Evaluation and Benchmarking

Robert Minneker

2026-03-03

Sources & Objectives

Key References: Liang et al. (2023) — HELM; Zheng et al. (2023) — Chatbot Arena; Sainz et al. (2023) — Contamination

By the end of this lecture, you will be able to:

1. Explain why LLM evaluation is fundamentally harder than classifier evaluation
2. Select appropriate metrics for different NLP tasks and explain their limitations
3. Detect and mitigate data contamination in benchmark evaluations
4. Design human evaluation protocols and identify biases in LLM-as-a-judge approaches
5. Apply red-teaming methodology to probe model weaknesses

Would you ship this summarizer?

Your team built a news summarization system. Your PM asks: **“Is it ready to ship?”**

Here’s what it produces for the article *“Talks in Geneva ended without agreement on climate funding targets”*:

Output 1:

"Geneva climate finance negotiations concluded without a deal."

Output 2:

"Talks in Geneva ended with agreement on climate funding targets."

Output 3:

"Talks in Geneva ended without agreement."

Vote: Ship / Don’t ship / Need more information?

- **Follow-up:** What *evidence* would you need to make this decision confidently? That’s what this lecture is about.

Course roadmap

BUILDING

Inference-Time Control ✓
Prompting, decoding, self-consistency

Training-Time Control ✓
SFT/PEFT, continued pretraining, pref. tuning

System-Time Augmentation ✓
RAG, tools, agents

UNDERSTANDING & GOVERNING

Understanding ✓
Interpretability, causal methods

Measuring TODAY
Evaluation, contamination, protocols

Governing & Shipping
Safety engineering, deployment constraints

Part 1: The Evaluation Challenge

LLM evaluation is fundamentally harder than classifier evaluation

Running example: Evaluating a news summarization system — how do you score “Talks in Geneva ended without agreement on climate funding targets”?

Classifiers

Discrete labels, well-defined ground truth, established metrics (F1, accuracy)

LLMs

Open-ended text, multiple valid outputs, metrics must assess fluency, factuality, coherence simultaneously

Goodhart's Law: "When a measure becomes a target, it ceases to be a good measure."

- e.g. BLEU-optimized systems produce disfluent text; RLHF models learn to exploit reward models rather than genuinely satisfy users.
- **Quick question:** How would *you* game ROUGE for summarization?
  - One trick: copy the most salient phrases from the source verbatim and pad with filler. ROUGE rewards the overlap; readers get garbage.

⚠ **Warning**

Models can “game” any single metric — multi-dimensional evaluation is essential. The evaluation challenge motivates everything else in this lecture.

Match the metric to the task — and know each metric's failure mode

Metric	Task	Formula (intuition)	Key Limitation
Perplexity	Language modeling	$2^{-\frac{1}{N}\sum \log_2 p(w_i)}$	Rewards fluency, ignores factuality
BLEU	Translation, summarization	n-gram precision vs. reference	Ignores semantics; penalizes valid paraphrases
ROUGE-L	Summarization	Longest common subsequence	Rewards surface overlap, misses abstraction
BERTScore	Open-ended generation	Embedding similarity	Better with paraphrases; still imperfect
Human Judgment	Any	Likert scales, pairwise comparison	Expensive, slow, annotator disagreement

You be the evaluator: Rank these summaries

Reference: “Talks in Geneva ended without agreement on climate funding targets.”

Candidate A

"Geneva climate finance negotiations concluded without a deal."

Candidate B

"Talks in Geneva ended *with* agreement on climate funding targets."

Candidate C

"Talks in Geneva ended without agreement."

Vote now: Rank these best → worst. Which would you ship?

The metrics disagree with you

Candidate	BLEU-4	ROUGE-L	BERTScore	Actual Quality
A (good paraphrase)	Low	Low	High	**Best**
B (factually wrong!)	**Very High**	**Very High**	High	**Worst**
C (extractive/incomplete)	Medium	Medium	Medium	**Incomplete**

⚠ **Warning**

BLEU and ROUGE rank the **factually wrong** summary (B) highest because they only measure surface overlap. BERTScore handles paraphrases better but still can't catch the negation flip. **Which failure mode happened here?** Surface overlap gaming + negation blindness.

ⓘ **Activity: Design a better metric (60 sec)**

Challenge: Propose a metric or test that would catch the negation flip in Candidate B ("without" → "with" agreement).

Think about: What would your metric actually *compute*? What input does it need beyond the candidate text?

Why designing metrics is hard

Your proposals likely fall into one of these categories — and each has a catch:

"Use NLI to check entailment"

Good idea — but NLI models themselves have error rates, especially on subtle negation

"Compare against the source document, not just a reference"

This is the key insight behind **factual consistency** metrics — but requires access to the source

"Use an LLM to judge correctness"

We're back to LLM-as-a-judge — which has its own biases (as we'll see in Part 3)

- **Bottom line:** Semantic and factual evaluation requires *understanding*, not just surface matching — and that's fundamentally hard to automate

Individual metrics aren't enough → systematic benchmark suites

1. **Single metrics are gameable** — as we just saw, BLEU rewards factually wrong answers
2. **Combining metrics helps** but requires principled selection — which metrics, on which tasks, with what weighting?
3. **This is exactly what benchmark suites solve** — standardized, multi-metric, multi-task evaluation

Part 2: Benchmarks and Data Contamination

Benchmarks should test real capabilities — but their design matters enormously

Benchmark	What It Tests	Design Innovation
MMLU	57-subject knowledge (STEM, humanities, social sciences)	Multiple-choice; easy to score
HELM	Multi-metric holistic evaluation	Evaluates accuracy + calibration + fairness + toxicity simultaneously
BIG-Bench	204 diverse tasks from 400+ researchers	Crowdsourced; tests emergent abilities
Chatbot Arena	Open-ended human preference	Head-to-head blind comparisons; Elo ratings

Well-designed benchmarks need: Breadth across tasks, held-out test sets, regular refreshing, resistance to gaming

- No single benchmark captures everything — always evaluate on multiple benchmarks + task-specific tests

Data contamination inflates benchmark scores by leaking test data into training corpora

The problem: LLM pretraining corpora are massive (trillions of tokens from the internet). Benchmark datasets are also on the internet. ⇒ Test data leaks into training data.

Detection Method	How It Works
n-gram overlap	Search for exact test set sequences in training data
Canary strings	Plant unique tokens in benchmarks; test if models generate them
Membership inference	Test if model assigns suspiciously high probability to test examples
Temporal splits	Only use test data created *after* the model's training cutoff

- **GPT-4 on Codeforces:** Performance drops sharply on problems published *after* training cutoff — strong evidence of contamination in the training data
- Contamination means that published benchmark scores are often **upper bounds** on actual capability

ⓘ **Activity: The contamination verdict (2 min)**

Scenario: Model X scores 92% on MMLU (state-of-the-art!) but drops to 78% on questions published *after* its training cutoff.

Step 1 — Vote (30 sec): What do you conclude?

- **(A)** The model is genuinely capable — the new questions are just harder
- **(B)** Contamination inflated the score — 78% is the real capability
- **(C)** Can't tell — need more evidence (membership inference, canary strings)

Step 2 — Pair justification (60 sec): Defend your choice to a neighbor.

Step 3 — Debrief: What additional evidence would change your answer?

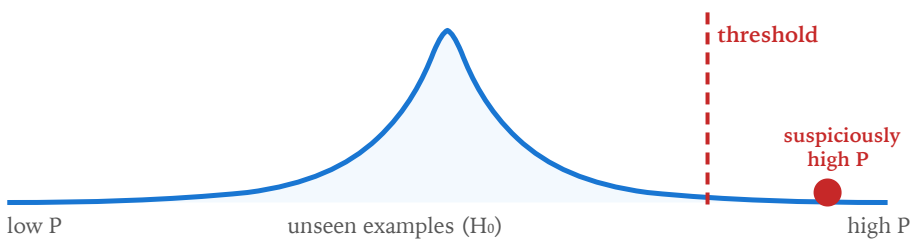
Contamination detection is hypothesis testing

H_0 (null): The test example was *not* in the training data

Test statistic: Model's log-probability (or perplexity) on the test example

Null distribution: Log-probabilities on held-out examples known to be unseen

Decision rule: If P(example) is in the far right tail → reject H_0 → likely contaminated



This same logic applies to **canary strings** — if the model generates a planted token sequence, that's an extreme-probability event under H_0.

Quick check: Spot the contamination

A model's perplexity on 6 benchmark examples (lower = more "familiar"):

Example	Perplexity
"What is the capital of France?"	**3.2**
"Calculate $\int_0^1 x^2 \sin(\pi x)\, dx$"	**45.7**
"According to Hendrycks (2021), Table 3, row 5..."	**1.8**
"Name three causes of the 1997 Asian financial crisis"	**12.4**

Vote: Which example(s) look contaminated? Why?

- Example 3's perplexity (1.8) on a highly specific table reference is suspiciously low — the model has likely memorized the exact benchmark question.

Part 3: Human Evaluation and LLM-as-a-Judge

Raw agreement is misleading — why we need κ

Two annotators rate 100 summaries as “Good” or “Bad”:

	B: Good	B: Bad
A: Good	85	5
A: Bad	5	5

- **Raw agreement:** (85 + 5) / 100 = **90%** — looks great!
- **But:** If 90% of summaries are “Good,” chance agreement alone ≈ **82%**
- **Cohen’s κ:** $\kappa = \frac{p_o - p_e}{1 - p_e} = \frac{0.90 - 0.82}{1 - 0.82} \approx 0.44$ — only moderate!

> ⚠ **Warning**
>
> κ corrects for base-rate imbalance. High raw agreement can mask the fact that annotators mostly agree by default, not by judgment.

Human evaluation is the gold standard — but requires careful protocol design

Likert Scale

"Rate fluency 1-5"

Quantitative scores

Anchoring bias, scale drift

Pairwise Comparison

"Which summary is better?"

More reliable; enables rankings

$O(n^2)$ pairs for n models

Best-Worst Scaling

"Pick best and worst from set"

Information-rich per judgment

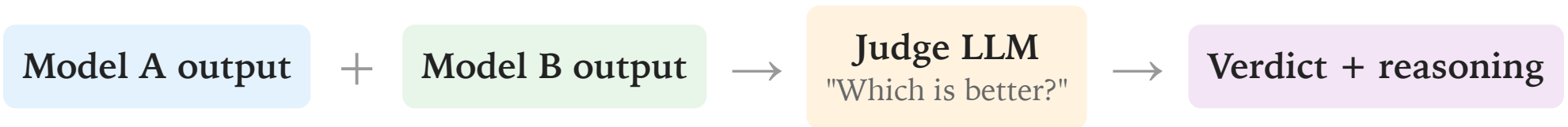
Requires diverse item sets

Inter-annotator agreement measures protocol quality — low agreement means the task is ambiguous, not that annotators are bad:

- **Cohen's** κ: chance-corrected agreement between 2 raters. $\kappa > 0.6$ = substantial agreement
- For open-ended generation tasks, even $\kappa = 0.4$–0.6 is typical

LLM-as-a-judge scales human-like evaluation but introduces systematic biases

The idea: Replace human annotators with an LLM prompted to evaluate outputs.



Known biases in LLM judges:

Position Bias
Prefers whichever response appears first

Verbosity Bias
Longer responses rated higher

Self-Enhancement
GPT-4 prefers GPT-4 outputs

Style Over Substance
Rewards formatting over correctness

- **Mitigations:** Randomize presentation order; require chain-of-thought justification; use multiple judge models; calibrate against human labels

Position bias in action

Prompt to judge model: “Which response better answers the question ‘What causes tides?’”

Trial	Order Shown	Judge's Verdict
1	Response A first, B second	"A is better"
2	Response B first, A second	"B is better" (same response, now shown first!)

A first, B second → "A wins" ⇄ B first, A second → "B wins"

The judge flipped its preference purely based on presentation order — the **same response** wins when shown first.

> ⚠ **Warning**
>
> **Mitigation:** Run every comparison twice (AB and BA order). Only count as a win if the same response wins both times; otherwise mark as a tie.

LLM-as-a-judge protocol checklist

- ☐ **Randomize presentation order** — run every pair as AB *and* BA
- ☐ **Require chain-of-thought** — judge must explain *before* giving a verdict
- ☐ **Allow ties** — forcing a winner inflates false distinctions
- ☐ **Use multiple judge models** — no single judge should be trusted alone
- ☐ **Calibrate against human labels** — measure judge-human agreement on a sample

Chatbot Arena: Crowdsourced head-to-head evaluation at scale

How it works:

1. Users submit prompts to **two anonymous models** simultaneously
2. Users vote for the better response (or tie)
3. Votes are aggregated into **Elo ratings** (same system as chess)

> **Why Chatbot Arena works:**
>
> - **Diverse, real user queries** — not synthetic benchmarks
> - **Anonymous:** users don't know which model they're rating
> - **Pairwise:** "which is better?" is easier and more reliable than rating on a scale
> - **Contamination-resistant:** prompts aren't in any training set

- Chatbot Arena has become the most trusted open leaderboard for LLM capabilities
- **Limitation:** Skewed toward chat/instruction tasks; doesn’t test specialized domains well

> Concept Check
>
> Let’s see Arena in action!

Part 4: Red-Teaming and Reproducibility

Red-teaming probes model weaknesses through structured adversarial evaluation

Red-teaming = systematically attempting to elicit failures, unsafe outputs, or unexpected behaviors.

Safety

Attempt to generate harmful, biased, or illegal content through adversarial prompts

Capability

Find edge cases where the model fails confidently — incorrect reasoning, hallucinated citations

Robustness

Test with typos, slang, code-switching, adversarial formatting

Red-teaming is not just "trying to break the model" — it follows a structured methodology:

1. Define **threat model** (who is the adversary? what are they trying to achieve?)
2. Design **attack vectors** (prompt injection, jailbreaking, social engineering)
3. Classify **failure severity** (cosmetic, misleading, harmful, dangerous)
4. Feed findings back into safety training

Rigorous evaluation requires statistical discipline

Report confidence intervals, not just point estimates

Bootstrap over test examples: does "87.3% vs 86.1%" actually matter?

Prompt sensitivity analysis

Report results across multiple prompt templates — not just the best one

Fix all random seeds and document API versions

API models change silently; results from March ≠ results from June

Disclose costs and environmental impact

A benchmark run costing $10K and emitting 500kg CO_2 has different implications than one costing $5

- **"Not reproducible"** is the default state for LLM evaluations — active effort is required to change this

Quick check: Does the prompt matter?

Two prompt templates for the **same** summarization task on the **same** test set:

Prompt A

```
"Summarize the following article in one sentence."
```

Prompt B

```
"You are an expert journalist. Write a one-sentence summary of this article."
```

Activity: Design your evaluation strategy (3 min)

Scenario: You're shipping the news summarization system from today's running example.

Step 1 — Vote (30 sec): Which evaluation approach would you prioritize first?

- **(A)** Automated metrics (BLEU + BERTScore) on a held-out test set
- **(B)** Pairwise human evaluation with 3 annotators
- **(C)** LLM-as-a-judge with GPT-4 + debiasing protocol

Step 2 — Pair justification (60 sec): Turn to a neighbor. Defend your choice.

Step 3 — Debrief: What did each choice trade off? (Cost vs. reliability vs. speed)

A Simplified Evaluation Playbook

If you're shipping...	Then do this
Any LLM system	Use multiple metrics — never trust a single number. Report confidence intervals.
A generation task	BERTScore + human pairwise eval. BLEU/ROUGE only as sanity checks.
Against a benchmark	Check for contamination (temporal splits, membership inference). Treat scores as upper bounds.
LLM-as-a-judge eval	Randomize order, run AB + BA, use multiple judges, calibrate against human labels.
To production users	Red-team first (safety + capability + robustness). Fix seeds, log API versions, run prompt sensitivity.

> **Tip**
>
> Evaluation is not a one-time gate — it’s a continuous practice. The field is evolving rapidly; contamination detection and LLM-as-a-judge are active research areas.

Looking ahead: Evaluating agentic systems

Everything we covered today evaluates a **single model turn**. But LLM agents act over multiple steps — browsing, coding, calling APIs, revising. This breaks our evaluation assumptions:

Single-Turn Eval	→	**Agentic Eval**
Input → Output → Score		Goal → *Trajectory* (plan, tool calls, revisions) → Outcome
One correct answer (or small set)		Many valid trajectories to the same goal
Errors are local (one bad generation)		Errors **compound** — one bad step derails the whole plan
Static benchmark dataset		Live environments (web, code execution, databases)

New dimensions in agentic evaluation

Task completion rate

Did the agent achieve the goal? Binary, but deceptively hard — partial credit for 80% of a code fix?

Trajectory efficiency

How many steps / tokens / tool calls? An agent that solves a task in 3 steps vs. 30 matters for cost and latency.

Recovery and self-correction

Does the agent detect its own errors and backtrack? Resilience matters more than first-try accuracy.

Safety under autonomy

An agent with tool access can *act* on the world — deleting files, sending emails, executing code. The blast radius of failures is much larger.

- **Open question:** How do you benchmark a system whose environment changes with every action it takes?
- Today's metrics (BLEU, human pairwise) don't capture any of this — new evaluation paradigms are emerging fast