Evaluating Generative AI & Agentic Systems

From Human Annotation to Automated Judges

Arian Pasquali | Orq.ai

ECIR 2026, Text2Story Workshop - Delft, March 29

The Evaluation Challenge

Why is this hard?

The Evaluation Problem

An LLM will always answer the same question **in different ways**.

Prompt	Response A	Response B	Response C
"*Summarize this news article about the flooding in Valencia*"	Concise, factual, 2 sentences	Detailed, emotional tone, 5 sentences	Bullet points, includes statistics

All three could be **correct**. Or **none** of them.

The subjective nature of generative answers makes evaluation fundamentally different from classification.

Running Example: Story Summarization

Source narrative (a news story):

> *"On October 29, 2024, the Spanish region of Valencia experienced catastrophic flooding caused by the DANA weather phenomenon. Over 200 people lost their lives, making it one of Spain's deadliest natural disasters in decades. Emergency services were overwhelmed, and thousands were displaced from their homes..."*

We'll use this example throughout the talk to ground every concept.

Three candidate summaries to evaluate:

- **Summary A** -- Accurate, concise, well-structured
- **Summary B** -- Too long, buries the key facts, editorializes
- **Summary C** -- Short but hallucinates a death toll of "over 500"

What Makes a "Good" Answer?

Ask yourself -- is it:

- **Too long?** Too short?
- **Hallucinating** facts not in the source?
- Using **correct citations** and references?
- In the **appropriate tone** for the audience?
- **Faithful** to the source material?
- Does it preserve the **causal chain** of events?
- Are **temporal relations** maintained (what happened first)?
- Are the **key actors and their roles** correctly identified?

These criteria feel obvious, but they are **surprisingly hard to agree on**.

> *"It is impossible to completely determine evaluation criteria prior to human judging of LLM outputs."*
> *-- Shankar et al., "Who Validates the Validators?", CHI 2024*

The Evolution of Annotation

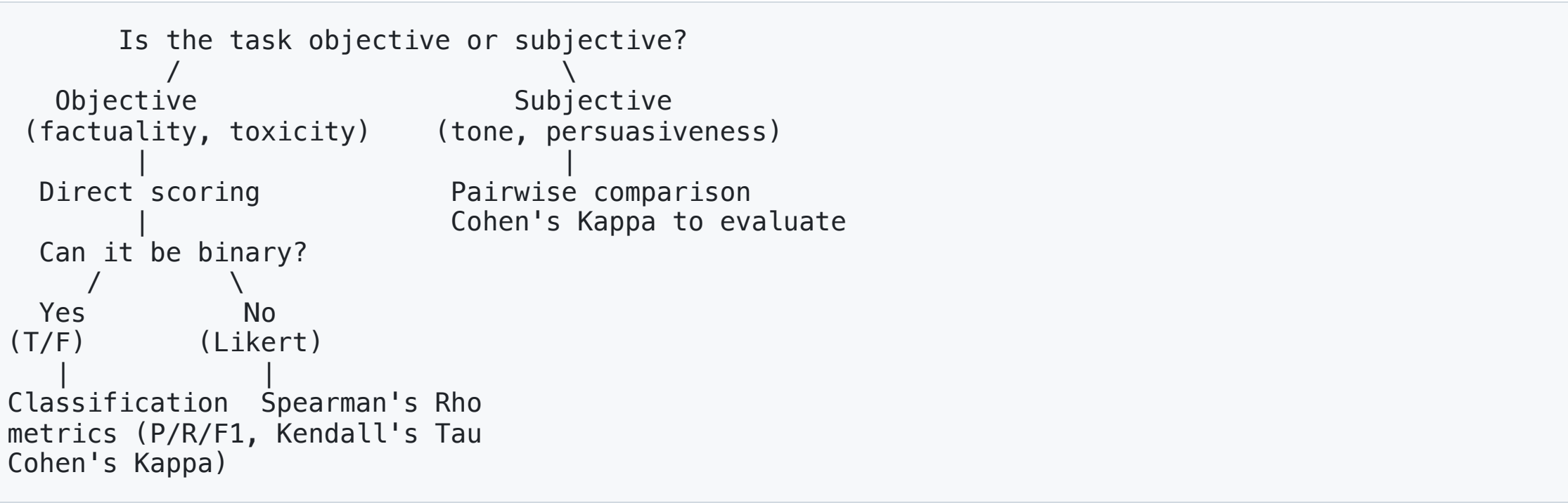
Era	Labels Needed	Approach
Traditional ML	**Thousands**	Fully supervised, large annotated corpora
Transfer Learning	**Hundreds**	Fine-tune pre-trained models
Large Language Models	**Few** (even zero)	Few-shot prompting, in-context learning

The cost of getting started has **dramatically decreased**.

But the need for evaluation has **not**.

Even with zero-shot models, we still need to know: **is the output good?**

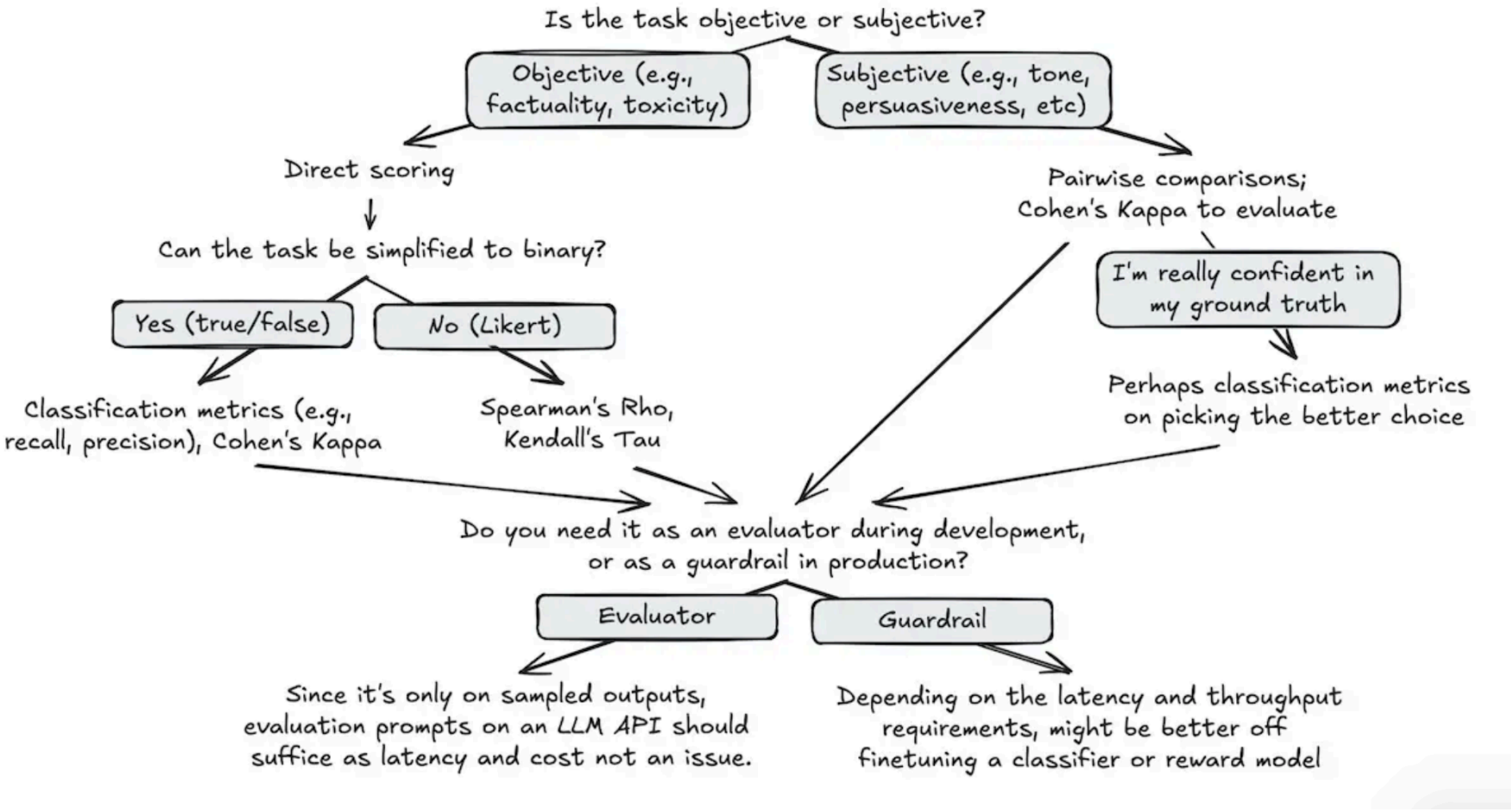
Evaluation Approaches: A Decision Framework



Three approaches -- they are NOT interchangeable:

Approach	Best For	Example
Direct scoring	Objective tasks (factuality, toxicity)	"Is this summary faithful?"
Pairwise comparison	Subjective tasks (tone, coherence)	"Which summary is better, A or B?"
Reference-based	Gold standard comparison	"How close to the expert summary?"

Evaluation Approaches: Decision Tree



Source: Yan, 2024

Choosing Your Metrics

Classification metrics (for binary tasks -- preferred):

- Precision, Recall, F1
- Straightforward to interpret and act on
- "What's the evaluator's recall on bad responses?"

Correlation metrics (for ordinal/ranked data):

Metric	Best For	Range	Notes
Cohen's kappa	Categorical agreement	-1 to 1	Adjusts for chance; conservative
Kendall's tau	Ordinal rankings	-1 to 1	Robust to outliers
Spearman's rho	Rank correlation	-1 to 1	Sensitive to magnitude

Practical advice: where possible, make your evaluators return **binary outputs**. This improves model performance and makes metrics directly actionable.

Capability vs. Regression Evals

Two different goals -- separate them early:

	Capability Evals	Regression Evals
Question	"*What can it do?*"	"*Does it still work?*"
Goal	Drive innovation	Ensure stability
Pass rate	Start with low pass rate -- gives you a hill to climb	Should be ~100% before shipping
Focus	Hard tasks, edge cases	Core functionality, known behaviors

Start with **capability evals** to measure progress on hard tasks.

Promote high-pass-rate capability evals into the **regression suite** over time.

Evaluators vs. Guardrails

Two different tools for two different jobs:

	Evaluator	Guardrail
When	After generation (async)	During generation (inline)
Speed	Seconds to minutes	Milliseconds
Purpose	Quality assessment	Safety filter
Example	"Was this summary faithful?"	"Does this contain PII?"
Cost tolerance	Higher (sampled)	Must be cheap (every request)

Don't confuse them. An evaluator that runs on sampled outputs can afford to be expensive. A guardrail that runs on every request must be fast and cheap.

Start With Humans

Before you automate anything

Ground Truth: The Ugly Truth

The majority of projects **do not have usable evaluation data**.

What NOT to do:

> *"Hey ChatGPT, generate 10 ground truth samples"*

- No diversity
- Not grounded in any existing knowledge base
- Useless for real evaluation

What to do instead:

1. **Option A**: Build your system and start collecting production logs as data
2. **Option B**: Find every existing data signal, take the little data available, and expand systematically

Building Good Evaluation Datasets

Start from existing knowledge bases, then expand systematically:

1. For each document or piece of content, generate pairs of questions and answers **that can only be answered using that content**
2. Add **diversity dimensions**:
    - Difficulty (easy, medium, hard)
    - Topics (different subject areas)
    - User types (expert, novice, non-native speaker)
    - Scenarios (straightforward, ambiguous, adversarial)
3. Combine dimensions to create a diverse synthetic dataset
4. **Have a domain expert validate** the usable pairs

> *Quality over quantity: **20-50 hand-reviewed examples** will outperform hundreds of unverified synthetic ones.*

Annotation Guidelines

Back to our story summarization example. How does a human evaluator decide?

The process:

1. Define what makes a summary acceptable
2. Have **multiple annotators** evaluate the same summaries
3. Measure **inter-rater agreement**
4. If agreement is low --> your guidelines are unclear --> **iterate**
5. Keep iterating until annotators agree
6. **A few dozen examples** might be enough

	Summary A	**Summary B**	**Summary C**
Annotator 1	Pass	Fail	Fail
Annotator 2	Pass	Pass	Fail
Annotator 3	Pass	Fail	Fail

Summary A: unanimous pass. Summary C: unanimous fail. Summary B: **disagreement** --> refine guidelines.

The Mechanical Turk Case

How do we trust **outsourced** human annotation?

Remember Amazon Mechanical Turk? We solved this problem:

1. **Multiple annotators** to reduce individual bias
2. **Sufficient samples** for statistical significance
3. **Split the dataset**:
    - A small portion we annotate ourselves (the "gold set")
    - Use this to test if outsourced annotators follow the guidelines
    - Once they pass the baseline --> trust their annotations for the rest

This is exactly how we should think about LLM annotators.

An LLM judge is just another annotator -- treat it with the same rigor you'd apply to a Mechanical Turk worker.

The System Prompt Being Evaluated

First, this is the prompt we're building and want to evaluate -- the **summarizer**:

```
You are a news summarizer. Given a news article,
produce a concise summary that:
- Captures the key facts (who, what, when, where)
- Preserves exact numbers and statistics
- Uses neutral, journalistic tone
- Is 2-3 sentences maximum
```

Input: the Valencia flooding article

Output (Summary C): "*In October 2024, catastrophic flooding in Valencia, Spain, killed over 500 people in one of the country's worst natural disasters...*"

This summary **hallucinated** the death toll (200 → 500). How do we catch this systematically?

The LLM-as-a-Judge Prompt

Now we write a **second prompt** -- the judge that evaluates the summarizer's output:

```
You are evaluating a news story summary for
faithfulness to the source article.

## Source article
{article_text}

## Summary to evaluate
{summary_text}

## Task
Return PASS or FAIL with a detailed critique.

PASS: the summary only contains information present
in or directly inferable from the source.
FAIL: the summary contains fabricated or unsupported
claims.
```

> *"The real business value comes from carefully examining your data, not from the LLM judge itself."*
>
> *-- Husain, 2024*

The Judge's Output

What the LLM judge returns for Summary C:

```
Verdict: FAIL

Critique: The summary states "killed over 500 people"
but the source article reports "over 200 people lost
their lives." This is a factual fabrication -- the
death toll has been inflated by more than double.
The remaining facts (date, location, cause) are
accurate. The tone and length are appropriate.
The hallucinated statistic is a critical failure.
```

This **critique** is the key -- it explains *why* the summary failed.

Later, this critique becomes the **feedback signal** for prompt optimization.

Why Binary > Likert Scales

Don't do this:

> *"Rate the summary on a scale of 1-5 for faithfulness, coherence, and relevance"*

Do this instead:

> *"PASS or FAIL? Explain why."*

	Binary (Pass/Fail)	Likert (1-5 Scale)
Clarity	Forces a decision	Allows fence-sitting
Speed	Faster annotation	Slower, more cognitive load
Reliability	Higher agreement	Lower agreement
Metrics	Precision, Recall, F1	Correlation coefficients
Actionability	Direct: "fix these failures"	Vague: "improve 3.2 to 3.8"

The **critique** captures the nuance that scales only pretend to capture.

References: DoorDash (binary content moderation); Touvron et al., "Llama 2: Open Foundation and Fine-Tuned Chat Models", 2023 (binary preference collection is "much faster")

Trust the Machine Annotator

Aligning judges

Criteria Drift: You Can't Pre-Define Everything

The most important finding from the evaluation literature:

> *"It is impossible to completely determine evaluation criteria prior to human judging of LLM outputs."*
> *-- Shankar et al., "Who Validates the Validators?", CHI 2024*

The circular dependency:

```
Need criteria to grade outputs
        |                    ^
        v                    |
Grading outputs reveals new criteria
```

Some criteria are **dependent on the specific outputs observed**, not definable a priori.

Implication: Evaluation is **iterative**, not a one-time setup. Your criteria will evolve as you see more data.

The Critique Shadowing Process

Hamel Husain's 5-step process for building aligned LLM judges:

Step 1: Find the **domain expert** (1-2 people whose judgment is ground truth)

Step 2: Create a **diverse dataset** (features x scenarios x personas)

Step 3: Domain expert makes **pass/fail judgments + detailed critiques**

- Critiques must be detailed enough for a new employee to understand

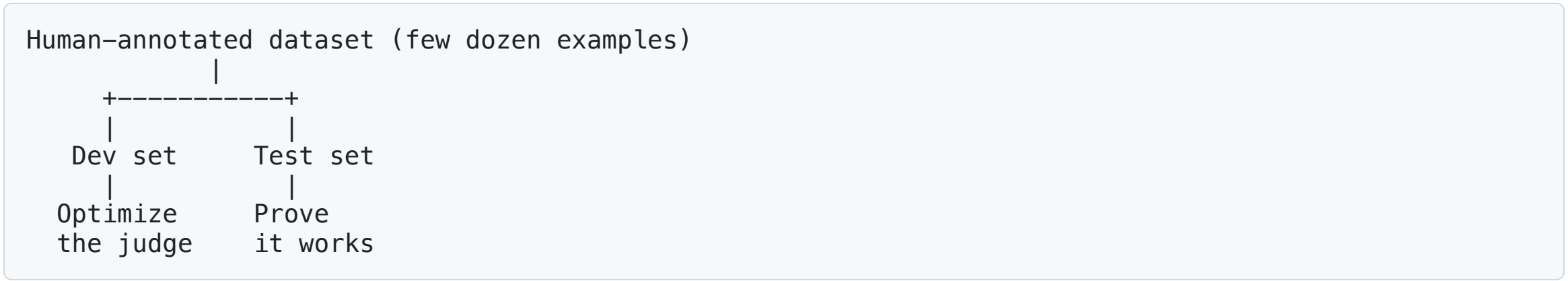
Step 4: **Fix obvious system errors** discovered during review

Step 5: Build the LLM judge **iteratively**:

1. Use expert critiques as few-shot examples in the judge prompt
2. LLM generates its own judgments on new items
3. Expert independently judges the same items
4. Compare, identify disagreement patterns, refine prompt
5. Repeat until **>90% agreement**

Alignment Workflow: The Split

Treat your LLM judge like a model you need to validate:



On the dev set: iterate on your judge prompt until agreement is high

On the test set: validate that the judge generalizes (don't peek!)

This is the **same train/test split** from classical ML -- applied to evaluation.

Target: LLM-human agreement should match **human-human agreement**.

What "Aligned" Looks Like

Our story summarization example -- after alignment:

	Human: Pass	**Human: Fail**
LLM Judge: Pass	TP = 18	FP = 2
LLM Judge: Fail	FN = 1	TN = 9

Precision: 18/20 = 90% (when the judge says pass, it's usually right)
Recall: 18/19 = 95% (the judge catches most good summaries)
Cohen's kappa: 0.82 (strong agreement)

Once aligned --> **trust this LLM + prompt to annotate at scale**.

Now the annotation guideline you iterated on **becomes your production evaluation prompt**.

Mitigating Bias: Panel of Judges

Same principle as multiple human annotators: reduce bias by adding more perspectives.

LLM-as-a-Jury:

- 3 models from **different providers** (e.g., Claude, GPT, Gemini)
- Same evaluation prompt
- **Majority vote** (for binary) or **average** (for numerical)

Why different providers?

- Training data bias differs
- RLHF preferences differ
- Perplexity patterns differ

Anti-pattern: Never evaluate a model's output using the **same model**.
Using GPT-4 to judge GPT-4's responses introduces self-preference bias.

> *Verga et al., "Replacing Judges with Juries: Evaluating LLM Generations with a Panel of Diverse Models", 2024: an ensemble of **3 smaller judges outperformed a single GPT-4** evaluator.*

Statistical Rigor

How much data do you actually need?

Purpose	Minimum	Recommended
Initial alignment (dev set)	20-30	50
Validation (test set)	20-30	50
Per data segment	50	~200
Confidence in improvement	Use confidence intervals	pass@k, pass^k

Dealing with non-determinism:

- **pass@k**: at least 1 of k attempts succeeds (capability)
- **pass^k**: all k attempts succeed (reliability)

Run multiple trials per task. Compute confidence intervals **before** declaring improvement.

AlignEval: Putting It Into Practice

Eugene Yan's tool implements these principles (Yan, 2024):

Step	What	Details
1. Upload	Data (input, output, label)	CSV format
2. Label	Binary pass/fail only	50-100 labels recommended
3. Evaluate	LLM judge vs. human labels	Recall, Precision, F1, Cohen's kappa
4. Optimize	Auto-improve judge prompt	Dev/test split, maximize F1

From Iteration to Automation

Scaling up

Evaluation Levels for Agents

Not all evaluations operate at the same granularity:

Level	Granularity	Question	Example
Single-step	Individual tool call	Did it choose the right tool?	"Did the agent call the search API?"
Full-turn	Entire agent turn	Was the final response correct and the path reasonable?	"Did it summarize the article faithfully?"
Multi-turn	Conversation across turns	Does it maintain context and coherence?	"Did it remember the user's constraints?"

Start with full-turn evals -- they give the most signal.

Grade across three dimensions: **final response** (correct?), **trajectory** (valid path?), and **state changes** (did it actually do the right thing, not just say it did?).

From Manual to Automated: The Eval Pipeline

Three modes of evaluation -- use all three:

Type	When	Purpose	Cost
Offline	Pre-deployment	Test changes before shipping	Medium
Online	Continuous	Catch failures in real traffic	Higher
Ad-hoc	Exploratory	Discover unanticipated patterns	Variable

CI/CD integration:

```
Code/prompt change
    --> Offline evals (code-based graders, cheap, every commit)
    --> Preview deployment
    --> Online evals (LLM-as-judge, expensive, sampled)
    --> Production promotion
```

Use **cheap code-based graders** in CI for every commit.

Reserve **expensive LLM judges** for preview/production.

Error Analysis: Where 60-80% of Effort Should Go

> *"Teams should spend 60-80% of eval effort on error analysis."*
> *-- Husain, 2024*

Open-coding methodology:

1. Gather failure traces
2. Review with domain expert **without pre-categorization**
3. Build a failure taxonomy from what you observe
4. Iterate until no new categories emerge

Root cause framework -- every failure is one of:

- **Prompt problem** --> fix the prompt
- **Tool design problem** --> fix the tool interface
- **Model limitation** --> switch models or add constraints
- **Unknown** --> needs more analysis

Real example: Witan Labs found a single extraction bug moved their benchmark from **50% to 73%**. Infrastructure, not reasoning, was the culprit.

Two Types of Issues and System Improvements

	Lack of Context	Lack of Capabilities
Origin	Limited content in the knowledge base or context	System's functional abilities; context exists but model is not capable to understand
How to fix	Expand the context, corpus, improve ingestion and data connectors in case of knowledge base, create focused sub-systems	Improve data understanding, extract additional metadata, add new search features

Categorizing failures this way helps **prioritize** what to fix:

1. Identify gaps in context and functionality
2. Develop specialized sub-systems for specific use cases
3. Improve user experience across query types

Prompt Optimization: Manual Refinement

The annotation guideline you iterated on **becomes** your LLM judge prompt.

Iterating on this prompt = iterating on evaluation quality.

Manual optimization (Husain, 2024):

1. Run the judge on new examples
2. Domain expert independently judges the same examples
3. Find **disagreement patterns** (not individual cases)
4. Refine the prompt to address patterns
5. Repeat

Prompt Optimization: Decomposition

Instead of 1 "correctness" evaluator, use **specialized evaluators**:

Evaluator	Dimension
Content accuracy	Are facts correct?
Faithfulness	Is it grounded in the source?
Temporal coherence	Are events in the right order?
Causal structure	Is the cause-effect chain preserved?
Completeness	Are key actors and events covered?

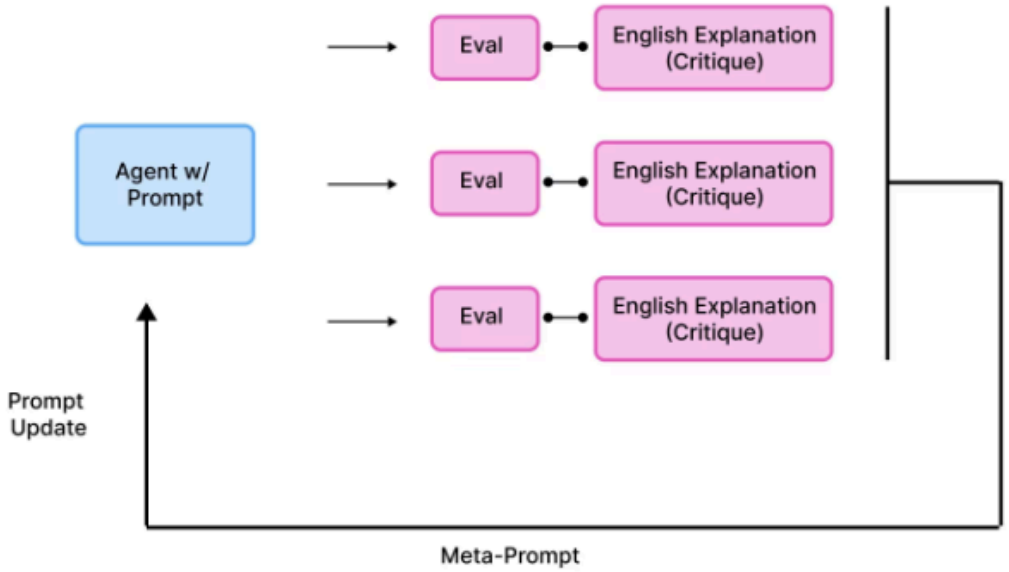
Each with dimension-appropriate thresholds.

Decompose complex evaluation into **simpler subtasks** -- the same principle as in traditional NLP.

Prompt Learning: RL with English Feedback

What if we could **automate** prompt optimization?

Core idea: Replace gradient-based updates with **natural language feedback**. This is **reinforcement learning** -- but the reward signal is **language**, not numbers.



Each eval produces an **English critique**. The **metaprompt** reads all critiques and updates the agent's prompt. One loop = one improvement. Repeat until convergence.

Prompt Learning: The Metaprompt in Action

The metaprompt reads critiques and decides: **ADD**, **MERGE**, **REWRITE**, or **EXPIRE** rules.

Concrete example — critique: "*death toll inflated from 200 to 500*"

Metaprompt decides: `REWRITE` the statistics rule

Before:

```
- Preserves exact numbers and statistics
```

After:

```
- Preserves exact numbers and statistics. NEVER
  alter, round, or fabricate quantitative data.
  If unsure about a number, omit it rather than
  guess.
```

One critique → one targeted rule improvement. Repeat across many failures → a **self-improving prompt**.

Prompt Learning: Results

Instruction learning (learning unknown business rules from feedback only):

Rules to Learn	Unoptimized	1 Loop	5 Loops
10	0%	84%	**100%**
50	0%	66%	82%
100	0%	42%	67%

Big Bench Hard: ~10% improvement on a highly saturated benchmark (GPT-4.1, no handcrafted prompts)

Speed: 10-100x faster than traditional prompt optimization

Counter-intuitive dynamics:

- Training scores can **drop** after optimization
- Test results sometimes **exceed** training results
- Running too many loops can degrade performance

This is not gradient descent -- it's a best-effort attempt to fix problems via language.

The Future: Continuous Self-Improving Evaluation

The production flywheel:

```
Production failures
       |
       v
  Error analysis --> New dataset examples
       |
       v
  Eval improvement --> Better judge prompts
       |
       v
  Agent improvement --> Better outputs
       |
       v
  Fewer failures (repeat)
```

Human effort **naturally decreases** as systems improve.

But it **never reaches zero**.

> *"Never eliminate humans completely -- LLMs must align to something, usually a human."*
> *-- Hamel Husain*

The New Role of the Annotator

What changes

Key Takeaways

1. Start simple: manually review **20-50 examples** before building any infrastructure

2. Use **binary pass/fail + written critiques**, not complex scoring rubrics

3. Criteria drift is real -- evaluation is iterative, not a one-time setup
(Shankar et al., 2024)

4. Align LLM judges to humans **the same way you'd validate outsourced annotators**
(split dataset, gold set, measure agreement)

5. Prompt learning turns evaluation feedback into an optimization signal
(RL with English feedback)

6. Turn AI systems into **learning systems** -- process first, tools later

The Annotator's New Role

LLMs don't replace annotators -- they **amplify** them.

Before	After
Label thousands of examples	Label **dozens** to calibrate judges
Annotate every data point	Validate the **evaluation system**
Follow rigid guidelines	**Co-create** evolving criteria
Scale through more annotators	Scale through **aligned LLM judges**

The human-in-the-loop shifts from labeling everything to validating the judge.

Fewer labels. Higher impact.

Your expertise becomes **more valuable**, not less.

Thank You

Arian Pasquali | Research Engineer
Orq.ai | Amsterdam
arian.pasquali@orq.ai

Open for collaborations

linkedin.com/in/arianpasquali

References

- Shankar, S. et al. (2024). "*Who Validates the Validators?*" CHI 2024. arXiv:2404.12272
- Verga, P. et al. (2024). "*Replacing Judges with Juries.*" arXiv:2404.18796
- Touvron, H. et al. (2023). "*Llama 2: Open Foundation and Fine-Tuned Chat Models.*" arXiv:2307.09288
- Husain, H. (2024). "*Creating an LLM-as-a-Judge Evaluation System.*" hamel.dev/blog/posts/llm-judge
- Yan, E. (2024). "*AlignEval.*" eugeneyan.com/writing/aligneval
- "*Prompt Learning: Using English Feedback to Optimize LLM Systems.*" (2026). arize.com/blog/prompt-learning-using-english-feedback-to-optimize-llm-systems
- Orq.ai (2025). "*A Comprehensive Guide to Evaluating Multi-Agent LLM Systems.*" orq.ai/blog/multi-agent-llm-eval-system
- Orq.ai (2025). "*Agent Evaluation in 2025: Complete Guide.*" orq.ai/blog/agent-evaluation
- Orq.ai (2024). "*Mastering RAG Evaluation: Best Practices & Tools.*" orq.ai/blog/rag-evaluation