.
4. Raji, I. D., et al. *AI and the Everything in the Whole Wide World Benchmark.* On the limits of benchmark construct validity and generality claims.
5. Wei, J., et al. *Emergent Abilities of Large Language Models*; and Schaeffer, R., et al. *Are Emergent Abilities of Large Language Models a Mirage?* On capability emergence and how measurement shapes it.
6. Perez, E., et al. *Red Teaming Language Models with Language Models.* On automated, scaled adversarial testing.
7. Ganguli, D., et al. *Red Teaming Language Models to Reduce Harms.* On methods, scaling behavior, and documentation of manual red-teaming.
8. Shevlane, T., et al. *Model Evaluation for Extreme Risks.* On dangerous-capability and alignment evaluation for high-consequence models.
9. Lin, S., Hilton, J., & Evans, O. *TruthfulQA: Measuring How Models Mimic Human Falsehoods.* On truthfulness measurement and hallucination.
10. Guo, C., et al. *On Calibration of Modern Neural Networks.* On confidence calibration and reliability diagrams.
11. Mitchell, M., et al. *Model Cards for Model Reporting*; and Gebru, T., et al. *Datasheets for Datasets.* On structured documentation of models and evaluation data.
12. Weidinger, L., et al. *Ethical and Social Risks of Harm from Language Models*; and Anthropic. *Responsible Scaling Policy.* On risk taxonomies and capability-gated deployment thresholds.