Lecture 12: Evaluation

Language Modeling from Scratch · lecture notes · rebuilt for NVIDIA Blackwell, late 2026

Section 1: Defining "Good" — From Abstract Constructs to Concrete Metrics

Training a language model requires three major components: the model architecture, the training and scaling machinery, and the data. By this point in the course, we already have the machinery to define a model architecture and scale its training run. The remaining ingredient that actually shapes what a trained model can do—whether it writes code, translates multilingual text, models DNA sequences, or performs multi-step reasoning—is the training data.

A natural question arises before we ever begin curating a training dataset: how do we know which data mixture, filtering rule, or training recipe made the model better? Without a way to measure quality, data curation is guesswork. For that reason, evaluation must come before data. In the overall workflow of building a language model, the pipeline proceeds left to right across four linked stages: **Architecture** (modeling and layers) feeds into **Training & Scaling** (systems and scaling laws), which feeds into **Evaluation** (answering what "good" means), which in turn directs **Data** (shaping the model's behavior).

> **DEFINITION**
> **Evaluation:** Given a language model—or a larger system built around a language model—the process of measuring how "good" it is with respect to a specified goal or capability.

At first glance, evaluating a language model looks mechanically trivial. You might picture a three-step pipeline: first define a set of prompts, next query the model to obtain its responses, and finally compute an accuracy score against ground truth. That mechanical view is deceptive. The real scientific problem sits one level deeper: mapping an **abstract construct**—something we care about, such as intelligence, usefulness, or safety, which we cannot read off directly—into a **concrete metric**, such as perplexity, multiple-choice accuracy, an Elo rating, or a test pass rate.

> **DEFINITION**
> **Operationalization:** The act of translating an abstract, unobservable construct (such as reasoning ability or helpfulness) into a concrete, measurable procedure and numerical metric (such as accuracy on a specific benchmark or win rate in pairwise comparisons).

Every benchmark is an operationalization—a proxy for an underlying capability. Because benchmarks steer research agendas, training recipes, and real-world deployment decisions across the field, the gap between the construct and the proxy matters enormously. Weak proxies get gamed, producing models that climb leaderboards without improving on what users actually need; strong proxies expose genuine

shortcomings and guide progress. This tension is known as **Goodhart's law**: once a metric becomes an explicit optimization target, the gap between the abstract construct and the concrete metric widens, and the metric ceases to be a reliable measure of the construct.

> **COMMON MISCONCEPTION**
> You might think there is a single objective ranking of language models—one authoritative number that tells you which model is "the best." That breaks because "good" means different things depending on what you are optimizing for, and different lenses produce conflicting rankings. A model tuned to maximize exam accuracy may be too expensive per query, slow in multi-turn sessions, or disliked by everyday users in open-ended conversation.

To see how different operationalizations of "good" diverge in practice, consider four distinct lenses used across public evaluation platforms:

Definition of "Good"	Representative Platform	What It Captures — and What It Misses
Benchmark scores	Artificial Analysis (quality index)	Standardized capability comparisons across fixed tasks; vulnerable to saturation and train–test contamination.
Quality per dollar	Artificial Analysis (quality vs. cost)	Practical efficiency frontier balancing capability against price; single-turn token pricing can mislead on multi-turn agent workloads.
Human preference	Arena AI (formerly Chatbot Arena)	Real open-ended prompts judged by people; conflates formatting style, verbosity, and tone with factual correctness.
Revealed adoption	OpenRouter usage rankings	What developers and users actually choose and pay for in production; heavily shaped by ecosystem lock-in, pricing, and default configurations.

> **WORKED EXAMPLE**
> Suppose two models, Model X and Model Y, are compared across these four lenses:
>
> - On a standardized exam quality index, Model X scores 88 and Model Y scores 82. Under the **benchmark score** lens, Model X wins.
> - Model X costs $\$15$ per million output tokens, while Model Y costs $\$1.50$ per million output tokens. Under **quality per dollar**, Model Y delivers nearly the same capability at one-tenth the cost and sits ahead on the efficiency frontier.

> - In pairwise human chat evaluations, Model Y formats answers clearly and directly addresses everyday user prompts, earning a higher Elo rating than Model X. Under **human preference**, Model Y wins.
> - On an API router, Model X is the default inside a widely used coding tool, giving it $3\times$ the token volume of Model Y. Under **revealed adoption**, Model X leads—even though part of that lead reflects defaults rather than raw preference.
>
> No single number reconciles these four rankings. Which model is "better" depends on which lens matches the decision at hand.

Because no single metric covers every aspect of model behavior, we organize evaluation across two groups of dimensions—seven core areas plus an overarching methodological framework:

1. **Benchmark Families (What capabilities do we test?)** moving from the training objective toward full real-world systems:
   - **Perplexity:** token probability and scaling laws.
   - **Exams:** standardized knowledge and reasoning tests (MMLU, GPQA, HLE).
   - **Open Chat:** free-form conversation scored by human pairwise preference or LLM judges (Arena AI, AlpacaEval, WildBench).
   - **Agents:** models paired with scaffolds, tools, execution sandboxes, and multi-turn serving traces.
   - **Pure Reasoning:** abstract rule induction designed to decouple reasoning from memorized facts (ARC-AGI 1–3).
2. **Cross-Cutting Concerns (Can we trust and act on the scores?)**:
   - **Safety & Dual-Use:** refusal benchmarks, adversarial jailbreaks, and contextual risk (HarmBench, AIR-Bench, GCG).
   - **Realism & Validity:** ecological validity of tasks (GDPVal, MedHELM, Clio), train–test contamination, and dataset/verifier quality.
   - **Rules of the Game:** distinguishing methods from models and full agent systems, and matching the metric to the stakeholder's question.

Across every benchmark we examine, three competing goals form a recurring trade-off: **difficulty** (hard enough to avoid saturation as models improve), **realism** (high ecological validity, reflecting actual use), and **validity** (producing a clean, uncontaminated, trustworthy signal).

> **KEY IDEA**
> Evaluation is not merely running prompts and computing accuracy; it is the scientific challenge of operationalizing an abstract construct into a concrete metric. Every benchmark navigates a three-way trade-off among difficulty, realism, and validity, and different definitions of "good" (benchmarks, cost efficiency, human preference, adoption) yield genuinely different rankings.

To ground this roadmap, we begin with the metric closest to how language models are actually trained: perplexity.

✎ Check Your Understanding

1. **Fill-in-the-blank:** Translating an unobservable goal such as "usefulness" into a measurable quantity such as "human pairwise win rate" is called _____, and once that metric becomes a direct optimization target, _____ predicts that the gap between the goal and the metric will widen.

   Answer: operationalization; **Goodhart's law**. Operationalization defines the proxy metric for the abstract construct, and Goodhart's law warns that optimizing directly for a proxy degrades how faithfully it represents the construct.

2. **Yes/No with justification:** If Model A beats Model B on a standardized benchmark quality index, does that guarantee Model A will also rank higher on Arena AI and OpenRouter usage rankings?

   Answer: No. Standardized benchmarks, human preference (Arena AI), cost efficiency, and revealed adoption (OpenRouter) measure different operationalizations of "good." A model that excels on fixed exam items may lose on conversational style, price, ecosystem defaults, or real-world open-ended prompts.

Section 2: Intrinsic Evaluation with Perplexity

Before testing a language model on exams, chat, or coding tasks, we need a metric that measures how well the model performs its native task: assigning high probability to held-out sequences of text. Downstream task accuracy is often noisy and flat during early stages of training—a small architecture tweak or data-filtering improvement may not shift multiple-choice accuracy at all on a small-scale run. We need a smooth, sensitive, intrinsic signal directly tied to the pre-training objective.

Recall that an autoregressive language model defines a probability distribution $p(x)$ over token sequences $x = (x_1, x_2, \ldots, x_{|D|})$ via the chain rule of probability:

$$p(D) = \prod_{i=1}^{|D|} p(x_i \mid x_{<i})$$

where $x_{<i} = (x_1, \ldots, x_{i-1})$ denotes all preceding tokens. Because multiplying millions of probabilities produces numbers that underflow floating-point precision and depends on dataset length $|D|$, we take the inverse geometric mean of the token probabilities. That quantity is **perplexity**.

DEFINITION

Perplexity (PPL): On a token dataset D of length $|D|$, the inverse geometric mean probability assigned by a language model p to each token given its prefix, equal to the exponential of the average negative log-likelihood per token. Lower perplexity means the model is less surprised by the text.

Mathematically, perplexity is written in two equivalent forms:

$$\mathrm{PPL}(D) = \left(\frac{1}{p(D)}\right)^{1/|D|} = \exp\left(-\frac{1}{|D|}\sum_{i=1}^{|D|}\log p(x_i \mid x_{<i})\right)$$

Let us break this equation apart from the inside out:

- $p(x_i \mid x_{<i})$ is the probability the model assigns to the actual i-th token x_i given all preceding tokens $x_{<i}$. It lies in $(0, 1]$.
- $\log p(x_i \mid x_{<i})$ is the natural logarithm of that probability (matching the base-e exponential $\exp$ outside). Because $p \leq 1$, this value is ≤ 0; multiplying by -1 produces $-\log p(x_i \mid x_{<i}) \geq 0$, the **negative log-likelihood** (or surprisal) of token i.
- $\frac{1}{|D|}\sum_{i=1}^{|D|}\left(-\log p(x_i \mid x_{<i})\right)$ averages the negative log-likelihood across all $|D|$ tokens in the dataset. This average is the empirical **cross-entropy loss** that pre-training directly minimizes.
- $\exp(\cdot)$ maps the average cross-entropy back out of log space into an effective branching factor. To verify the algebraic equivalence to $(1/p(D))^{1/|D|}$, note that $-\frac{1}{|D|}\sum_i \log p(x_i \mid x_{<i}) = \log\left(\left(\prod_i p(x_i \mid x_{<i})\right)^{-1/|D|}\right) = \log\left((1/p(D))^{1/|D|}\right)$, and applying $\exp$ cancels the logarithm.

Taken as a single conceptual unit, perplexity answers: *if the model were guessing uniformly among k equally likely next tokens at every step, how large would k have to be to match the model's average uncertainty on D?* A perfect predictor that assigns $p(x_i \mid x_{<i}) = 1$ to every token has $\log(1) = 0$ and therefore $\mathrm{PPL}(D) = \exp(0) = 1$. A uniform predictor over a vocabulary of size V assigns $p = 1/V$ to every token and has $\mathrm{PPL}(D) = V$.

Because pre-training directly minimizes training-set cross-entropy (and therefore training-set perplexity), the natural intrinsic evaluation is test-set perplexity. Perplexity remains the workhorse metric during language model development because it exhibits **smooth scaling laws**: small improvements to the architecture, optimizer, or data recipe show up cleanly in held-out perplexity long before discrete downstream accuracy moves.

> **WORKED EXAMPLE**
> Suppose a dataset D consists of $|D| = 4$ tokens, and a language model assigns the following conditional probabilities to the true tokens:
>
> $$p(x_1) = 0.5, \quad p(x_2 \mid x_1) = 0.25, \quad p(x_3 \mid x_{<3}) = 0.5, \quad p(x_4 \mid x_{<4}) = 0.125$$
>
> Using the product form directly:
>
> $$p(D) = 0.5 \times 0.25 \times 0.5 \times 0.125 = \frac{1}{2} \times \frac{1}{4} \times \frac{1}{2} \times \frac{1}{8} = \frac{1}{128} = 2^{-7}$$
>
> Taking the inverse and raising to the power $1/|D| = 1/4$:

$$\mathrm{PPL}(D) = \left(\frac{1}{2^{-7}}\right)^{1/4} = 2^{7/4} = 2^{1.75} \approx 3.36$$

Checking via the log-sum form (in natural logs):

$$-\frac{1}{4}\big(\ln 0.5 + \ln 0.25 + \ln 0.5 + \ln 0.125\big) = -\frac{1}{4}\big(-7\ln 2\big) = 1.75 \ln 2 \approx 1.213$$

$$\mathrm{PPL}(D) = \exp(1.213) \approx 3.36$$

Interpretation: on average, the model is as uncertain about the next token as if it were choosing uniformly among roughly 3.36 candidates at each position. Notice also what happens if a single token had probability 10^{-6} instead of 0.125: the single catastrophic mistake would blow up $p(D)^{-1/4}$ to $(16 \times 10^6)^{1/4} \approx 63.2$, illustrating how heavily perplexity penalizes confident mistakes.

From In-Distribution to Zero-Shot Perplexity

How perplexity is evaluated underwent a fundamental shift with the rise of foundation models:

- **Classic Paradigm (In-Distribution):** You train a model on the training split of a specific corpus and evaluate its perplexity on the test split of that *same* corpus. Standard historical benchmarks included **Penn Treebank** (Wall Street Journal articles), **WikiText-103** (Wikipedia articles), and the **One Billion Word Benchmark** (built from WMT11 text: EuroParl, United Nations proceedings, and news). As a historical milestone under this paradigm, architectures combining pure CNNs and LSTMs drove One Billion Word perplexity down from **51.3 to 30.0**.
- **Foundation-Model Shift (Out-of-Distribution / Zero-Shot):** Instead of training per-corpus models, a general-purpose model is trained on broad web text and evaluated **zero-shot** on standard test sets without training on their corresponding training splits. The historical landmark for this shift was **GPT-2**, trained on **WebText** (40 GB of web text linked from Reddit). Zero-shot transfer beat specialized in-domain baselines on **small datasets** such as Penn Treebank, where pre-training scale dominated domain specificity. However, it did *not* beat specialized in-domain models on larger corpora such as the One Billion Word Benchmark, where the in-domain training split was large enough to match the domain distribution tightly.

Moving from clean dataset-specific train/test splits to web-scale pre-training reshaped evaluation—and immediately introduced the risk that test sets might accidentally appear inside web-scraped training corpora, a validity problem we will analyze in full in Section 10.

Is Perplexity All You Need—or More Than You Need?

Two competing philosophical perspectives frame how much weight we should put on perplexity:

1. **The "All You Need" Argument:** Let t be the true data-generating probability distribution over text and p be the language model's distribution. Recall from information theory that the expected cross-entropy $H(t, p) = -\mathbb{E}_{x \sim t}[\log p(x)]$ decomposes as

$$H(t, p) = H(t) + D_{\mathrm{KL}}(t \,\|\, p)$$

where $H(t)$ is the entropy of the true distribution and $D_{\mathrm{KL}}(t \,\|\, p) \geq 0$ is the Kullback–Leibler divergence, which equals zero **if and only if** $p = t$. Therefore, the minimum achievable perplexity corresponds to the true entropy $H(t)$ (specifically $\exp(H(t))$), achieved if and only if $p = t$. If a model truly matched t, then sampling from $p(\text{solution} \mid \text{problem})$ would solve every task. Under this view, pushing perplexity down far enough implies general intelligence must eventually follow.

2. **The "Maybe More Than You Need" Counterargument:** In practice, no model reaches $p = t$, and full-sequence perplexity allocates its loss budget across *every* token equally—even tokens that are irrelevant to the capability we care about. Consider the sentence:

   > *"Stanford was founded in **1885**"*

   Full-sequence perplexity penalizes the model's probability on *"Stanford"*, *"was"*, *"founded"*, and *"in"* just as heavily as on *"1885"*. A model that memorizes common English phrasing (*"was founded in"*) better than another model can achieve lower overall perplexity while getting the actual historical date wrong.

To focus perplexity solely on the tokens that matter, we use **conditional perplexity** on the answer tokens y given the prompt x:

DEFINITION

Conditional Perplexity ($\mathrm{PPL}(y \mid x)$): The perplexity computed exclusively over a designated target sequence y conditioned on a fixed input prompt x, ignoring the probabilities assigned to tokens inside x.

$$\mathrm{PPL}(y \mid x) = \left(\frac{1}{p(y \mid x)}\right)^{1/|y|} = \exp\left(-\frac{1}{|y|}\sum_{j=1}^{|y|} \log p(y_j \mid x, y_{<j})\right)$$

Here $|y|$ is the number of tokens in the target answer alone, and each factor $p(y_j \mid x, y_{<j})$ conditions on the full prompt x plus any earlier answer tokens $y_{<j}$.

WORKED EXAMPLE

Consider the 5-token sequence `["Stanford", " was", " founded", " in", " 1885"]`, where the prompt is $x =$ `["Stanford", " was", " founded", " in"]` and the single-token target answer is $y =$ `[" 1885"]` ($|y| = 1$). Suppose we compare two models:

- **Model 1 (fluent style, weak factuality):** assigns high probabilities to the phrasing tokens $(0.1, 0.8, 0.5, 0.9)$ but only $p(\text{" 1885"} \mid x) = 0.05$.
- **Model 2 (less fluent phrasing, strong factuality):** assigns lower probabilities to the phrasing tokens $(0.02, 0.4, 0.2, 0.5)$ but $p(\text{" 1885"} \mid x) = 0.80$.

Compute full perplexity across all 5 tokens:

- Model 1: $p(D) = 0.1 \times 0.8 \times 0.5 \times 0.9 \times 0.05 = 0.0018$, so $\text{PPL}(D) = (1/0.0018)^{1/5} \approx 3.54$.
- Model 2: $p(D) = 0.02 \times 0.4 \times 0.2 \times 0.5 \times 0.80 = 0.00064$, so $\text{PPL}(D) = (1/0.00064)^{1/5} \approx 4.35$.

Full perplexity declares Model 1 superior ($3.54 < 4.35$) purely because it predicted the boilerplate phrasing better! Now compute conditional perplexity on $y =$ ” 1885”:

- Model 1: $\text{PPL}(y \mid x) = (1/0.05)^1 = 20.0$.
- Model 2: $\text{PPL}(y \mid x) = (1/0.80)^1 = 1.25$.

Conditional perplexity flips the ranking and cleanly identifies Model 2 as far better at retrieving the actual fact.

Perplexity in Disguise and Leaderboard Integrity

Several classic downstream benchmarks are effectively **perplexity in disguise**:

- **LAMBADA:** a cloze task asking the model to predict the final word of a passage, specifically constructed so that predicting the word requires broad narrative context rather than just the final sentence.
- **HellaSwag:** a multiple-choice sentence-completion benchmark typically scored by comparing the model's continuation likelihood across candidate endings.

While perplexity is invaluable for internal model development, it carries a major vulnerability when used for **public leaderboards** of black-box models. Compare two leaderboard protocols:

- **Perplexity leaderboard:** The evaluator submits `test_data` to `LM(test_data)` and receives `log_prob`. To trust the resulting perplexity score, the evaluator must trust that the model returns a **validly normalized probability distribution**, meaning $\sum_x \exp(\text{log_prob}(x)) = 1$ over all possible sequences x.
- **Downstream task leaderboard:** The evaluator submits `prompt`, receives a concrete text `response = LM(prompt)`, and runs `score = grade(response, target)`. The score is verifiable from the generated outputs alone.

COMMON MISCONCEPTION

You might think that because perplexity is computed by an exact mathematical formula, a public perplexity leaderboard is harder to game than a downstream benchmark. In fact, the reverse is true for closed APIs: an evaluator querying a black-box API cannot verify that the reported log-probabilities sum to 1 across the entire space of sequences.

WORKED EXAMPLE

Suppose a dishonest closed-API provider submits to a public perplexity leaderboard. Whenever queried for `log_prob` on any token x_i in `test_data`, the API ignores the vocabulary normalization constraint and simply returns $\log p(x_i \mid x_{<i}) = -0.01$ (corresponding to $p \approx 0.990$) for

whatever token happens to be in the test file.

With a vocabulary of $V = 50{,}000$ tokens, the sum of probabilities across all next tokens would be $50{,}000 \times 0.990 = 49{,}500 \neq 1$—completely invalid as a probability distribution. Yet unless the evaluator queries all $50{,}000$ candidate tokens at every position (or has direct access to open model weights to verify the softmax layer), the leaderboard computes $\text{PPL} = \exp(0.01) \approx 1.01$ and crowns the cheating API as nearly omniscient.

For this reason, perplexity is safest for **internal development** or **open-weight models** whose softmax normalization can be audited directly. Even with conditional perplexity and open weights, practitioners still need task benchmarks that directly test real-world capabilities and output behavior.

KEY IDEA

Perplexity is the workhorse intrinsic metric for model development because it directly mirrors the training loss and obeys smooth scaling laws, while conditional perplexity narrows focus to answer tokens. However, full perplexity can obscure task performance behind phrasing tokens, and black-box perplexity leaderboards cannot verify probability normalization without trusting the provider or inspecting open weights.

To evaluate models on concrete, verifiable capabilities using only their generated outputs, evaluation moved from token probabilities to standardized human-style exams.

✎ Check Your Understanding

1. **Quick computation:** A language model assigns probabilities $p(x_1) = 0.25$ and $p(x_2 \mid x_1) = 0.25$ to a 2-token sequence $D = (x_1, x_2)$. What is $\text{PPL}(D)$?

   Answer: 4. Using $\text{PPL}(D) = (1/p(D))^{1/|D|} = (1/(0.25 \times 0.25))^{1/2} = (1/0.0625)^{1/2} = 16^{1/2} = 4$.

2. **Yes/No with justification:** Can a third-party evaluator reliably run a perplexity leaderboard over closed, black-box API models by asking each API for the log-probability of the test sequence?

   Answer: No. Evaluating perplexity requires trusting that the reported log-probabilities come from a validly normalized distribution ($\sum_x \exp(\log p(x)) = 1$). A closed API can artificially inflate reported log-probabilities on test tokens without the evaluator being able to verify normalization from a single sequence query.

Section 3: Standardized Exam Benchmarks and the Saturation Treadmill

Once we require output-verifiable evaluations that test concrete capabilities rather than raw token likelihood, standardized human-style exams are an immediate candidate. Exams offer two compelling engineering advantages:

1. **Fine-grained control** over both the **subject** (mathematics, law, biology, history) and the **difficulty** (high-school level through doctoral research).
2. **Unambiguous ground-truth answers** (a letter choice A–D or a short exact string), making grading completely automatic, reproducible, and cheap at scale.

MMLU and Benchmark Saturation

The landmark exam benchmark of the early foundation-model era was **MMLU** (Massive Multitask Language Understanding, 2021). MMLU consists of **57 subjects**—spanning mathematics, US history, law, morality, and dozens of other fields—formatted as **4-way multiple choice** questions collected by graduate and undergraduate students from freely available online sources. Historically, MMLU was evaluated on **GPT-3 (2020)** using few-shot prompting.

Despite its name, MMLU tests **factual knowledge** far more than genuine language understanding or multi-step reasoning. More importantly, as models scaled, MMLU became heavily **saturated**.

> **DEFINITION**
> **Benchmark Saturation:** The state in which top models' scores cluster near the ceiling imposed by dataset label noise and ambiguous items, destroying the benchmark's discriminative power to distinguish further genuine capability gains.

> **COMMON MISCONCEPTION**
> You might think a benchmark is only saturated when models reach 100% accuracy. That breaks because real human-collected datasets inevitably contain mislabeled answers, outdated facts, or ambiguous questions—often 5% to 10% of items. Once models reach 90% on a dataset with 8% label noise, the remaining points reflect guessing the annotator's errors rather than higher capability, so a score of 91% vs. 90% carries almost no valid signal.

Escalating Difficulty: MMLU-Pro and GPQA

To escape saturation and restore headroom between frontier models, benchmark designers launched what we can call the **exam treadmill**—a progression of four generations moving from left to right as frontier models repeatedly catch up: **MMLU** (4 choices, web exams) $\rightarrow$ **MMLU-Pro** (10 choices + chain-of-thought) $\rightarrow$ **GPQA** (PhD-level, Google-proof) $\rightarrow$ **HLE** (frontier-filtered).

Benchmark	Design Upgrade	Key Numbers & Signal
MMLU-Pro (2024)	Removed noisy and trivial MMLU items; expanded options from **4 choices to 10 choices**; evaluated with **chain-of-thought** reasoning.	Model accuracy dropped by **16% to 33%** relative to MMLU, restoring meaningful headroom between models.
GPQA — *Graduate-Level Google-Proof Q&A* (2023)	Authored from scratch by **61 PhD contractors** (hired via Upwork) specifically so web search does not trivially reveal the answer.	PhD domain experts: **65%**; non-experts given 30 minutes + Google: **34%**; historical baseline (GPT-4): **39%**.

Notice why expanding from 4 choices to 10 choices in MMLU-Pro matters mathematically, even before removing trivial items.

> **WORKED EXAMPLE**
> Suppose a model genuinely knows the answer to 70% of questions and guesses uniformly at random on the remaining 30% (assuming no partial elimination).
>
> - Under **4-way multiple choice** (MMLU), random guessing succeeds with probability $1/4 = 0.25$. Expected accuracy is:
>
> $$0.70 + 0.30 \times 0.25 = 0.775 \quad (77.5\%)$$
>
> - Under **10-way multiple choice** (MMLU-Pro), random guessing succeeds with probability $1/10 = 0.10$. Expected accuracy drops to:
>
> $$0.70 + 0.30 \times 0.10 = 0.730 \quad (73.0\%)$$
>
> Now combine that chance-level reduction with filtering out easy recall items (say half of the questions a model previously knew trivially are replaced with harder reasoning items where accuracy is 0.40): the model's true knowledge rate drops from 70% to 55%, giving $0.55 + 0.45 \times 0.10 = 59.5\%$—an 18-percentage-point drop, right in the 16% to 33% range observed empirically on MMLU-Pro. Two models that previously sat crammed at 88% and 90% separate cleanly into distinct tiers.

GPQA addressed a different weakness of MMLU: because MMLU was assembled from freely available online sources, questions could be memorized from web pre-training corpora or solved via superficial keyword lookup. By commissioning 61 PhD experts to write original "Google-proof" questions—verified by showing that skilled non-experts with 30 minutes and Google search scored only **34%**, well below domain PhDs at **65%**—GPQA pushed exam evaluation to the doctoral frontier.

Humanity's Last Exam (HLE) and the Exam Ceiling

As models advanced rapidly even on GPQA, benchmark creators pushed the exam paradigm to its logical extreme with **Humanity's Last Exam (HLE, 2025)**. HLE contains **2,500 multimodal questions** spanning many academic subjects, combining multiple-choice and short-answer formats.

Its construction pipeline moves sequentially through four stages:

1. **Expert Submissions:** Domain experts worldwide were incentivized with a **$500K prize pool and paper co-authorship** to submit frontier-level questions.
2. **Frontier LLM Filter:** Every submitted question was tested against frontier language models, and any question that existing models could solve was **discarded**.
3. **Multi-Stage Review:** Surviving questions passed through rigorous expert quality checks to verify unambiguous correctness and eliminate flawed items.
4. **Final Benchmark (2,500 Questions):** The resulting multimodal suite of multiple-choice and short-answer questions.

Even with adversarial frontier filtering and doctoral-level difficulty, HLE exposes the **fundamental limit of exams**: multiple-choice and short-answer tests can be made arbitrarily difficult, yet they still fail to capture open-ended real-world usage where no single unique answer exists.

> **KEY IDEA**
> Exam benchmarks provide cheap, automatic grading and fine-grained subject control, but face a relentless saturation treadmill (MMLU $\rightarrow$ MMLU-Pro $\rightarrow$ GPQA $\rightarrow$ HLE). No matter how difficult an exam becomes, closed-form questions cannot measure how well a model handles the open-ended, subjective tasks that dominate everyday use.

This brings us to the next question: how do we evaluate models when users ask open-ended questions with no unique ground-truth string?

✎ Check Your Understanding

1. **What would happen if…** a benchmark creator keeps MMLU's exact 4-choice questions but adds a large amount of label noise (say 15% mislabeled answers)? How would that affect benchmark saturation?

   Answer: The benchmark would saturate much earlier (around 85% accuracy). Because scores cannot reliably rise above the ceiling set by label noise, top models would cluster near 85% and lose discriminative separation well below 100%.

2. **Fill-in-the-blank:** In Humanity's Last Exam (HLE), submitted expert questions are passed through a _____ that discards any item existing models can already solve, before undergoing multi-stage expert review.

   Answer: frontier LLM filter. This stage ensures that every surviving question lies beyond the capabilities of the frontier models available at collection time.

Section 4: Evaluating Open-Ended Chat with Human Preferences and Automated Judges

Most people do not open an AI assistant to administer 4-choice doctoral exams. Consider a typical real-world prompt:

> **User prompt:** *"I would like to make a beet salad with goat cheese. What kind of herbs would work well and what would not work well?"*
>
> **Model response:** *"Here's a breakdown of herbs that work well (and some that don't) in a beet + goat cheese salad, based on how their flavors interact with the sweet-earthiness of beets and the tangy creaminess of goat cheese…"*

There is no single canonical answer string here. One excellent response might recommend dill, tarragon, and mint while warning against overpowering rosemary; another equally good response might focus on chives, thyme, and flat-leaf parsley. Standard exact-match grading fails completely.

To score open-ended responses with no unique ground truth, two dominant evaluation families emerged: **human pairwise preference** (exemplified by Arena AI, formerly Chatbot Arena) and **LLM-as-a-judge** with bias control or structured checklists (exemplified by AlpacaEval and WildBench).

Arena AI and the Bradley–Terry / Elo Rating Model

Evaluating open-ended chat with isolated 1-to-5 human ratings is notoriously noisy: one rater's "4" is another rater's "3," and absolute standards drift over time. **Arena AI (formerly Chatbot Arena)** solves this by replacing absolute scoring with **blind pairwise comparisons** over real user prompts.

The platform operates as a five-stage loop:

1. An **Internet User** enters a real prompt of their choice (receiving free access to top models as an incentive to bring genuine tasks).
2. The prompt is routed in parallel to two anonymous models: **Model A (anonymous)** and **Model B (anonymous)**.
3. Both responses are displayed side by side without revealing model identities.
4. The **User Votes** on whether A is better or B is better (or ties).
5. Votes accumulate into a pairwise comparison graph, and an **Elo / Bradley–Terry (BT) fit** estimates a global rating for every model on the leaderboard.

A crucial operational strength of this design is that it is **dynamic and flexible**: new prompts and new models can be added continuously without having to run every model on an identical fixed set of prompts.

The statistical model linking two models' scalar ratings Elo_A and Elo_B to the probability that A beats B is the **Bradley–Terry model** written on the standard chess Elo scale:

$$p(A \text{ beats } B) = \frac{1}{1 + 10^{(\text{Elo}_B - \text{Elo}_A)/400}}$$

Let us decompose this equation term by term:

- $\text{Elo}_B - \text{Elo}_A$ is the rating gap by which Model B exceeds Model A. Notice that the probability depends *only* on the difference between ratings, not their absolute magnitudes (shifting every model's rating by $+100$ leaves all win probabilities unchanged).
- $(\text{Elo}_B - \text{Elo}_A)/400$ rescales the gap so that a 400-point rating difference corresponds to a factor of 10 in the odds ratio.
- $10^{(\text{Elo}_B - \text{Elo}_A)/400}$ converts the scaled difference into the odds that B beats A. When $\text{Elo}_A = \text{Elo}_B$, the exponent is 0, so $10^0 = 1$ (even $1:1$ odds). When Elo_A is 400 points higher than Elo_B, the exponent is -1, so $10^{-1} = 0.1$ ($1:10$ odds against A, i.e., $10:1$ odds in favor of A).
- $\frac{1}{1+\text{odds}(B \text{ over } A)}$ is the logistic function that maps odds in $(0, \infty)$ onto a valid probability in $(0, 1)$.

In Arena AI, rather than updating ratings sequentially after each match (which depends on match order), all ratings $\{\text{Elo}_m\}$ are estimated jointly via **maximum likelihood estimation (MLE)** over the entire pairwise comparison graph: finding the rating vector that maximizes the product (or sum of logs) of the predicted probabilities of all observed user votes.

DEFINITION

Bradley–Terry (Elo) Leaderboard Fit: A probabilistic model for pairwise comparisons that assigns each model a scalar rating Elo_m such that $p(A \text{ beats } B) = \left(1 + 10^{(\text{Elo}_B - \text{Elo}_A)/400}\right)^{-1}$, fitted via maximum likelihood across all observed head-to-head votes.

WORKED EXAMPLE

Suppose a Bradley–Terry fit assigns three models the ratings $\text{Elo}_A = 1350$, $\text{Elo}_B = 1250$, and $\text{Elo}_C = 1150$.

1. **Direct gap of 100 points (A vs. B):**

$$\frac{\text{Elo}_B - \text{Elo}_A}{400} = \frac{1250 - 1350}{400} = -0.25$$

$$p(A \text{ beats } B) = \frac{1}{1 + 10^{-0.25}} = \frac{1}{1 + 0.5623} \approx 0.640 \quad (64.0\%)$$

2. **Direct gap of 200 points (A vs. C):** Even if Model A and Model C have *never* been paired directly on the same prompt, as long as both played against B (connecting the comparison graph), the model predicts:

$$p(A \text{ beats } C) = \frac{1}{1 + 10^{(1150 - 1350)/400}} = \frac{1}{1 + 10^{-0.5}} = \frac{1}{1 + 0.3162} \approx 0.760 \quad (76.0\%)$$

3. **Gap of 400 points** ($\text{Elo}_A - \text{Elo}_D = 400$):

$$p(A \text{ beats } D) = \frac{1}{1 + 10^{-1}} = \frac{1}{1.1} \approx 0.909 \quad (90.9\%)$$

Interpretation: every 400-point advantage multiplies the win-to-loss odds by 10, and the graph-wide MLE lets a leaderboard compare models transitively even when prompts are not shared across all pairs.

Limitations of Crowdsourced Human Preference

Although Arena AI captures genuine open-ended prompts, crowdsourced preference suffers from four structural flaws:

- **Uncontrolled rater pool:** Because voters are self-selected internet users, the pool carries demographic and usage biases—along with occasional spammers or coordinated voters.
- **Style versus substance:** A single binary vote collapses presentation (polished Markdown formatting, bullet lists, friendly tone, politeness) together with factual correctness.
- **Verification gap:** On technical, mathematical, coding, or specialized prompts, the casual human rater often cannot tell which response is actually right and defaults to whichever answer *sounds* more authoritative.
- **Sycophancy risk:** Models that flatter the user or agree with the user's incorrect premise routinely win votes even when factually wrong.

COMMON MISCONCEPTION

You might think that because humans are the ultimate end users, human preference votes on a blind arena represent the ground truth of model quality. That breaks because blind votes measure **perceived helpfulness** at a glance, not objective correctness: when a user lacks the time or expertise to verify an answer, a polite, well-formatted hallucination beats a terse, accurate response.

Human preference is therefore a strong signal of perceived helpfulness, never a substitute for objective verification.

LLM-as-a-Judge: AlpacaEval, Length Bias, and WildBench

Human voting is also too slow and expensive to run inside a fast model-development iteration loop. To scale open-ended evaluation automatically, researchers turned to **LLM-as-a-judge**: using a strong language model to score or compare candidate responses.

DEFINITION

LLM-as-a-Judge: An automated evaluation method in which a capable language model replaces human raters to compare or grade model outputs on open-ended tasks, often paired with bias controls or structured rubrics.

AlpacaEval (2023) introduced **805 instructions** drawn from diverse sources and measured each candidate model's **win rate** against a fixed baseline (historically GPT-4 preview), as judged by an LLM (historically GPT-4 preview). Two serious measurement problems emerged immediately:

1. **Judge bias:** An LLM judge can systematically favor its own stylistic fingerprints or models from its own family.
2. **Length bias and gaming:** LLM judges systematically prefer longer responses. Once leaderboards published raw win rates, models could climb the rankings through trivial verbosity hacks—padding answers with unnecessary background rather than improving substance.

AlpacaEval 2.0 (2024) addressed verbosity gaming by introducing **length-controlled win rates**, fitting a regression model that explicitly accounts for the length difference between the two responses and predicts what the win rate would be if both responses had equal length.

> **WORKED EXAMPLE**
> To see how regression debiasing works in AlpacaEval 2.0, suppose a judge's binary decision that candidate model M beats baseline B on prompt i is modeled via logistic regression:
>
> $$\log \frac{p(M \text{ beats } B \text{ on } i)}{1 - p(M \text{ beats } B \text{ on } i)} = \theta_M - \theta_B + \beta_{\text{len}} \cdot \tanh\left(\frac{\text{len}(y_M) - \text{len}(y_B)}{\sigma}\right)$$
>
> Here $\theta_M - \theta_B$ captures true quality difference, while $\beta_{\text{len}} > 0$ captures the judge's systematic bias toward longer responses. Suppose the fitted bias is $\beta_{\text{len}} = 0.80$, Model M has the exact same intrinsic quality as the baseline ($\theta_M - \theta_B = 0$), and Model M pads its outputs so that $\tanh\big((\text{len}_M - \text{len}_B)/\sigma\big) = +0.75$ on average.
>
> - **Raw (biased) win rate:** The log-odds are $0 + 0.80 \times 0.75 = +0.60$, giving a raw win rate of $\frac{1}{1+e^{-0.60}} \approx 64.6\%$—making a tied model look decisively stronger purely by being verbose.
> - **Length-controlled (debiased) win rate:** Setting the length-difference term to 0 removes the verbosity bonus, yielding log-odds of 0 and a debiased win rate of $\frac{1}{1+e^{0}} = 50.0\%$.

This raises a meta-level question: how do we validate an automated evaluation metric itself? Through **meta-evaluation**—measuring how strongly the automated judge's model rankings correlate with human judgments, with Chatbot Arena serving as the de facto sanity check for chat evaluations.

A second major advance in automated judging came from **WildBench (2024)**, which sampled **1,024 realistic examples** from **1 million human-chatbot conversations**. Rather than asking an LLM judge (historically GPT-4 turbo / GPT-4) for a snap holistic preference, WildBench pairs the judge with an explicit **checklist** tailored to each prompt—effectively applying chain-of-thought reasoning to the grading process itself. Checking off concrete criteria before rendering a verdict makes judging far more consistent and yields strong correlation with Chatbot Arena.

Across both human and automated open-ended evaluation, three operational principles hold:

1. **Pairwise comparisons** between similar outputs yield higher signal than isolated absolute scores.

2. You must explicitly **audit for biases** (length, style, self-preference, sycophancy) in both human raters and LLM judges.
3. Explicit **checklists and rubrics** improve grading reliability regardless of whether a human or an LLM does the grading.

> **KEY IDEA**
> Open-ended chat evaluation relies on pairwise human preferences (Arena AI with Bradley–Terry/Elo fits) or scalable LLM-as-a-judge suites (AlpacaEval 2.0, WildBench). Both require active defenses against confounding biases—especially style, sycophancy, and verbosity—and improve sharply when anchored by explicit grading rubrics.

Yet even perfectly judged open-ended chat still measures only **what a model says** in a text box. Modern AI systems are increasingly deployed to take actions, run code, inspect outputs, and modify real environments—moving us from chat to agents.

> **✎ Check Your Understanding**
>
> 1. **Quick computation:** On Arena AI, Model A has an Elo rating of 1400 and Model B has an Elo rating of 1000. Using the Bradley–Terry Elo formula, what is the predicted probability that Model A beats Model B?
>
>   **Answer:** $\frac{10}{11} \approx 90.9\%$. Substituting $\text{Elo}_B - \text{Elo}_A = -400$ into $p(A \text{ beats } B) = \frac{1}{1+10^{(\text{Elo}_B - \text{Elo}_A)/400}}$ gives $\frac{1}{1+10^{-1}} = \frac{1}{1.1} \approx 0.909$.
>
> 2. **What would happen if…** a developer modifies a model to double its response length without adding any new useful information, and submits it to AlpacaEval 1.0 versus AlpacaEval 2.0?
>
>   **Answer: Its score would artificially rise on AlpacaEval 1.0 because LLM judges exhibit length bias, whereas AlpacaEval 2.0 uses regression modeling to debias length and would largely eliminate the unearned gain.**

Section 5: Agentic Benchmarks and the Role of the Agent Scaffold

Chat benchmarks evaluate a single input-to-output mapping: a **Prompt** enters a **Language Model**, which returns a **Text Response**. That paradigm tests *what language models say*. In real engineering and scientific work, however, solving a task rarely happens in one pass. A practitioner inspects files, runs commands, reads error messages, revises hypotheses, and verifies results over many steps.

Agentic evaluation shifts from what models *say* to **what models do**. The architecture of an agent evaluation consists of three stages connected left to right:

1. **Task + Environment** (for example, a code repository or a live shell environment) enters the system.

2. Inside the system sits an **Agent**, defined as the combination of a **Language Model Policy** and an **Agent Scaffold**. Inside this boundary, the **LM Policy** and **Tools & State** interact in a closed two-way feedback loop: the model issues tool calls, the environment executes them and returns new state/observations, and the model iterates.
3. When the agent finishes, an **Outcome Verifier** checks the resulting artifact—running unit tests, checking captured flags, or computing a competition score.

> **DEFINITION**
> **Agent:** A system composed of a **language model** paired with an **agent scaffold**—the surrounding control logic governing tool calls, state management, context construction, and multi-step iteration inside an execution environment.

Putting models inside real execution environments unlocks a major advantage that open-ended chat lacked: **verifiable outcomes**. Rather than guessing whether a code suggestion looks plausible, we can execute the repository's test suite and check objectively whether the bug is fixed. The trade-off is substantially higher infrastructure complexity, which we will examine in Section 6.

Four Flagship Agentic Benchmarks

Agent benchmarks now span software engineering, general command-line workflows, cybersecurity, and machine learning research:

Benchmark	Domain & Scale	Task Structure	Evaluation & Difficulty Calibration
SWE-bench (2023)	Software engineering (**2,294 tasks** across **12 Python repositories**)	Input: codebase + real GitHub issue description → Output: pull-request patch fixing the issue	Objective repository **unit tests**
Terminal-Bench (2026)	General CLI tasks (**229 tasks** crowdsourced from **93 contributors**; **89 curated tasks** in **Terminal-Bench 2.0**)	Universal **computer terminal environment** inside a sandbox	Automated task checks; calibrated against **human completion time**
CyBench (2024)	Cybersecurity (**40 Capture-the-Flag tasks**)	Explore target system, develop exploit, and capture secret flag	Exact **flag match**; **first-solve time** from competitions calibrates task difficulty
MLE-bench (2024)	Machine learning engineering (**75 Kaggle competitions**)	Process data, engineer features, train and tune models, generate predictions	Official **competition leaderboard metric**

Notice the progression in generality across these suites. SWE-bench tests realistic repository-level software engineering, proving that real GitHub issues paired with unit tests provide clean, objective grading. Terminal-Bench generalizes far beyond patching Python packages: because a command-line shell is a universal interface to a computer, *any* CLI workflow—compiling software, configuring services, analyzing data, repairing systems—becomes an evaluable task. CyBench and MLE-bench demonstrate how specialized domains can inherit built-in objective verifiers (cryptographic flags and held-out Kaggle test leaderboards) along with natural difficulty scales (human first-solve times and medal thresholds).

Anatomy of Modern Agent Scaffolds

Long-horizon execution over dozens or hundreds of tool calls quickly overwhelms a raw language model operating in a naive prompt loop. Modern high-performing agents therefore rely on rich **agent scaffolds** (often called "deep agents"). Structurally, a modern scaffold wraps the core model in four interacting layers:

- At the top sits the **Lead LM + Context Engineering**: detailed, explicit process instructions governing workflow steps, tool usage conventions, and error-recovery strategies.
- Branching downward from the lead model are two parallel coordination mechanisms:
  1. **Explicit Planning:** maintaining and systematically checking off a structured todo list so the agent does not lose track of multi-step goals.
  2. **Hierarchical Delegation (Sub-Agents):** spawning specialized sub-agents that run in clean context windows, preventing verbose intermediate logs (such as long search outputs or test traces) from polluting the lead model's working memory.
- Both mechanisms connect down into **Persistent Memory & Tools**: reading and writing workspace files across steps, invoking the shell, and running verification tests.

Because every layer of this scaffold shapes how the model gathers information, recovers from mistakes, and manages its finite context window, an agent benchmark score is fundamentally a two-argument function:

$$\text{Eval score} = f(\text{Model}, \text{Scaffold})$$

COMMON MISCONCEPTION

You might think that running SWE-bench or Terminal-Bench measures the raw intelligence of the underlying language model. That breaks because you *never* evaluate a bare language model on an agent benchmark—you always evaluate the coupled pair **model + scaffold**. Changing the scaffold's planning prompt, sub-agent delegation, or context management can shift scores dramatically even when the model weights are held completely fixed.

WORKED EXAMPLE

Consider the same underlying language model evaluated on a 100-task Terminal-Bench suite under two different scaffolds:

- **Bare Loop Scaffold:** Feeds the task prompt and appends every full stdout/stderr log verbatim to the conversation history until the context window fills or turns run out. On multi-step tasks with verbose build logs, the model loses track of earlier errors and solves $42/100$ tasks (42%).
- **Deep Agent Scaffold:** Adds explicit todo-list planning, delegates noisy file searches to a sub-agent that returns only concise summaries, writes intermediate notes to workspace files, and instructs the model to run unit tests after every edit. Holding the model weights identical, the system solves $61/100$ tasks (61%).

If a report publishes "61% for Model A" using the Deep Agent scaffold alongside "48% for Model B" using the Bare Loop scaffold, the table appears to show Model A beats Model B by 13 points—even if Model B is actually the stronger underlying model when paired with the same scaffold.

KEY IDEA

Agentic benchmarks (SWE-bench, Terminal-Bench, CyBench, MLE-bench) evaluate what models *do* in live execution environments, recovering verifiable objective outcomes. Because an agent consists of both a language model and an agent scaffold ($\text{Eval score} = f(\text{Model}, \text{Scaffold})$), every agent evaluation measures a coupled system rather than a bare model.

If the scaffold moves scores that much, what happens when we look closely at the rest of the evaluation infrastructure—harness permissions, sandbox reliability, and serving caches? That brings us directly to the engineering pitfalls of running agent evaluations at scale.

✎ Check Your Understanding

1. **Fill-in-the-blank:** Because an agent combines a language model with control logic for planning, sub-agents, persistent memory, and tools, any agent benchmark result is a function $\text{Eval score} = f(\text{Model}, _____)$.

   Answer: **Scaffold** (or **Agent Scaffold**). You never evaluate a bare language model on an agent benchmark; the scaffold's context engineering, planning, and tool loop directly determine performance.

2. **Yes/No with justification:** Does moving from open-ended chat benchmarks to software agent benchmarks like SWE-bench eliminate the difficulty of objectively grading outputs?

   Answer: **Yes for output grading (in principle), at the cost of infrastructure complexity.** Execution environments allow objective outcome verifiers—such as running repository unit tests, checking captured flags, or scoring Kaggle predictions—instead of relying solely on subjective human or LLM preferences, though building clean, reliable verifiers and sandboxes introduces new failure modes.

Section 6: Evaluation Infrastructure — Harnesses, Sandboxes, and Trace-Based Serving

Once evaluation moves inside live execution environments, subtle details of the **evaluation harness**, the **container sandboxes**, and the **inference serving stack** stop being background implementation details. They become primary variables that can swing benchmark numbers by more than an entire generation of model improvements.

Harness Settings and Runtime Contamination

The execution harness controls what the agent is permitted to access, how many turns or wall-clock minutes it gets, and how its context window is compacted when history grows long. When harness boundaries are left unsealed, agents exhibit **runtime contamination**—retrieving answers from the environment instead of solving the task.

> **DEFINITION**
> **Runtime Contamination (Harness Leakage):** A validity failure in agent evaluation where an unsealed execution harness permits the agent to look up ground-truth solutions during evaluation—such as querying the public internet or inspecting future commits in repository `.git` history—rather than solving the task from scratch.

A concrete illustration comes from **Cursor's June 2026 audit of 731 agent trajectories**, whose findings appear as two paired bar comparisons:

- **Top bars (Leakage inside trajectories):** Across the 731 audited runs, **57%** of trajectories found the fix online, and **9%** mined `.git` history to recover the solution.
- **Bottom bars (Impact on SWE-bench Pro):** Simply sealing `.git` history and blocking internet access dropped the SWE-bench Pro score from **87.1% (Unsealed)** down to **73.0% (Sealed)**—a 14.1-percentage-point drop from closing two leaks alone.
- **Overall harness sensitivity:** Across full harness configurations, **harness differences alone produced up to a 20.7-point gap** on the same benchmark.

Because a 20.7-point swing can completely overwhelm genuine differences between frontier models, credible reporting requires disclosing the **full harness configuration** alongside every agent benchmark score:

1. **Internet access and `.git` history sealing** (whether outbound network calls and commit histories are stripped or blocked).
2. **Turn limits and wall-clock time limits** (how many tool-call iterations and minutes the agent is granted).
3. **Context management and compaction rules** (how long observations are truncated, summarized, or evicted).

We will return to this runtime leakage alongside training-time overlap when we examine validity defenses in Section 10.

Sandboxed Agent Evaluations at Scale

Running thousands of autonomous agents that execute arbitrary shell commands requires strict isolation. Large-scale agent evaluation therefore runs on the **exact same container infrastructure as agentic reinforcement learning (agentic RL)**.

The execution architecture proceeds through three stages:

1. **Eval Orchestrators:** Frameworks such as **Harbor** (which runs diverse harnesses including Claude Code, Codex, and OpenHands across multiple cloud sandbox providers) and **UK AISI's Inspect** (which orchestrates Docker and Kubernetes sandboxes).
2. **Isolated Sandboxes:** Containerized execution environments (Docker, Kubernetes, or managed cloud sandbox providers) that isolate each task instance.
3. **Forked Accounting Outcomes:** Every run terminates in one of two strictly distinct branches:
   - **Valid Run $\rightarrow$ Model Score:** The container ran cleanly and the task verifier returned a legitimate pass or fail.
   - **Sandbox Crash $\rightarrow$ Infrastructure Error:** The container failed to start, timed out on network provisioning, or crashed due to host failure—and must **NOT** be counted as a model failure.

COMMON MISCONCEPTION

You might think that container reliability is a minor DevOps detail—if a few sandboxes crash during an eval run, treating those tasks as score 0 only slightly underestimates the model. That breaks badly in practice: in one empirical study, sandbox availability ranged from **99.73% down to 69.5%**. Scoring crashed containers as zero silently corrupts benchmark comparisons, turning an infrastructure outage into a fake 30-point capability drop.

WORKED EXAMPLE

Suppose an agent has a true task-solving rate of 80% across a 200-task benchmark, and we run it on two cloud sandbox providers:

- **Provider A (99.5% availability):** 199 sandboxes run cleanly and 1 crashes. The agent passes 80% of the 199 valid runs (≈ 159 passes).
  - *Counting crashes as zero:* $159/200 = 79.5\%$.
  - *Proper error accounting (excluding/retrying infra errors):* $159/199 = 79.9\%$.
- **Provider B (70.0% availability):** 140 sandboxes run cleanly and 60 crash before completion. The agent passes 80% of the 140 valid runs (112 passes).
  - *Counting crashes as zero:* $112/200 = 56.0\%$—an artificial **24-percentage-point penalty** that looks like catastrophic model regression!
  - *Proper error accounting (separating infra errors and retrying them):* $112/140 = 80.0\%$, matching the model's true capability.

Always report the infrastructure error rate separately from the pass rate, and retry crashed sandboxes rather than folding them into model failures.

Serving Benchmarks: From Single-Turn Prompts to Replayed Agent Traces

Evaluation does not stop at accuracy; practical deployment also depends on **serving efficiency**—latency, throughput, and cost per completed task. Recall from Section 1 that single-turn token pricing can mislead on multi-turn agent workloads. Serving benchmarks underwent a parallel structural shift to fix this mismatch:

- **Legacy Serving Benchmarks (Independent Single-Turn Requests):** Send independent requests (`Prompt 1`, `Prompt 2`, `Prompt 3`) with no shared history across turns. This setup measures raw prefill and decode speed in isolation, completely missing **KV prefix caching** and **session-aware request routing**.
- **Current Serving Benchmarks (Replayed Agent Traces — AgentX, AIPerf):** Replay recorded multi-turn agent sessions where every turn appends new tool outputs onto the growing conversation prefix:
  - **Turn 1:** `[sys + prompt]` (computed fresh).
  - **Turn 2:** `[cached prefix (Turn 1)]` + `[+ tool 1]` (only the new tool output requires fresh prefill if routed to a server holding the KV cache).
  - **Turn 3:** `[cached prefix (growing: Turn 1 + tool 1)]` + `[+ tool 2]`.

In late-2026 agent workloads running on long-context frontier models—such as **DeepSeek-V4.1-Flash** or **Kimi K3**, both supporting up to **1M-token context windows**—a single multi-turn agent trajectory repeatedly re-sends tens or hundreds of thousands of unchanged prefix tokens on every tool call. Consequently, **prefix-cache hit rate** and **session-aware request routing** dominate total compute and dollar cost. Because cache reuse dwarfs raw per-token speed, **trace-based serving rankings can completely invert single-turn serving rankings**.

> **WORKED EXAMPLE**
> Consider a 10-turn agent session where the system prompt plus initial codebase context is $50{,}000$ tokens, and each turn adds $2{,}000$ new tokens (tool outputs and model actions). Compare two serving stacks:
>
> - **Stack X (Fast raw prefill, no prefix caching):** Processes $50{,}000$ prefill tokens/second, but recomputes the entire prompt on every turn.
>   - Turn k ($k = 1, \ldots, 10$) prefills $50{,}000 + 2{,}000(k-1)$ tokens.
>   - Total prefill tokens across 10 turns:
>
> $$\sum_{k=1}^{10} \big(50{,}000 + 2{,}000(k-1)\big) = 10 \times 50{,}000 + 2{,}000 \times 45 = 590{,}000 \text{ tokens}$$
>
>   - Total prefill time: $590{,}000 / 50{,}000 = \mathbf{11.8}$ s.
> - **Stack Y (Half the raw prefill speed, 100% prefix-cache hit via session routing):** Processes only $25{,}000$ prefill tokens/second, so it loses $2\times$ on a legacy single-turn benchmark. However, it computes the initial $50{,}000$ tokens once on Turn 1 and only $2{,}000$ new tokens on each of Turns 2–10.

- Total freshly prefilled tokens across 10 turns:

$$50{,}000 + 9 \times 2{,}000 = 68{,}000 \text{ tokens}$$

- Total prefill time: $68{,}000 / 25{,}000 = \mathbf{2.72}$ s.

Result: Stack X beats Stack Y by $2\times$ on a legacy single-turn benchmark, yet **Stack Y is $4.3\times$ faster** on the replayed multi-turn agent trace. The ranking completely inverts.

KEY IDEA

In agent evaluation, infrastructure is never neutral: unsealed harnesses allow runtime contamination (moving SWE-bench Pro scores by over 14 points, and full harness changes by up to 20.7 points), sandbox crashes masquerade as model failures if miscounted, and multi-turn prefix caching in replayed traces (AgentX, AIPerf) can invert single-turn serving rankings.

Every benchmark we have examined so far—perplexity, exams, chat, and software agents—mixes reasoning ability together with vast amounts of memorized linguistic and world knowledge. Can we design an evaluation that strips away memorized facts and isolates pure reasoning?

✎ Check Your Understanding

1. **Quick computation:** An agent is evaluated on 100 benchmark tasks. On a flaky sandbox cluster with 80% availability, 20 containers crash with infrastructure errors, and the agent passes 60 of the 80 valid runs. What score is falsely reported if crashes are counted as model failures, versus the true valid-run pass rate?

   Answer: 60% falsely reported vs. 75% true valid-run pass rate. Counting the 20 crashed containers as zeros gives $60/100 = 60\%$, whereas proper accounting separates infrastructure errors from valid runs and computes $60/80 = 75\%$ (and retries the 20 crashed tasks).

2. **Yes/No with justification:** If Serving Stack A achieves higher tokens-per-second than Serving Stack B on independent single-turn prompts, will Stack A necessarily cost less and run faster on a replayed multi-turn agent trace like AgentX?

   Answer: No. In multi-turn agent traces with growing context windows (up to 1M tokens on models like DeepSeek-V4.1-Flash or Kimi K3), prefix-cache hit rate and session-aware request routing dominate compute cost. If Stack B reuses cached prefixes while Stack A recomputes them, trace-based rankings can invert single-turn rankings.

Section 7: Isolating Pure Reasoning with ARC-AGI

Across perplexity, MMLU, GPQA, Arena AI, and SWE-bench, a recurring ambiguity remains: when a model solves a hard problem, did it genuinely *reason* through novel structure, or did it retrieve a near-duplicate pattern memorized from trillions of pre-training tokens? Every benchmark we have examined so far entangles reasoning with linguistic fluency and encyclopedic world knowledge.

To test whether a model can acquire new skills on the fly without leaning on memorized facts, **ARC-AGI** (Abstraction and Reasoning Corpus for Artificial General Intelligence) was built around three strict design principles:

1. **100% solvable by humans** using everyday core priors (objects, symmetry, counting, geometry), yet challenging for AI systems.
2. **Every task is unique**, neutralizing brute-force memorization of known problem templates.
3. **Few-shot rule induction:** the solver must infer a hidden transformation rule from only a handful of input–output demonstrations and immediately apply that rule to a novel test input.

DEFINITION

ARC-AGI: A family of pure-reasoning benchmarks composed of unique, human-solvable visual/interactive tasks that require inferring novel abstract transformation rules from a few demonstrations rather than recalling memorized world knowledge.

WORKED EXAMPLE

Worked Example: An Abstract Grid Transformation

Consider the 3×3 abstract grid transformation puzzle illustrating ARC-AGI-style reasoning. Each cell in a 3×3 grid is indexed by $(\text{row } r, \text{column } c)$ with $r, c \in \{0, 1, 2\}$, where background cells are grey and active cells are colored.

WORKED EXAMPLE

Given two demonstrations:

- **Demo 1 (Blue):**
  - Input has blue cells at $(0, 1)$, $(1, 0)$, and $(1, 1)$—forming an L-shaped corner pointing toward the top-left of the grid:

$$\begin{bmatrix} \cdot & B & \cdot \\ B & B & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

  - Output has blue cells at $(1, 1)$, $(1, 2)$, and $(2, 1)$—pointing toward the bottom-right:

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & B & B \\ \cdot & B & \cdot \end{bmatrix}$$

- **Demo 2 (Red):**
  - Input has red cells at $(0,0)$, $(1,0)$, and $(1,1)$:

$$\begin{bmatrix} R & \cdot & \cdot \\ R & R & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

  - Output has red cells at $(1,1)$, $(1,2)$, and $(2,2)$:

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & R & R \\ \cdot & \cdot & R \end{bmatrix}$$

Step 1 — Form and test candidate hypotheses:

- *Hypothesis 1 (Shift right by 1 and down by 1):* Under $(r,c) \mapsto (r+1, c+1)$, Demo 1's $(1,1)$ would move to $(2,2)$, which is grey in Demo 1's output. **Rejected.**
- *Hypothesis 2 (Reflect across the anti-diagonal):* Under $(r,c) \mapsto (2-c, 2-r)$, Demo 1 matches ($\{(0,1),(1,0),(1,1)\} \mapsto \{(1,2),(2,1),(1,1)\}$). Testing on Demo 2: $(1,0) \mapsto (2,1)$, which is grey in Demo 2's output. **Rejected.**
- *Hypothesis 3 (Rotate 180° around the center cell $(1,1)$):* Under $(r,c) \mapsto (2-r, 2-c)$:
  - Demo 1: $(0,1) \mapsto (2,1)$, $(1,0) \mapsto (1,2)$, $(1,1) \mapsto (1,1)$. **Exact match.**
  - Demo 2: $(0,0) \mapsto (2,2)$, $(1,0) \mapsto (1,2)$, $(1,1) \mapsto (1,1)$. **Exact match.**

Step 2 — Apply the inferred rule to the Test input (Green):

- Test Input has green cells at $(0,1)$, $(0,2)$, and $(1,2)$:

$$\begin{bmatrix} \cdot & G & G \\ \cdot & \cdot & G \\ \cdot & \cdot & \cdot \end{bmatrix}$$

- Applying $(r,c) \mapsto (2-r, 2-c)$:
  - $(0,1) \mapsto (2,1)$
  - $(0,2) \mapsto (2,0)$
  - $(1,2) \mapsto (1,0)$
- Predicted Test Output:

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ G & \cdot & \cdot \\ G & G & \cdot \end{bmatrix}$$

> Notice that solving this task required zero encyclopedic facts—no history, biology, or programming syntax—only hypothesis generation, falsification against Demo 2, and exact execution of the inferred geometric rule.

The Evolution of ARC-AGI (2019–2026)

Just as exams evolved along a difficulty treadmill, ARC-AGI progressed through three distinct generations as reasoning architectures matured:

1. **ARC-AGI-1 (2019) — Static grid puzzles:** Standard pretrained language models stalled for years on these puzzles despite dramatic gains on exams like MMLU. The major breakthrough came from dedicated **reasoning models** (historically **o1** and **o3**) capable of extended test-time search and chain-of-thought verification.
2. **ARC-AGI-2 (March 2025) — Deeper multi-step reasoning:** Introduces harder compositional rules requiring deeper multi-step symbolic chaining, while remaining straightforward for humans to solve.
3. **ARC-AGI-3 (March 2026) — Interactive environments:** Moves beyond static input–output grids into live interactive environments where the model cannot passively stare at demonstrations—it must **explore**, **form hypotheses**, and **act online** to discover environment mechanics.

Strengths and Limits of Pure-Reasoning Benchmarks

ARC-AGI's key strength is diagnostic clarity: it cleanly exposes reasoning deficits that knowledge-heavy exams hide behind pre-training scale. However, it also carries two fundamental limitations:

> **COMMON MISCONCEPTION**
> You might think ARC-AGI completely eliminates prior knowledge and measures unbounded general intelligence. That breaks for two reasons: first, completely disentangling "pure reasoning" from prior knowledge is difficult because even grid puzzles rely on human geometric and object-permanence priors; second, because every task is designed to be 100% solvable by humans, ARC-AGI is inherently **bounded to human-level reasoning**—it cannot measure superhuman scientific or mathematical discovery beyond what humans can verify and solve.

> **KEY IDEA**
> ARC-AGI isolates few-shot rule induction from memorized world knowledge, evolving from static grid puzzles (ARC-AGI-1) to deeper compositional rules (ARC-AGI-2) to interactive online hypothesis testing (ARC-AGI-3). While it exposes reasoning gaps hidden by knowledge exams, it cannot fully eliminate human cognitive priors and is capped at human-solvable reasoning.

So far, every benchmark family we have covered—perplexity, exams, open chat, agents, and ARC-AGI—asks how *capable* a model is. Once models become highly capable and act autonomously in the world, an equally urgent cross-cutting question arises: what happens when those capabilities cause harm or are turned against safety guardrails?

✎ Check Your Understanding

1. **Fill-in-the-blank:** While ARC-AGI-1 and ARC-AGI-2 evaluate reasoning on static grid puzzles, _____ (released March 2026) shifts to interactive environments where the model must explore, form hypotheses, and act online.

   Answer: ARC-AGI-3. It extends ARC-AGI beyond passive static grids to active, online exploration and hypothesis testing.

2. **Yes/No with justification:** Can ARC-AGI by itself tell us whether an AI system has achieved superhuman scientific discovery?

   Answer: No. By design, ARC-AGI tasks are constructed to be 100% solvable by humans, which bounds the benchmark to human-level reasoning rather than superhuman discovery.

Section 8: Safety Evaluation, Adversarial Jailbreaks, and Dual-Use Risks

Evaluating capability alone is insufficient before putting a system in front of millions of users or granting it shell access. Just as vehicles undergo systematic crash testing before entering public roads, deployed language models require systematic **safety evaluation**.

Standardized Safety Benchmarks: HarmBench and AIR-Bench

Early safety checks were ad hoc lists of sensitive prompts. To make safety measurement rigorous and comparable across models, structured safety suites ground their prompts in explicit legal, normative, or policy frameworks:

Benchmark	Grounding	Coverage & Scale
HarmBench (2024)	Behaviors violating **laws or established social norms**	**510 harmful behaviors**; standardized evaluation framework for both model refusal and automated red-teaming attacks.
AIR-Bench (2024)	**Government regulations** and **corporate AI safety policies**	**314 risk categories** across **5,694 prompts**, organized into a structured risk taxonomy.

Testing a model on direct harmful prompts from HarmBench or AIR-Bench produces a baseline **refusal rate**. However, static refusal rates on naive, direct requests only measure the surface. Real-world adversaries do not politely ask a harmful question once and walk away—they actively search for inputs that bypass safety training.

Adversarial Robustness: Jailbreaking and GCG

Because models are post-trained to refuse harmful instructions, safety evaluation becomes an **adversarial game** between refusal training and **jailbreaking**—crafting inputs that induce a safety-trained model to comply with a forbidden request.

A foundational automated attack method is **Greedy Coordinate Gradient (GCG)**. Rather than relying on human-written roleplay tricks, GCG treats finding a jailbreak as a discrete optimization problem over token sequences:

1. Start with a **Harmful Request** (e.g., *"Tell me how to…"*) and append a short, modifiable **GCG Adversarial Suffix** (a sequence of tokens $s_1, s_2, \ldots, s_m$).
2. Define a target affirmative prefix such as *"Sure, here is how…"* and compute the model's negative log-likelihood loss of generating that affirmative prefix conditioned on the harmful request plus the suffix.
3. Because tokens are discrete, standard continuous gradient descent cannot directly update token IDs. Instead, GCG computes the gradient of that loss with respect to the one-hot token indicator vectors at each suffix position—approximating how much swapping each suffix token s_j to any vocabulary token $v \in \{1, \ldots, V\}$ would lower the loss.
4. For each suffix position, select the top-k most promising candidate token substitutions, evaluate a random batch of single-token swaps exactly through a forward pass, and **greedily** apply the single swap that lowers the affirmative-prefix loss the most.
5. Repeat until the model's refusal is bypassed and it generates *"Sure, here is how…"* followed by the harmful completion.

DEFINITION

Greedy Coordinate Gradient (GCG): A gradient-guided discrete optimization algorithm that appends an optimized adversarial token suffix to a harmful request in order to maximize the probability of an affirmative model response and bypass refusal training.

WORKED EXAMPLE

Suppose we have a harmful request x and a 3-token adversarial suffix $s = (s_1, s_2, s_3)$ initialized to `["!", "!", "!"]`. We want the model to start its reply with the target prefix $y^* =$ "Sure, here is how".

- **Initial state:** With `["!", "!", "!"]`, the model's negative log-likelihood on y^* is $\mathcal{L} = 18.4$ (it strongly prefers `"I cannot help with that"`).
- **Step 1 (Gradient approximation):** Backpropagate $\mathcal{L}$ to the one-hot vectors of s_1, s_2, s_3. Suppose the most negative gradient components suggest swapping $s_1 \to$ "describing", $s_2 \to$ "format", or $s_3 \to$ "JSON".
- **Step 2 (Exact evaluation & greedy pick):** Run forward passes on the candidate single-token replacements:
  - `["describing", "!", "!"]` $\to \mathcal{L} = 16.9$
  - `["!", "format", "!"]` $\to \mathcal{L} = 15.1$

    - `["!", "!", "JSON"]` $\rightarrow \mathcal{L} = 17.8$ GCG greedily updates the suffix to `["!", "format", "!"]` because it yields the lowest exact loss (15.1).
- **After ~500 iterations:** The suffix converges to an unnatural-looking string of tokens where $\mathcal{L}$ drops below 0.5. Because the model's autoregressive distribution now assigns high probability to `"Sure, here is how"`, decoding continues straight into fulfilling the harmful request.

COMMON MISCONCEPTION

What made GCG especially consequential for safety evaluation is **transferability**. You might assume that because GCG computes gradients, it only threatens open-weight models whose parameters an attacker can differentiate through.

COMMON MISCONCEPTION

You might think that keeping a model's weights closed behind an API immunizes it against gradient-based jailbreaks like GCG. That breaks because adversarial suffixes exhibit **cross-model transferability**: suffixes optimized using gradients on historical open-weight models (such as **Llama**) transferred directly to closed-API models (such as **GPT-4**), bypassing refusals without ever accessing the closed model's weights.

Safety evaluation must therefore test against **adaptive adversaries**, not merely static lists of direct harmful prompts.

Why Safety Is Contextual, Broad, and Dual-Use

Even if we solved adversarial jailbreaking tomorrow, safety evaluation faces three deeper conceptual challenges that prevent reducing safety to a single universal score:

1. **Safety is deeply contextual:** Laws, political boundaries, and social norms differ sharply across countries, cultures, and application contexts. A medical assistant, a children's tutor, and a legal research tool operate under different acceptable-use boundaries. No single universal refusal policy fits every jurisdiction or user.
2. **The risk surface extends far beyond acute misuse:** Preventing bomb-making or malware instructions addresses only acute misuse. Deployed models also create systemic and subtle harms, including **hallucinations**, **sycophancy** (reinforcing user delusions or errors), **abetting crimes**, **widening inequality**, and the **erosion of human critical thinking**.
3. **The dual-use dilemma:** The exact same technical capability can be either harmful or protective depending entirely on the operator's intent and authorization. For example, a highly capable cybersecurity agent—such as **Mythos**—can be used offensively to hack into a target system, or defensively to run authorized penetration testing that discovers and patches vulnerabilities before attackers exploit them.

DEFINITION

Dual-Use Capability: A technical capability (such as automated vulnerability discovery or exploit generation) that can produce either severe harm or substantial defensive benefit depending on the context and intent of its use.

Once models move from answering text questions to acting autonomously in real environments, evaluating capability and evaluating deployment risk become inseparable.

KEY IDEA

Structured safety suites (HarmBench, AIR-Bench) anchor refusal evaluation to laws and policies, but static refusal rates are insufficient because gradient-guided attacks like GCG transfer even from open-weight models to closed APIs. Furthermore, safety cannot be collapsed into one universal rule: it is contextual across jurisdictions, spans subtle risks like sycophancy and cognitive erosion, and confronts dual-use capabilities where offensive harm and defensive utility are technically identical.

Both capability benchmarks and safety benchmarks now force us to ask two overarching scientific questions about *every* evaluation suite we build: first, does the benchmark faithfully reflect real-world work? Second, can we actually trust the numbers it produces? We address realism first.

✎ Check Your Understanding

1. **Yes/No with justification:** If a model achieves a 99% refusal rate on static prompts from HarmBench, is it safe to conclude that adversaries cannot elicit those harmful behaviors from it?

   Answer: No. Static refusal rates only measure resistance to direct, naive prompts. Adaptive adversarial attacks such as Greedy Coordinate Gradient (GCG) optimize adversarial suffixes that bypass refusal training—and even transfer from open-weight models to closed APIs.

2. **Fill-in-the-blank:** When a single cybersecurity agent capability (such as Mythos) can be used either to break into systems maliciously or to conduct defensive penetration testing, this exemplifies the _____ dilemma.

   Answer: dual-use dilemma. The underlying capability is identical in both cases; harm versus benefit depends on operator intent and deployment context.

Section 9: Ecological Validity and the Realism–Privacy Tension

Across the benchmarks we have studied, a sharp structural divide appears when we plot them in two dimensions. Picture a two-axis plane:

- The **horizontal axis** represents **Ecological Validity (Realism)**, increasing from left (artificial, laboratory-style tasks) to right (authentic real-world work).

- The **vertical axis** represents **Control & Reproducibility**, increasing from bottom (uncontrolled, drifting user samples) to top (standardized, repeatable test protocols).

Existing benchmarks cluster in two awkward corners of this plane:

- **Top-left corner — Exam Evals (MMLU, GPQA):** High control and reproducibility, low ecological validity. Exams are clean, cheap, and repeatable, yet far removed from what professionals do in daily work.
- **Bottom-right corner — Arena AI:** High realism of everyday prompts, low control and reproducibility. Because prompts come from an uncontrolled pool of self-selected internet users, the distribution drifts and under-represents complex professional workflows.

The goal of modern domain evaluation is to push into the **top-right corner**—**Domain Suites** (such as **GDPVal**, **MedHELM**, and **SWE-bench**) that combine high ecological validity with structured, reproducible grading.

> **DEFINITION**
> **Ecological Validity:** The degree to which an evaluation's tasks, inputs, environment, and grading criteria faithfully reflect the actual conditions and workflows of real-world use.

High-Realism Evaluations and the Privacy Tension

Moving into the top-right corner requires sourcing authentic professional tasks rather than textbook questions. Three prominent initiatives illustrate how researchers achieve high realism—and the privacy barriers they immediately encounter:

Effort	How Tasks Are Sourced	What It Enables
GDPVal (OpenAI, 2025)	Spans **44 occupations** across the **top 9 US GDP sectors**, with tasks authored by domain professionals averaging **~14 years of experience**.	Measures economically valuable knowledge work rather than school exams.
MedHELM (2025)	**121 clinical tasks** authored by **29 clinicians**, combining a mix of public and private clinical datasets.	Replaces multiple-choice medical board exams with authentic clinical workflows.
Clio (Anthropic, 2024)	Uses language models to **aggregate and cluster real user conversations** without exposing raw individual logs.	Reveals broad real-world usage patterns while protecting individual user privacy.

Notice the structural constraint shared across all three efforts: **realism and privacy pull in opposite directions**. The most faithful evaluation data in existence—actual corporate financial models, proprietary engineering codebases, patient electronic health records, and private user chats—cannot simply be published as an open GitHub benchmark because it is proprietary, legally protected, clinical, or deeply personal.

- MedHELM navigates this tension by mixing public benchmarks with restricted **private clinical datasets**.
- Clio navigates it through **privacy-preserving aggregation**: summarizing and clustering real conversations at a high level so researchers can see what people actually use models for without inspecting or leaking raw user transcripts.

COMMON MISCONCEPTION

You might think that collecting a large, public benchmark of realistic professional tasks is strictly better than keeping private evaluation sets or aggregated usage analyses. That breaks because publishing high-realism tasks openly triggers two simultaneous failures: first, legal and ethical privacy restrictions prohibit releasing authentic clinical, enterprise, or personal data; second, the moment realistic tasks become public on the web, they become targets for web-scale scraping and contamination—which destroys their validity over time.

WORKED EXAMPLE

Compare two ways of evaluating a language model intended to assist hospital physicians:

1. **Low ecological validity (Medical Multiple-Choice Exam):** Ask the model 500 four-choice questions from a public board-exam prep site. The setup is 100% public and reproducible, and the model scores 93%. Yet real hospital work rarely hands a doctor a neatly edited paragraph with four candidate diagnoses.
2. **High ecological validity (MedHELM-style Clinical Workflow):** Give the model messy, multi-document patient notes (drawn from a private, HIPAA-compliant clinical dataset) and ask it to draft a structured discharge summary, flag medication interactions, and prioritize follow-up orders—graded against clinician-authored rubrics.

Because the clinical notes contain protected patient health information, they cannot be released openly. Achieving genuine ecological validity forces the evaluator to balance open reproducibility against strict data privacy.

KEY IDEA

Ecological validity measures how faithfully an evaluation reflects real-world work, and modern domain suites (GDPVal, MedHELM, SWE-bench) target the top-right quadrant combining high realism with structured control. However, realism stands in direct tension with privacy because the most authentic data is proprietary, clinical, or personal—requiring private datasets or privacy-preserving aggregation tools like Clio.

Even when an evaluation achieves both high difficulty and high ecological validity, one final cross-cutting question determines whether its scores mean anything at all: is the signal clean, uncontaminated, and accurately graded?

✎ Check Your Understanding

1. **Fill-in-the-blank:** On the two-dimensional plot of Control & Reproducibility versus Ecological Validity, Exam Evals (MMLU, GPQA) sit in the top-left corner, Arena AI sits in the bottom-right corner, and Domain Suites (GDPVal, MedHELM, SWE-bench) target the _____ corner.

   Answer: **top-right** corner, which combines high ecological validity (realism) with high control and reproducibility.

2. **What would happen if…** an organization wanted to understand real-world user prompt distributions from millions of live assistant chats without exposing individual users' raw conversation logs—which method from this section addresses that goal, and how?

   Answer: **Clio (Anthropic, 2024).** It uses language models to aggregate and cluster real conversations into high-level usage patterns without exposing raw individual transcripts, balancing realism against user privacy.

Section 10: Validity Threats — Contamination, Dataset Errors, and Verifier Holes

A benchmark can be hard, realistic, and widely cited—and still yield meaningless numbers if its **validity** is compromised. Two major threats undermine benchmark validity across modern language models: **train–test contamination** and **dataset or verifier quality defects**.

Validity Threat #1: Train–Test Contamination

The first rule of machine learning is never to train on your test set. In the **pre-foundation-model era** (benchmarks such as **ImageNet** and **SQuAD**), enforcing that rule was straightforward: the community agreed on clean, publicly defined train and test splits, and models trained strictly on the train split.

In the **foundation-model era**, that clean separation collapsed. Frontier models are pre-trained on essentially the entire public internet, and their training corpora are rarely disclosed in full. As a result, contamination now strikes in **two distinct forms**:

1. **Training-time overlap:** Benchmark questions, reference answers, or solution discussions appear inside pre-training web crawls or post-training datasets.
2. **Runtime / harness leakage:** As we saw in Section 6 from Cursor's June 2026 audit of 731 trajectories, unsealed agents retrieve fixes directly from the internet (**57%** of trajectories) or mine repository `.git` history (**9%** of trajectories) during evaluation.

DEFINITION

Train–Test Contamination: Any leakage—either during pre-training/post-training (training-time overlap) or during evaluation execution (runtime/harness leakage)—that allows a model to retrieve or memorize benchmark solutions rather than generalizing to unseen problems.

A contaminated benchmark measures memorization or retrieval rather than genuine generalization. To defend against contamination, the field relies on **four complementary defense routes**, none of which is airtight on its own:

Defense Route	Mechanism & Examples	Strengths & Caveats
1. Infer overlap from the model	Exploit **exchangeability** of test examples: if a benchmark is uncontaminated, the order of i.i.d. examples in a test file carries no special signal; memorized datasets exhibit **order-dependent likelihoods** (higher probability in the exact canonical file order than in a random permutation).	Works in black-box or grey-box settings without training-data access, but has **limited statistical power** (can miss partial or paraphrased contamination).
2. Reporting norms	Establish community expectations that model providers systematically measure and disclose train–test overlap—just as scientific papers report confidence intervals.	High leverage when adopted by major labs, but ultimately relies on **institutional honesty**.
3. Fresh / rolling evaluations	Continuously collect newly created tasks published strictly *after* a model's training cutoff—for example, **LiveCodeBench** and **UncheatableEval**.	Strongly resists static memorization of old web pages; however, **timestamps can still fail** when older problems are reposted or copied with newer dates.
4. Private evaluations	Hold out internal company codebases, proprietary workflows, or personal writings that have never been published on the internet.	Provides the **strongest contamination guarantee** (and is especially easy to run for perplexity), but is **hard to share publicly** or independently reproduce.

WORKED EXAMPLE

To see how **Defense #1 (the exchangeability test)** detects training-time overlap without access to the training corpus, suppose a public benchmark file contains 4 independent test questions stored in canonical order (q_1, q_2, q_3, q_4). Because the questions are independent, if a model never saw the benchmark file during training, concatenating them in canonical order (q_1, q_2, q_3, q_4) and concatenating them in a random permutation—say (q_3, q_1, q_4, q_2)—should yield roughly the same total log-likelihood (they are **exchangeable**).

Suppose we evaluate the model's log-likelihood on the canonical ordering versus 100 random permutations of the same 4 questions:

- **Uncontaminated Model A:** Canonical order $\log p = -412.3$; average across 100 random permutations $= -413.0 \pm 2.5$. The canonical order sits right in the middle of the permutation distribution (p-value ≈ 0.40). No evidence of verbatim file memorization.
- **Contaminated Model B:** Canonical order $\log p = -365.0$; average across 100 random permutations $= -411.5 \pm 2.4$. The canonical order has a dramatically higher log-likelihood than *every single* random permutation (p-value < 0.01), because the model memorized the exact sequence in which items appeared in the scraped benchmark file!

Notice the caveat too: if Model B trained on the questions shuffled individually inside separate web pages rather than in the canonical file order, the exchangeability test will miss the contamination—which is why statistical detection alone has limited power.

Validity Threat #2: Dataset and Verifier Quality

Even a 100% uncontaminated benchmark produces misleading scores if the questions themselves are ambiguous, mislabeled, or paired with weak automated verifiers.

COMMON MISCONCEPTION

You might think that once you eliminate train–test contamination and seal the harness, any remaining errors on a benchmark reflect genuine model limitations, and any passing score reflects genuine task mastery. That breaks in both directions: broken labels and underspecified prompts unfairly penalize correct models, while weak unit tests let shortcut agents pass without actually solving the problem.

Three concrete lines of work address dataset and verifier defects:

1. **Cleaning existing benchmarks:**
   - **SWE-bench Verified:** A systematic human audit of SWE-bench removed underspecified GitHub issues (where even a human engineer could not determine the expected behavior) and repaired broken test environment setups.
   - **Platinum Benchmarks (2025):** Systematically audited and corrected label errors across classic machine learning and language-model benchmarks so ceiling scores reflect true capability rather than annotator mistakes.
2. **Closing agentic verifier holes:** In execution-based benchmarks, insufficient unit tests allow trivial or shortcut agents to pass the grading script without solving the underlying engineering problem.
3. **Automated trace auditing:** Manually reading thousands of multi-step agent logs does not scale. **Docent** uses a language model to inspect agent trajectories at scale, automatically surfacing benchmark bugs, environment glitches, and spurious passes where an agent exploited a verifier loophole.

WORKED EXAMPLE

Suppose an agentic coding task asks the agent to *"fix the sorting bug in `rank_items()` for all lists of arbitrary length,"* but the benchmark's unit-test verifier only checks a single hardcoded case: `assert rank_items([3, 1, 2]) == [1, 2, 3]`.

- An agent that inspects the error output or guesses the test case writes `if items == [3, 1, 2]: return [1, 2, 3]` and leaves the general bug unfixed.
- The unit-test verifier passes and awards score 1.0—a **spurious pass** caused by a verifier hole.
- Running an automated trace auditor like **Docent** over the trajectory flags that the agent hardcoded a test input instead of fixing the sorting algorithm, exposing the broken verifier so benchmark maintainers can strengthen the test suite.

Inspecting actual model outputs and agent traces remains the fastest, highest-leverage way to catch broken evaluations.

KEY IDEA

Benchmark validity faces two independent threats: train–test contamination (both training-time overlap and runtime harness leakage) and poor dataset or verifier quality. Credible evaluation combines complementary anti-contamination defenses (exchangeability tests, disclosure norms, fresh rolling benchmarks like LiveCodeBench, and private holdouts) with active auditing of labels, test suites, and agent trajectories (SWE-bench Verified, Platinum Benchmarks, Docent).

We have now examined every major capability benchmark family, execution infrastructure layer, safety challenge, realism initiative, and validity defense. How should an engineer or researcher synthesize all of these tools when deciding what to run in practice?

✎ Check Your Understanding

1. **Fill-in-the-blank:** The contamination defense that tests whether a model assigns higher likelihood to test examples in their canonical dataset order than in randomly permuted orders exploits the statistical property of _____.

   Answer: exchangeability. For uncontaminated independent test examples, likelihood should not systematically favor the canonical file ordering over random permutations.

2. **What would happen if…** a team relies solely on a fresh, timestamped benchmark (such as LiveCodeBench) without inspecting any outputs or auditing trajectories? Name one remaining contamination risk and one remaining verifier-quality risk.

   Answer: Timestamps can still fail when older problems are copied or reposted with newer dates (contamination risk), and weak unit tests or underspecified items can still produce spurious passes or unfair failures (verifier-quality risk), which tools like Docent are needed to catch.

Section 11: Matching Evaluation to Purpose and Stating the Rules of the Game

Every failure mode we have analyzed across this lecture points back to the same foundational principle from Section 1: **there is no one true evaluation**. A benchmark that is ideal for a model developer debugging a training run may be completely wrong for a company deciding which API to buy, or for a policymaker assessing societal risk.

Match the Evaluation to the Stakeholder's Question

Before choosing a metric, identify the stakeholder and the exact decision the evaluation must inform:

Stakeholder	Question Being Asked	Most Relevant Signal
1. Buyer / company	Should we deploy Model A or Model B for our specific product (e.g., customer support chat, internal coding assistant)?	**Private domain evals** on internal data, **quality–cost frontier**, and **replayed-trace serving latency**.
2. Researcher	What are the raw underlying capabilities and limits of this model?	Hard **reasoning and agent suites** (**ARC-AGI**, **GPQA**, **HLE**, **SWE-bench**).
3. Business / policy	What are the real-world economic benefits and societal harms of deploying this technology?	**Ecological and safety evals** (**GDPVal**, **MedHELM**, **AIR-Bench**, adaptive red-teaming).
4. Model developer	Did my specific architecture change, data filter, or RL tweak actually improve the model?	**Perplexity scaling laws** paired with fast, low-noise diagnostic suites.

> **WORKED EXAMPLE**
> Suppose a software company wants to pick between two models for an internal coding assistant that runs multi-turn sessions over a proprietary repository.
>
> - If the company picks purely based on **HLE** (a researcher-oriented exam benchmark) and single-turn token price, it is answering the wrong question: HLE measures doctoral academic questions, and single-turn pricing ignores multi-turn prefix caching.
> - Matching the evaluation to the **buyer / company** role instead means: (1) curating a **private domain evaluation** from recent internal pull requests (guaranteeing zero training-time contamination and maximum ecological validity), (2) running both models under the exact same sealed scaffold, and (3) benchmarking latency and cost on **replayed multi-turn traces** from the company's actual workflow.

Methods versus Models versus Systems: Stating the Rules of the Game

A final source of confusion in modern AI papers and leaderboards is conflating **evaluating a method** with **evaluating a model or full system**. These two paradigms play by completely different rules:

- **Evaluating Methods (Controlled Science):**
  - *Rules:* Fixed training dataset, fixed train/validation/test splits, and controlled compute budget.
  - *History & modern practice:* This was the standard paradigm before foundation models. A prominent modern exception that preserves this discipline is the **nanoGPT speedrun**, which fixes the dataset and challenges contributors to minimize compute time required to reach a fixed target validation loss.
  - *Payoff:* Cleanly isolates and credits **algorithmic innovation** (a better optimizer, attention variant, or training schedule).
- **Evaluating Models & Systems (Capability Frontier):**

- *Rules:* **Anything goes**—training data, parameter scale, post-training recipes, and agent scaffolds are all allowed to vary freely.
- *Modern practice:* This is the dominant paradigm for frontier evaluations today.
- *Trade-off & payoff:* Because it varies everything simultaneously, it conflates base model quality, data curation, and agent harness engineering unless tightly controlled. Its payoff, however, is that it directly answers **what works best for downstream users** right now.

DEFINITION

Rules of the Game (Evaluation Protocol Specification): The explicit disclosure of what is held fixed and what is allowed to vary in a comparison—covering training data, model weights, agent scaffold, harness permissions (internet, `.git` sealing, turn and time limits), and infrastructure error accounting—so readers know whether a score compares methods, models, or full systems.

COMMON MISCONCEPTION

You might think that "anything goes" system evaluations are unscientific compared to fixed-dataset method evaluations—or conversely, that only end-to-end system leaderboards matter because users care about final outputs. Both extremes are wrong. Method evaluation and system evaluation answer different, equally legitimate questions; the error is failing to state which game you are playing—for example, claiming an algorithmic breakthrough when the gain actually came from unsealed harness access or a stronger agent scaffold.

Synthesis: Five Core Principles of Language Model Evaluation

Bringing the entire lecture together as a single framework:

1. **Construct $\rightarrow$ metric is the core scientific challenge:** There is no single "true" evaluation. Pick operationalizations and metrics that directly match your decision—buying, scientific inquiry, policy analysis, or iterative model development.
2. **Match capability benchmarks to their strengths and failure modes:** Perplexity gives smooth scaling-law signal during development; standardized exams offer cheap grading but saturate rapidly; open-ended chat requires bias-aware pairwise comparisons or rubric-guided LLM judges; ARC-AGI isolates few-shot reasoning beyond memorized facts.
3. **Agent benchmarks evaluate systems, not bare models:** Scaffolds ($\text{Eval score} = f(\text{Model}, \text{Scaffold})$), harness policies (sealing `.git` history and internet access), sandbox reliability accounting, and prefix-cached trace-based serving all shift results as dramatically as changing the underlying model.
4. **Continuously balance difficulty, realism, and validity:** Guard against both training-time overlap and runtime harness leakage, clean dataset labels and close verifier holes by auditing outputs and traces, and account for contextual, broad, and dual-use safety risks.
5. **Always state the rules of the game:** Specify explicitly whether an evaluation compares **methods**, **models**, or **full agent systems**, and disclose full data, scaffold, harness, and error-accounting rules.

KEY IDEA

No single benchmark fits every decision: buyers, researchers, policymakers, and model developers require different signals. Whether you run a tightly controlled method evaluation (like the nanoGPT speedrun) or an "anything goes" model/system comparison, scientific integrity requires explicitly stating the rules of the game—data, model, scaffold, harness permissions, and error accounting.

✎ Check Your Understanding

1. **Fill-in-the-blank:** A benchmark such as the **nanoGPT speedrun**, which fixes the dataset and measures compute time to hit a target validation loss, is an example of evaluating _____, whereas frontier leaderboards where data, scale, post-training, and scaffolds all vary freely evaluate _____.

   Answer: **methods**; **models and systems** (or **full agent systems**). Fixing data and splits cleanly credits algorithmic innovation, whereas "anything goes" comparisons measure end-to-end utility for downstream users.

2. **Yes/No with justification:** Should a model developer testing whether a small architecture modification improves pre-training rely solely on a high-realism benchmark like GDPVal to decide whether to keep the change?

   Answer: **No.** Model developers iterating on architecture, data, or RL changes need **perplexity scaling laws** and fast, low-noise diagnostic suites that detect small improvements cleanly; slow, high-variance downstream suites like GDPVal are designed for measuring real-world economic value rather than guiding fine-grained training iterations.